
A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory
Las Palmas de Gran Canaria, Spain, February 2001
Revised Papers

A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory
Las Palmas de Gran Canaria, Spain, February 19-23, 2001
Revised Papers
Preface

The concept of CAST as Computer Aided Systems Theory, was introduced by F. Pichler in the late 1980s to include those computer theoretical and practical developments as tools to solve problems in System Science. It was considered as the third component (the other two being CAD and CAM) necessary to build the path from Computer and Systems Sciences to practical developments in Science and Engineering.

The University of Linz organized the first CAST workshop in April 1988, which demonstrated the acceptance of the concepts by the scientific and technical community. Next, the University of Las Palmas de Gran Canaria joined the University of Linz to organize the first international meeting on CAST, (Las Palmas, February 1989), under the name EUROCAST’89. This was a very successful gathering of systems theorists, computer scientists, and engineers from most European countries, North America, and Japan.

It was agreed that EUROCAST international conferences would be organized every two years, alternating between Las Palmas de Gran Canaria and a continental European location. Thus, successive EUROCAST meetings have taken place in Krems (1991), Las Palmas (1993), Innsbruck (1995), Las Palmas (1997), and Vienna (1999), in addition to an extra-European CAST Conference in Ottawa in 1994. Selected papers from those meetings were published by Springer-Verlag in the Lecture Notes in Computer Science series, as volumes 410, 585, 763, 1030, 1333, and 1728, and in several special issues of Cybernetics and Systems: an International Journal. EUROCAST and CAST meetings are definitely consolidated, as is demonstrated by the number and quality of the contributions over the years.

EUROCAST 2001 (Las Palmas, February 2001) presented a new approach to the conferences, which will be adopted for future meetings. Besides the classical core on generic CAST (Chaired by Pichler and Moreno-Díaz), in the form of a CAST workshop, there were three other specialized workshops devoted to Computer Algebra and Automated Theorem Proving (CAL, chaired by Buchberger from Linz), to Functional Programming and \( \lambda \) Calculus (FP, chaired by Freire from La Coruña), and to Abstract State Machines (ASM, chaired by Glässer from Paderborn and Börger from Pisa),

This volume contains selected full papers from the CAST, CAL, and FP workshops and two invited lectures. Papers form the ASM workshop will be published in a separate volume.

The editors would like to thank all contributors for their quickness in providing their material in hard and electronic forms. Special thanks are due to Dr. Alexis Quesada, from the Institute of Cybernetics of the University of Las Palmas, for his great help in the preparation of the volume, and to the Staff of Springer-Verlag, Heidelberg for their valuable support.

July 2001
Roberto Moreno-Díaz
Bruno Buchberger
José-Luis Freire
Table of Contents

Invited Lectures

Gröbner Bases: A Short Introduction for Systems Theorists.................................1
   B. Buchberger

Design for Reuse via Structuring Techniques for ASMs.........................................20
   E. Börger

1 Computer Aided Systems Theory

1.1 Mathematical and Logic Formalisms

On CAST.FSM Computation of Hierarchical Multi-layer Networks of Automata.................................................................36
   M. Affenzeller, F. Pichler, R. Mittelmann

A Holarchical Organized Design Assistant for Hierarchical Decompositions............................................................45
   S. Meine

Conflict Management in an Intelligent Multiagent Robotics System Based on FSM Approach................................................52
   W. Jacak, K. Pröll, S. Dreiseitl

Computer-Aided Simulations of Gaussian Processes and Related Asymptotic Properties..................................................67
   E. Di Nardo, A.G. Nobile, E. Pirozzi, L.M. Ricciardi

Hybrid Systems’ Properties – Classification and Relation to Computer Science...................................................................79
   T. Stauner

Assertion-Based Analysis of Hybrid Systems with PVS........................................94
   E. Ábrahám-Mumm, U. Hannemann, M. Steffen

Algebraic Description of Physical Systems........................................................................110
   H. von Issendorff
# 1.2 Information and Decision

Information Structure and Logical Design of Social Rules........................................125  
*T. Shimogawa*

Towards a Generic Model of Information Processing Systems.................................139  
*G. Németh*

Evolutionary Information and Decision Support Systems:  
An Integration Based on Ontologies.........................................................................146  
*M.V. Hurtado, J. Parets*

A Proposal for Project Management Using CommonKADS........................................160  
*I. del Aguila, S. Túnez, J. Cañadas, A. Bosch, R. Marín*

A General Planning Method for Allocation of Human Resource Groups...............172  
*M. Mauerkirchner*

Optimizing Organizational Structure: A Hausdorff Benchmark for  
Complex Social Systems.........................................................................................182  
*M. Schwaninger*

A Planning under Uncertainty Model........................................................................196  
*E. Paniagua-Aris, J.T. Palma-Méndez, F. Martin-Rubio*

# 1.3 Complexity

Knowledge Based Modeling of the Design Process as a Base of Design  
Tools. Application to the Development of Agricultural Structures........................209  
*J.F. Bienvenido, I.M. Flores-Parra, R. Guirado, R.L. Marin*

Complexity Analysis of Systems from a Functional and Technical  
Viewpoint................................................................................................................223  
*S. Dierndörfer, R. Scheidl*

A Complexity-Theoretic Approach to the Design of Good Measures  
of Cryptographic Strength.......................................................................................233  
*J.D. González Cobas, J.A. López Brugos*

Characterizing the Software Development Process: A New Approach  
Based on Kolmogorov Complexity..........................................................................242  
*C.A.P. Campani, P.B. Menezes*

Application of Signed Kolmogorov Hashes to Provide Integrity and  
Authenticity in Web-Based Software Distribution...............................................257  
*J. Scharinger*
Toward a Formalisation of Evolutionary Hypermedia Systems Based on System Theory...............................................................................................................270
  L. García-Cabrera, M.J. Rodríguez-Fórtiz, J. Parets-Llorca

Optimization Problems Categories................................................................................285
  L.A. dos Santos Leal, P.B. Menezes, D. Moraes Claudio,
  L. Vieira Toscani

1.4 CAST in Neural-like Computation. Applications

  Systems Methodology for Classical Neural Nets at Various Levels.........................300
  R. Moreno-Díaz, G. de Blasio

  Biologically Based CAST-mechanism for Visual Motion Analysis..............................316
  A. Quesada-Arencibia, R. Moreno-Díaz, jr., M. Aleman-Flores

  Towards a Model of Volume Transmission in Biological and Artificial
  Neural Networks: A CAST Approach........................................................................328
  C.P. Suarez Araujo, P. Fernandez Lopez, P. Garcia Baez

  An Online Information Retrieval Systems by Means of Artificial
  Neural Networks........................................................................................................343
  M.E. Zorrilla, J.L. Crespo, E. Mora

  Kinematic in Robotics by the Morphogenetic Neuron..............................................352
  G. Resconi, A. Borboni, R. Faglia, M. Tiboni

  Analysis and Verification Queries over Object-Oriented Petri Nets.........................369
  M. Češka, V. Janoušek, T. Vojnar

1.5 CAST in Automation and Control

  Integrated Design and Implementation of Digital Controllers...............................385
  P. Albertos, A. Crespo

  Symbolic Computation Tools for Dynamical Nonlinear Control Systems................393
  J. Rodríguez-Millán

  Integrated Symbolic-Graphic-Numeric Analysis and Design in
  Nonlinear Control through Notebooks in Mathematica............................................405
  J. Rodríguez-Millán

  Towards a Computer-Aided Design of Reactive Systems.......................................421
  M. Frappier, R. St-Denis
<table>
<thead>
<tr>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributions to the Control and Stabilization of the Pole-Cart System</td>
<td>437</td>
</tr>
<tr>
<td>A. Raya, D. Maravall</td>
<td></td>
</tr>
<tr>
<td>A Qualitative System as a Frame to Control Unmanned Vehicles</td>
<td>450</td>
</tr>
<tr>
<td>R. García Rosa, T. de Pedro</td>
<td></td>
</tr>
<tr>
<td>Hyper-Automation System Applied to Geometry Demonstration Environment</td>
<td>457</td>
</tr>
<tr>
<td>2 Computer Algebra and Automated Theorem Proving</td>
<td></td>
</tr>
<tr>
<td>The PCS Prover in THEOREMA</td>
<td>469</td>
</tr>
<tr>
<td>B. Buchberger</td>
<td></td>
</tr>
<tr>
<td>Higher-Order Lazy Narrowing Calculus: A Solver for Higher-Order Equations</td>
<td>479</td>
</tr>
<tr>
<td>T. Ida, M. Marin, T. Suzuki</td>
<td></td>
</tr>
<tr>
<td>Classifying Isomorphic Residue Classes</td>
<td>494</td>
</tr>
<tr>
<td>A. Meier, M. Pollet, V. Sorge</td>
<td></td>
</tr>
<tr>
<td>CAL: A Computer Assisted Learning System for Computation and Logic</td>
<td>509</td>
</tr>
<tr>
<td>M. Sato, Y. Kameyama, T. Izumi</td>
<td></td>
</tr>
<tr>
<td>A Set Theory Prover Within Theorema</td>
<td>525</td>
</tr>
<tr>
<td>W. Windsteiger</td>
<td></td>
</tr>
<tr>
<td>3 Functional Programming and λ Calculus</td>
<td></td>
</tr>
<tr>
<td>The Tertiary Level in a Functional Cluster-Based Hierarchical VoD Server</td>
<td>540</td>
</tr>
<tr>
<td>M. Barreiro, V.M. Gulías, J.J. Sánchez, S. Jorge</td>
<td></td>
</tr>
<tr>
<td>Hidden Specification of a Functional System</td>
<td>555</td>
</tr>
<tr>
<td>C. Domínguez, L. Lambán, V. Pascual, J. Rubio</td>
<td></td>
</tr>
<tr>
<td>Distributed Execution of Functional Programs Using the JVM</td>
<td>570</td>
</tr>
<tr>
<td>A.R. Du Bois, A.C. da Rocha Costa</td>
<td></td>
</tr>
<tr>
<td>Fusion in Coq</td>
<td>583</td>
</tr>
<tr>
<td>J.L. Freire Nistal, J.E. Freire Brañas, A. Blanco Ferro,</td>
<td></td>
</tr>
<tr>
<td>J.J. Sánchez Penas</td>
<td></td>
</tr>
</tbody>
</table>
A Language Prototyping Tool Based on Semantic Building Blocks.........................597
    J.E. Labra Gayo, J.M. Cueva Lovelle, M.C. Luengo Diez,
    B.M. González Rodríguez

Verifying an Applicative ATP Using Multiset Relations.................................612
    F.J. Martín-Mateos, J.A. Alonso, M.J. Hidalgo, J.L. Ruiz-Reina

Adding Traces to a Lazy Monadic Evaluator.....................................................627
    C. Pareja, R. Peña, F. Rubio, C. Segura

Functional Vs Object-Oriented Distributed Languages......................................642

Persistent Haskell.................................................................................................657
    J. Quintela, J.J. Sánchez

Author Index.......................................................................................................669
Gröbner Bases: 
A Short Introduction for Systems Theorists

Bruno Buchberger
Research Institute for Symbolic Computation
University of Linz, A4232 Schloss Hagenberg, Austria
Buchberger@RISC.Uni−Linz.ac.at

Abstract. In this paper, we give a brief overview on Gröbner bases theory, addressed to novices without prior knowledge in the field. After explaining the general strategy for solving problems via the Gröbner approach, we develop the concept of Gröbner bases by studying uniqueness of polynomial division ("reduction"). For explicitly constructing Gröbner bases, the crucial notion of S-polynomials is introduced, leading to the complete algorithmic solution of the construction problem. The algorithm is applied to examples from polynomial equation solving and algebraic relations. After a short discussion of complexity issues, we conclude the paper with some historical remarks and references.

1 Motivation for Systems Theorists

Originally, the method of Gröbner bases was introduced in [3, 4] for the algorithmic solution of some of the fundamental problems in commutative algebra (polynomial ideal theory, algebraic geometry). In 1985, on the invitation of N. K. Bose, I wrote a survey on the Gröbner bases method for his book on n-dimensional systems theory, see [7]. Since then quite some applications of the Gröbner bases method have been found in systems theory. Soon, a special issue of the Journal of Multidimensional Systems and Signal Processing will appear that is entirely devoted to this topic, see [11]. Reviewing the recent literature on the subject, one detects that more and more problems in systems theory turn out to be solvable by the Gröbner bases method:

- factorization of multivariate polynomial matrices,
- solvability test and solution construction of unilateral and bilateral polynomial matrix equations, Bezout identity,
- design of FIR / IIR multidimensional filter banks,
• stabilizability / detectability test and synthesis of feedback stabilizing compensator / asymptotic observer,
• synthesis of deadbeat or asymptotic tracking controller / regulator,
• constructive solution to the $n$D polynomial matrix completion problem,
• computation of minimal left annihilators / minimal right annihilators,
• elimination of variables for latent variable representation of a behaviour,
• computation of controllable part; controllability test,
• observability test,
• computation of transfer matrix and "minimal realization",
• solution of the Cauchy problem for discrete systems,
• testing for inclusion; addition of behaviors,
• test zero / weak zero / minor primeness,
• finite dimensionality test,
• computation of sets of poles and zeros; polar decomposition,
• achievability by regular interconnection,
• computation of structure indices.

In [11], I gave the references to these applications and I also presented an easy introduction to the theory of Gröbner bases by giving a couple of worked-out examples. In this paper, I will give an introduction to Gröbner bases in the style of a flyer for promotion that just answers a couple of immediate questions on the theory for newcomers. Thus, [11] and the present paper are complementary and, together, they may provide a quick and easy introduction to Gröbner bases theory, while [7] provides a quick guide to the application of the method to fundamental problems in commutative

2 Why is Gröbner Bases Theory Attractive?

Gröbner bases theory is attractive because

• the main problem solved by the theory can be explained in five minutes (if one knows the operations of addition and multiplication on polynomials),
• the algorithm that solves the problem can be learned in fifteen minutes (if one knows the operations of addition and multiplication on polynomials),
the theorem on which the algorithm is based is nontrivial to (invent and to) prove,

many problems in seemingly quite different areas of mathematics can be reduced to the problem of computing Gröbner bases.

3 What is the Purpose of Gröbner Bases Theory?

The method (theory plus algorithms) of Gröbner bases provides a uniform approach to solving a wide range of problems expressed in terms of sets of multivariate polynomials. Areas in which the method of Gröbner bases has been applied successfully are:

- algebraic geometry, commutative algebra, polynomial ideal theory,
- invariant theory,
- automated geometrical theorem proving,
- coding theory,
- integer programming,
- partial differential equations,
- hypergeometric functions,
- symbolic summation,
- statistics,
- non-commutative algebra,
- numerics (e.g. wavelets construction), and
- systems theory.

The book [9] includes surveys on the application of the Gröbner bases method for most of the above areas. In commutative algebra, the list of problems that can be attacked by the Gröbner bases approach includes the following:

- solvability and solving of algebraic systems of equations,
- ideal and radical membership decision,
- effective computation in residue class rings modulo polynomial ideals,
- linear diophantine equations with polynomial coefficients ("syzygies"),
- Hilbert functions,
algebraic relations among polynomials,
implicitization,
inverse polynomial mappings.

4 How Can Gröbner Bases Theory be Applied?

The general strategy of the Gröbner bases approach is as follows: Given a set \( F \) of polynomials in \( K[x_1, \ldots, x_n] \) (that describes the problem at hand)

- we transform \( F \) into another set \( G \) of polynomials "with certain nice properties" (called a "Gröbner basis") such that
  
- \( F \) and \( G \) are "equivalent" (i.e. generate the same ideal).

From the theory of GB we know:

- Because of the "nice properties of Gröbner bases", many problems that are difficult for general \( F \) are "easy" for Gröbner bases \( G \).
- There is an algorithm for transforming an arbitrary \( F \) into an equivalent Gröbner basis \( G \).
- The solution of the problem for \( G \) can often be easily translated back into a solution of the problem for \( F \).

Hence, by the properties of Gröbner bases and the possibility of transforming arbitrary finite polynomial sets into Gröbner bases, a whole range of problems definable in terms of finite polynomial sets becomes algorithmically solvable.

5 What are Gröbner Bases?

5.1 Division ("Reduction") of Multivariate Polynomials

We first need the notion of division (or "reduction") for multivariate polynomials. Consider, for example, the following bivariate polynomials \( g, f_1, \) and \( f_2 \), and the following polynomial set \( F \):

\[
g = x^2 y^3 + 3 x y^2 - 5 x, \tag{1}
\]

\[
f_1 = x y - 2 y, \quad f_2 = 2 y^2 - x^2, \tag{2}
\]

\[
F = \{ f_1, f_2 \}. \tag{3}
\]
The monomials in these polynomials are ordered. There are infinitely many orderings that are "admissible" for Gröbner bases theory. The most important ones are the lexicographic orderings and the orderings that, first, order power products by their degree and, then, lexicographically. In the example above, the monomials are ordered lexicographically with $y$ ranking higher than $x$ and are presented in descending order from left to right. The highest (left-most) monomial in a polynomial is called the "leading" monomial in the polynomial.

One possible division ("reduction") step that "reduces the polynomial $g$ modulo $f_1$" proceeds as follows:

$$h = g - (3y)f_1 = -5x + 6y^2 + x^2y^3,$$

i.e. in a reduction step of $g$ modulo $f_1$, by subtracting a suitable monomial multiple of $f_1$ from $g$, one of the monomials of $g$ should cancel against the leading monomial of $-(3y)f_1$. We write

$$g \rightarrow_{f_1} h$$

for this situation (read: "$g$ reduces to $h$ modulo $f_1$.")

### 5.2 In General, Many Reductions are Possible

Given a set $F$ of polynomials and a polynomial $g$, many different reductions of $g$ modulo polynomials in $F$ may be possible. For example, for $g$ and $F$ as above, we also have

$$h_2 = g - (xy^2)f_1 = -5x + 3xy^2 + 2xy^3,$$

and, hence,

$$g \rightarrow_{f_1} h_2,$$

$$g \rightarrow_{f_2} h_3.$$

### 5.3 Multivariate Polynomial Division Always Terminates But is Not Unique

We write

$$g \rightarrow_F h$$

if
for some $f \in F$, and we write
\[ g \rightarrow_f^* h \] (12)
if $g$ reduces to $h$ by finitely many reduction steps w.r.t. $F$. Also, we write
\[ h_F \] (13)
if $h$ cannot be reduced further (is "in reduced form") w.r.t. $F$. Here are a couple of fundamental facts on the notion of reduction:

**Fact (Termination):** For any $g$ and $F$, there are no infinite chains of reduction steps modulo $F$ starting from $g$.

**Fact (Reduction is Algorithmic):** There is an algorithm $RF$ that produces a reduced form w.r.t. $F$ for any given polynomial $g$, i.e., for all $g$ and $F$,
\[ g \rightarrow_F^* RF(F, g). \] (14)

An example of such an algorithm is the iteration of the following operation: Given $g$, consider the polynomials $f \in F$ until you find one whose leading power product divides one of the power products in $g$. If you found such an $f$ and power product in $g$ execute the corresponding reduction step. If not, stop.

**Fact (Non-uniqueness):** Given $g$ and $F$, there may exist $h$ and $k$, such that
\[ h_F \leftarrow_{F^*} g \rightarrow_{F^*} k_F \] (15)
but $h \neq k$.

### 5.4 Definition of Gröbner Bases

Now we define Gröbner bases to be sets of polynomials whose corresponding reduction is unique:

\[ F \text{ is a Gröbner basis } \iff \rightarrow_F \text{ is unique, i.e.} \]

\[ \forall g, h, k \ ( h_F \leftarrow_{F^*} g \rightarrow_{F^*} k_F \implies h = k ). \]
5.5 The "Application Theory of Gröbner Bases"

At first sight, one may not see why the defining property of Gröbner bases should play any fundamental role. The importance of this property stems from the following facts:

**Fact:** Gröbner bases have many "nice properties" and hence, for Gröbner bases, many fundamental problems can be solved by "easy" algorithms.

**Example (The "Main Problem of Polynomial Ideal Theory"):**

Let $F$ be a set of polynomials:

If $F$ is a Gröbner basis, then:

$$f \in \text{Ideal}(F) \iff f \rightarrow^* F 0.$$ 

Here, $\text{Ideal}(F)$ is the ideal generated by $F$, i.e. the set of all polynomials of the form $\sum_{i=1}^{m} p_i f_i$ with $f_i$ in $F$ and arbitrary polynomials $p_i$. As a consequence of the above property, the question whether or not $f \in \text{Ideal}(F)$, for Gröbner bases $F$, can be decided by just reducing $f$ modulo $F$ and checking whether or not the result of the reduction is 0. For general $F$, this question is very hard to decide and, in fact, in the older literature on polynomial ideal theory was called the "main problem of polynomial ideal theory".

**Example (The "Elimination Problem"):**

Let $F$ be a set of polynomials in the indeterminates $x_1, \ldots, x_n$, and let $i \leq n$:

If $F$ is a Gröbner basis, then:

$$\text{Ideal}(F) \cap K[x_1, \ldots, x_i] = \text{Ideal}(F \cap K[x_1, \ldots, x_i]).$$

As a consequence, a basis for the "$i$-th elimination ideal" $\text{Ideal}(F) \cap K[x_1, \ldots, x_i]$ of a finite Gröbner basis $F$ can be obtained by just taking those polynomials in $F$ that depend only on the first $i$ indeterminates. Again, this problem is very hard for general $F$. Having bases for all elimination ideals of a the ideal generated by a given $F$, one can now find all the solutions of the system of equations determined by $F$. One just starts by finding all the solutions of the univariate polynomial that forms the basis of the first elimination ideal and then proceeds by substituting these solutions into the bivariate basis polynomials of the second elimination ideal etc.
6 How Can GB be Constructed?

6.1 The Main Problem

The main problem now is how, given an arbitrary finite set $F$ of (multivariate) polynomials, one can find a set of polynomials $G$ such that $\text{Ideal}(F) = \text{Ideal}(G)$ and $G$ is a Gröbner basis.

6.2 An Algorithm

This problem can be solved by the following algorithm:

Start with $G := F$.
For any pair of polynomials $f_1, f_2 \in G$:

- Compute the "S-polynomial" of $f_1, f_2$
  and reduce it to a reduced form $h$ w.r.t. $G$.

- If $h = 0$, consider the next pair.

- If $h \neq 0$, add $h$ to $G$ and iterate.

6.3 S-Polynomials

The above algorithms needs the computation of "S-polynomials". Again, we give their definition in an example:

$$f_1 := x y - 2 y, \quad f_2 := 2 y^2 - x^2,$$

(16)

$$\text{S-polynomial}[f_1, f_2] = y f_1 - \frac{1}{2} x f_2 = \frac{x^3}{2} - 2 y^2.$$  

(17)

Note that the computation of the S-polynomial of two polynomials $f_1$ and $f_2$, first, involves multiplication of the two polynomials by such monomial factors that the leading power product of both polynomials becomes equal, namely the least common multiple of the leading power products of the two polynomials. By the subsequent subtraction, this least common multiple power product then vanishes! The intuition behind this notion is the following: The least common multiple of the "leading power products" of $f_1$ and $f_2$ is "the first possible polynomial" that allows two essentially different reductions modulo $\{f_1, f_2\}$. The main theorem of Gröbner bases theory then
shows that, given a finite $F$, if you "master" the finitely many $S$-polys, then you master the infinitely many polynomials that allow two or more essentially different reductions.

The notion of $S$-polynomials is the nucleus of algorithmic Gröbner bases theory. Note, however, that the notion of Gröbner bases is independent of the notion of $S$-polynomials and gives many interesting results also for nonalgorithmic polynomial ideal theory.

6.4 Specializations

It is interesting to note that the Gröbner bases algorithm,
- for linear polynomials, specializes to Gauss’ algorithm, and
- for univariate polynomials, specializes to Euclid’s algorithm.

7 Why Does This Work?

7.1 Termination of the Algorithm

Termination of the algorithm is nontrivial: At the beginning, there are only finitely many pairs of polynomials in $G$ for which the corresponding $S$-polynomials have to be computed. However, the reduction of some of the $S$-polynomials may result in a polynomial unequal zero that has to be adjoined to $G$. Hence, $G$ is growing and, consequently, the number of $S$-polynomials that have to be considered may also grow. However, by an application of "Dickson’s Lemma", [15], it can be shown that, ultimately, this process must always stop.

7.2 Correctness of the Algorithm

The correctness of the algorithm is based on the following "Main Theorem of Gröbner Bases Theory":

$$F \text{ is a Gröbner basis } \iff \forall_{f_1, f_2 \in F} RF[F, \text{ S-polynomial}[f_1, f_2]] = 0.$$  

The entire power of the Gröbner bases method lies in this theorem and its proof. The proof of this theorem is nontrivial. It proceeds by induction over the ordering of power products and needs a detailed analysis of the cases that may occur when polynomials are reduced, in one step, to different polynomials modulo two polynomials. The proof was first given in the PhD thesis of the author and then published in aequationes
mathematicae, see [3, 4]. An English translation of the 1970 paper is contained in the appendix of [9]. A modern version of the proof is spelled out in [10].

8 Examples

8.1 A Simple Set of Equations

We now show how Gröbner bases can be applied to solving systems of polynomial equations. Let us, first, consider again the example:

\[
\begin{align*}
    f_1 &= xy - 2y, \\
    f_2 &= 2y^2 - x^2, \\
    F &= \{f_1, f_2\}.
\end{align*}
\]

The Gröbner basis \( G \) of \( F \) is

\[
G := \{-2x^2 + x^3, -2y + xy, -x^2 + 2y^2\}.
\]

(If you have a mathematical software system like, for example, Mathematica available, you may compute Gröbner bases by just entering

\[
\text{GroebnerBasis}[F]
\]

into the system.)

By the fact that \( F \) and \( G \) generate the same ideal, \( F \) and \( G \) have the same solutions. The elimination property of Gröbner bases guarantees that, in case \( G \) has only finitely many solutions, \( G \) contains a univariate polynomial in \( x \). (Note that, here, we use the lexicographic order that ranks \( y \) higher than \( x \). If we used the lexicographic order that ranks \( x \) higher than \( y \) then, correspondingly, the Gröbner basis would contain a univariate polynomial in \( y \).) In fact, the above Gröbner basis is "reduced", i.e. all polynomials in the basis are reduced modulo the other polynomial in the basis. It can be shown that reduced Gröbner bases (with finitely many solutions) contain exactly one univariate polynomial in the lowest indeterminate. In our example, the univariate polynomial in \( x \) contained in \( G \) is

\[
-2x^2 + x^3.
\]

We now can solve this polynomial for \( x \), which gives us the possible solutions

\[
\{x \to 0, \; x \to 0, \; x \to 2\},
\]

that is

\[
x_1 = 0, \; x_2 = 0, \; x_3 = 2.
\]
If we now plug in, say, $x_2$ in the second and third polynomial of $G$, we obtain the two polynomials

$$0$$

and

$$-4 + 2 y^2,$$

i.e. two univariate polynomials in $y$. Theory tells us that, whatever the resulting polynomials in $y$ will be, they will always have a nontrivial greatest common divisor which, in fact, is just the non-vanishing polynomial of lowest degree. In our case, this is the polynomial

$$-4 + 2 y^2.$$  \hfill (24)

Now we can solve this polynomial for $y$, and we obtain the solutions

$$y_{3,1} = \sqrt{2}, \ y_{3,2} = -\sqrt{2}.$$  \hfill (25)

In this way, we can obtain all the solutions of $G$ and, hence, of $F$.

### 8.2 A More Complicated Set of Equations

Here is a more complicated set of equations:

\[
\begin{align*}
    f_1 &= xy - 2yz - z, \\
    f_2 &= y^2 - x^2z + xz, \\
    f_3 &= z^2 - y^2x + x, \\
    F &= \{f_1, f_2, f_3\}.
\end{align*}
\]

The corresponding Gröbner basis, w.r.t. the lexicographic ordering ranking $x$ higher than $y$ higher and $y$ higher than $z$, is

\[
\begin{align*}
    G := \{ & z + 4z^3 - 17z^4 + 3z^5 - 45z^6 + \\
    & 60z^7 - 29z^8 + 124z^9 - 48z^{10} + 64z^{11} - 64z^{12}, \\
    & -22001z + 14361yz + 16681z^2 + 26380z^3 + \\
    & 226657z^4 + 11085z^5 - 90346z^6 - 472018z^7 - \\
    & 520424z^8 - 139296z^9 - 150784z^{10} + 490368z^{11}, \\
    & 43083y^2 - 11821z + 267025z^2 - 583085z^3 + 663460z^4 - \\
    & 2288350z^5 + 2466820z^6 - 3008257z^7 + 4611948z^8 - \\
    & 2592304z^9 + 2672704z^{10} - 1686848z^{11}, \\
\end{align*}
\]  \hfill (27)
You may again observe that \( G \) contains a univariate polynomial in the lowest indeterminate \( z \). This time the degree of this polynomial is 12. The roots of this polynomial cannot be expressed by radicals. In principle, one may represent the roots as algebraic numbers (see the literature on computer algebra and the implementations in the various mathematical software systems) and then proceed by substituting the roots of the first polynomial into the other polynomials of the Gröbner basis. In this introductory paper, we rather switch to a numerical approximation of the roots:

\[
\begin{align*}
z_1 & = -0.3313043000789449 - 0.5869344538646171 i \\
z_2 & = -0.3313043000789449 + 0.5869344538646171 i
\end{align*}
\]

If we now substitute, say, \( z_1 \) into the other polynomials of \( G \) we obtain the three polynomials

\[
\begin{align*}
(-523.5194758552393 - 4967.646241304139 i) - (4757.861053433728 + 8428.965691949767 i) y, & \quad (31) \\
(-7846.89647617919 - 8372.055369776885 i) + 43083 y^2, & \quad (32) \\
(-16311.7 + 16611. i) + 43083 x. & \quad (33)
\end{align*}
\]

Theory tells us that the first polynomial is (an approximation to) the greatest common divisor of the first and the second polynomial. Hence, its solution gives us the common solution of the first and the second polynomial. Thus, we obtain

\[
y_{1,1} = -0.4735346386353353 - 0.20518443210789426 i
\]

Finally, we can substitute \( y_{1,1} \) into the last polynomial (which, in this particular case does not change it since \( y \) does not occur as an indeterminate) and we can obtain the solution

\[
x_{1,1,1} = 0.3786106927760740 - 0.3855581188501717 i
\]

In this way, we can obtain all the finitely many solutions of \( G \) and, hence, of \( F \).

### 8.3 Algebraic Relations

The problem of algebraic relations in invariant theory is the problem of asking whether, for example, the polynomial

\[
43083 x - 118717 z + 69484 z^2 + 402334 z^3 + 409939 z^4 + 1202033 z^5 - 2475608 z^6 + 354746 z^7 - 6049080 z^8 + 2269472 z^9 - 3106688 z^{10} + 3442816 z^{11}
\]
\[ p := x_1^7 x_2 - x_1 x_2^7 \]  

(36)

can be expressed as a polynomial in, for example,

\[
\begin{align*}
i_1 &:= x_1^2 + x_2^2 \\
i_2 &:= x_1^2 x_2^2 \\
i_3 &:= x_1^3 x_2 - x_1 x_2^3.
\end{align*}
\]  

(37)

(Note that the polynomials \(i_1, i_2, i_3\) form a system of fundamental invariants for \( \mathbb{Z}_4 \), i.e. a set of generators for the ring

\[ \{ f \in \mathbb{C}[x_1, x_2] \mid f(x_1, x_2) = f(-x_2, x_1) \}, \]  

(38)

e.g. \( i_1, i_2, i_3 \) are in this ring and, furthermore, all polynomials in this ring can be expressed as polynomials in \( i_1, i_2, i_3 \).

The theory of Gröbner bases tells us now that the above question can be answered by, first, computing a Gröbner basis \( G \) of the following polynomial set

\[ \{-i_1 + x_1^2 + x_2^2, -i_2 + x_1^2 x_2^2, -i_3 + x_1^3 x_2 - x_1 x_2^3\} \]  

(39)

w.r.t. a lexicographic ordering that ranks \( x_1, x_2 \) higher than \( i_1, i_2, i_3 \) and by reducing \( p \) modulo \( G \) and analyzing the result. In our example the (reduced) Gröbner basis is

\[
\begin{align*}
G := \{-i_1^2 i_2 + 4 i_1^2 + i_3^2, -i_2 + i_1 x_1^2 - x_1^4, \\
i_1^2 i_3 x_1 - 2 i_2 i_3 x_1 - i_1 i_3 x_1^3 + i_2^2 i_2 x_2 - 4 i_2^2 x_2, \\
i_1^2 x_1 - 2 i_2 x_1 - i_1 x_1^3 + i_3 x_2, \\
-i_1 i_3 + 2 i_3 x_1^2 - i_1^2 x_1 x_2 + 4 i_2 x_1 x_2, -i_3 x_1 - 2 i_2 x_2 + i_1 x_1^2 x_2, \\
-i_3 - i_1 x_1 x_2 + 2 x_1^3 x_2, -i_1 + x_1^2 + x_2^2\},
\end{align*}
\]  

(40)

and reduction of \( p \) modulo \( G \) yields

\[ h := i_1^2 i_3 - i_2 i_3. \]  

(41)

(Please use mathematical software system like Mathematica for carrying out these computations.)

The theory of Gröbner bases now tells us that \( p \) can be expressed as a polynomial in \( i_1, i_2, i_3 \) if and only if \( h \) is a polynomial only in the indeterminates \( i_1, i_2, i_3 \), i.e. does not contain the indeterminates \( x_1, x_2 \). This is the case in our example. Thus, we know that \( p \) is representable as a polynomial in the polynomials \( i_1, i_2, i_3 \) and, furthermore, \( h \) gives us the actual representation, namely

\[ p = (x_1^2 + x_2^2)^2 (x_1^3 x_2 - x_1 x_2^3) - (x_1^2 x_2^2) (x_1^3 x_2 - x_1 x_2^3). \]  

(42)
9 How Difficult is it to Construct Gröbner Bases?

Very Easy

The structure of the algorithm is easy. The operations needed in the algorithm are elementary: "Every high-school student can execute the algorithm."

Very Difficult

The intrinsic complexity of the problems that can be solved by the Gröbner bases method is proven to be "exponential". Hence, the worst-case complexity of any algorithm that computes Gröbner bases in full generality must be high. Thus, examples in three or four variables with polynomials of degree three or four may already fail to terminate in reasonable time or exceed available memory even on very fast machines.

For example, trying to find a Gröbner basis, w.r.t. to a lexicographic ordering, for the set

\[
\{x y^3 - 2 y z - z^2 + 13, \\
y^2 - x^2 z + x z^2 + 3, \\
z^2 x - y^2 x^2 + x y + y^3 + 12\}
\]

may already exhaust your computer resources.

Sometimes Easy

Mathematically interesting examples often have a lot of "structure" and, in concrete examples, Gröbner bases can be computed in reasonably short time. Thus, a lot of interesting new theoretical insight in various areas of mathematics has been obtain by using the Gröbner bases technique for concrete, theoretically interesting, examples. Also, sometimes, it is possible to derive closed formulae for the Gröbner bases of certain ideals that depend on various parameters and, then, various conclusions can be drawn from the form of these Gröbner bases, see for example [8]. Hence, as a first attempt, it is always recommendable to try Gröbner bases if one encounters a problem formulated in terms of multivariate polynomial sets.

Enormous Potential for Improvement

The positive aspect of an intrinsically complex problem as the one of constructing Gröbner bases is that more mathematical knowledge can lead to a drastic speed-up. In the literature, the following ideas have led to drastically improved versions of the above Gröbner basis algorithm:
The use of "criteria" for eliminating the consideration of certain S-polynomials, see [6].

Several $p$-adic and floating point approaches, see [21, 20].

The "Gröbner Walk" approach, see [13].

The "linear algebra" approach, see [16].

All these approaches do, however, not change the main idea of the algorithmic construction given above based on the fundamental role of the "S-polynomials". For the practical implementation of the Gröbner basis algorithm, tuning of the algorithm is also important, for example by

- heuristics and strategies for choosing favorable orderings of power products and for the sequence in which S-polynomials should be selected etc,
- good implementation techniques and data structures.

There is a huge literature on the complexity of Gröbner bases algorithms and on improving the efficiency of these algorithms.

10 Why are Gröbner Bases Called Gröbner Bases?

Professor Wolfgang Gröbner was my PhD thesis supervisor in 1965. He gave me the problem of finding a linearly independent basis for the residue class ring modulo an arbitrary polynomial ideal given by finitely many generators. On the way of answering this question, I developed the theory of what I later (1976, see [5]), in honor of my former advisor, called "Gröbner bases". In more detail, in my thesis (1965) and journal publication (1970), I introduced the following notions, theorems, and methods:

- the concept of Gröbner bases and reduced Gröbner bases,
- the concept of S-polynomial,
- the main theorem with proof,
- the algorithm with termination and correctness proof,
- the uniqueness of reduced Gröbner bases,
- first applications (algorithmic computing in residue class rings, Hilbert function computation, solution of algebraic systems),
- the technique of base-change w.r.t. to different orderings,
- a complete running implementation with examples,
- first complexity considerations.
Later, I contributed mainly the following two ideas to the theory of Gröbner bases:

- the technique of criteria for eliminating unnecessary reductions,
- an abstract characterization of rings ("reduction rings") in which a Gröbner bases approach is possible.

In my view, the main additional ideas that have been contributed to the theory of Gröbner bases by other authors are the following:

- Gröbner bases can be constructed w.r.t. arbitrary "admissible" orderings (W. Trinks 1978).
- Gröbner bases w.r.t. to "lexical" orderings have the elimination property (W. Trinks 1978).
- Gröbner bases can be used for computing syzygies, and the S-polys generate the module of syzygies (G. Zacharias 1978).
- A given $F$, w.r.t. the infinitely many admissible orderings, has only finitely many Gröbner bases and, hence, one can construct a "universal" Gröbner bases for $F$ (L. Robbiano, V. Weispfenning, T. Schwarz 1988).
- Starting from a Gröbner bases for $F$ for ordering $O_1$ one can "walk", by changing the basis only slightly, to a basis for a "nearby" ordering $O_2$ and so on ... until one arrives at a Gröbner bases for a desired ordering $O_k$ (Kalkbrenner, Mall 1995).
- Numerous applications of Gröbner bases for solving problems in various fields of mathematics that, sometimes, needed ingenious ideas for establishing the reduction of the problems considered to the computation of Gröbner bases.

11 Where Can You Find Information on Gröbner Bases?

11.1 The Gröbner Bases Conference 1998

The proceedings of this conference, [9], contain tutorials on nearly all currently known applications of Gröbner bases in various fields of mathematics. Unfortunately, no tutorial on applications of Gröbner bases in systems theory is contained in these proceedings.

These proceedings contain also a couple of original papers and an introduction to Gröbner bases including a complete formal proof of the main theorem, see [10]. Also, in the appendix, an English translation of the original paper [4] is included.
11.2 On Your Desk

Implementations of the Gröbner basis algorithms and many application algorithms based on Gröbner bases are contained in any of the current mathematical software systems like Mathematica, Maple, Magma, Macsyma, Axiom, Derive, Reduce, etc. Also, there exist special software systems that are mainly based on the Gröbner bases technique, for example, CoCoA [12], Macaulay [17], Singular [18].

11.3 In Your Palm

Gröbner bases are now available on the TI-92 (implemented in Derive) and other palm-top calculators so that literally every high-school student has access to the method.

11.4 Textbooks

By now, a couple of very good textbooks are available on Gröbner bases, see for example, [19], [2], [1], [14]. The textbook [19], in the introduction, contains a complete list of all current textbooks.

11.5 In the Web

Searching in the web, for example starting at http://citeseer.nj.nec.com/ with the key word "Groebner" will quickly lead you to hundreds of papers on Gröbner bases and their applications.

11.6 Original Publications

By now, more than 500 papers appeared meanwhile on Gröbner bases. Many of them are appearing in the Journal of Symbolic Computation (Academic Press, London) or at the ISSAC Symposia (International Symposia on Symbolic and Algebraic Computation).
References


Design for Reuse via Structuring Techniques for ASMs

Egon Börger

Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy
boerger@di.unipi.it

Abstract. Gurevich’s Abstract State Machines (ASMs), characterized by the parallel execution of abstract atomic actions in a global state, have been equipped with a refinement by standard composition concepts for structuring large machines that allows reusing machine components. Among these concepts are parameterized (possibly recursive) sub-ASMs. Here we illustrate their power for incremental and modular system design by unfolding, via appropriate ASM components, the architecture of the Java Virtual Machine (JVM), resulting from the language layering in combination with the functional decomposition of the JVM into loader, verifier, and interpreter. We survey the ASM models for Java and the JVM that appear in, together with the mathematical and experimental analysis they support.

1 The Method: Structuring ASMs by Submachines

Although it was by a foundational concern, namely of reconsidering Turing’s thesis in the light of the problem of the semantics of computer programs, that Gurevich was led to formulate the idea of Abstract State Machines, it did not take a long time that the concept was recognized to be of practical importance. ASMs were soon successfully applied for the modeling and a rigorous analysis of a variety of complex real-life computing systems: programming languages and their implementations, processor architectures, protocols, embedded software, etc., see [5, 6] for a historical account. The first industrial application showed up as early as 1990 in the ASM model defining the semantics of PROLOG [2, 3, 9], which became the official ISO standard [28] and has been run for experimentation at Quintus, see [4] for a survey of these early applications of ASMs in the context of logic programming. By now a powerful method has been built around the
concept of ASM, which supports industrial system design by rigorous high-level modeling that is seamlessly linked to executable code, namely by mathematically verifiable, experimentally validatable, and objectively documentable refinement steps. Here are some highlights:

- The reengineering of a central component in a large software package for constructing and validating timetables for railway systems, work done at Siemens from May 1998 to March 1999. A high-level ASM model for the component was built, compiled to C++ and successfully integrated into the existing software system which since then is in operation at Vienna subways [14].
- The ASM definition of the International Telecommunication Union standard for SDL2000 [29].
- The investigation (verification and validation) of Java and its implementation by the Java Virtual Machine in terms of ASM models and their Asm-Gofer executable refinements for the language and the VM [34].
- The recent ASM model for the UPnP architecture at Microsoft [15].

For the impressive up-to-date list of annotated references to ASM publications and tools the reader may consult the ASM website [27].

One of the reasons for the simplicity of Gurevich’s notion of Abstract State Machine—which its mathematical content can be explained in less than an hour, see Chapter 2 of [34] for a textbook definition starting from scratch—lies in the fact that its definition uses only conditional assignments, so-called rules of form

\[
\text{if } \text{Condition} \text{ then } f(t_1, \ldots, t_n) := t
\]

expressing guarded atomic actions that yield updates in a well-defined (a global) state. In this respect ASMs are similar to Abrial’s Abstract Machines [1] that are expressed by non-executable pseudo-code without sequencing or loop (Abstract Machine Notation, AMN). It is true that this leaves the freedom—so necessary for high-level system design and analysis—to introduce during the modeling process any control or data structure whatsoever that may turn out to be suitable for the application under study. However, the other side of the coin is that this forces the designer to specify standard control or data structures and standard component based design structures over and over again, namely when it comes to implement the specifications, thus making effective reuse difficult. For some time it was felt as a challenge to combine, in a practically viable manner, the simplicity of the parallel execution model of atomic actions in a global state with the structuring capabilities of modules and components as part of a large system architecture, whose execution implies duration and scheduling.

In [13] a solution has been developed that naturally extends the characteristic ASM notion of synchronous parallel execution of multiple atomic actions (read: rules) by allowing as rules also calling and execution of submachines, technically speaking named, parameterized, possibly recursive, ASMs. This definition gently embeds the result of executing an a priori unlimited number \( n \) of micro steps—namely steps of a submachine that has been called for execution.
in a given state—into the \textit{macro step} semantics of the calling ASM, which is defined as the overall result of the simultaneous execution of all its rules in the given state. The same treatment covers also the classical control constructs for \textit{sequentialization} and \textit{iteration} and opens the way to structuring large ASMs by making use of instantiatable machine components. Whereas for the AMN of the B method Abrial explicitly excludes e.g. sequencing and loop from the specification of abstract machines \cite{Abrial92}, we took a more pragmatic approach and defined these control constructs, and more generally the notion of ASM submachine in such a way that they can be used coherently in two ways, depending on what is needed, namely to provide black-box descriptions of the behavior of components or glass-box views of their implementation (refinement).

In the present survey we illustrate that this notion of submachines, which has been implemented in AsmGofer \cite{Borger05}, suffices for a hierarchical decomposition of the Java Virtual Machine into components for the loader, the verifier, and the interpreter, each of them split into subcomponents for the five principal language layers (imperative core, static classes, object oriented features, exception handling and concurrency). We can do this in such a way that adding a component corresponds to what in logic is called extending a theory conservatively. This incremental design approach is the basis for a transparent yet far reaching mathematical analysis of Java and its implementation on the JVM (correctness and completeness proofs for the compilation, the bytecode verification, and the execution, i.e. interpretation), which appears in \cite{Borger06a}.

\textbf{Graphical notation.} Before we proceed in the next section to explain the problem of a mathematically transparent model for Java and its implementation on the JVM, and the solution offered in \cite{Borger06a}, we review here the basic graphical (UML like) notation we will use for defining structured ASMs. To describe the overall structure of the JVM we only need special ASMs that resemble the classical Finite State Machines (FSMs) in that their execution is governed by a set of internal or control states (often also called \textit{modes}) which split the machine into finitely many submachines. Formally these ASMs, which I have called \textit{control state} ASMs in \cite{Borger03}, are defined and pictorially depicted as shown in Fig. 1 with \textit{transition rules} of form

\begin{verbatim}
if Condition then f(t_1, \ldots, t_n) := t
\end{verbatim}

whose execution is to be understood as \textit{changing} (or defining, if there was none) the value of the function \(f\) at the given parameters. Note that in a given control state \(i\), these machines do nothing when no condition \(\text{cond}_j\) is satisfied.

\footnote{The atomicity of this ASM iteration constructor is the key for a rigorous definition of the semantics of event triggered exiting from compound actions of UML activity and state machine diagrams, where the intended instantaneous effect of exiting has to be combined with the request to exit nested diagrams sequentially following the subdiagram order, see \cite{Borger01,Borger02}.}

\footnote{In \cite{Borger03} we also incorporate into standard ASMs a syntax oriented form of information hiding, namely through the notion of \textit{local machine state}, of machines with \textit{return values} and of \textit{error handling} machines.
Assume disjoint cond\_i. Usually the "control states" are notationally suppressed.

**Fig. 1.** Control state ASM diagrams

The notion of ASM *states*, differently from FSMs, is the classical notion of mathematical *structures* where data come as abstract objects, i.e., as elements of sets (domains, one for each category of data) that are equipped with basic operations (partial *functions*) and predicates (attributes or relations). The notion of ASM *run* is the classical notion of computation of transition systems. An ASM computation step in a given state consists in executing *simultaneously* all updates of all transition rules whose guard is true in the state, if these updates are consistent.

The synchronous parallelism inherent in the simultaneous execution of all ASM rules is enhanced by the following concise notation for the simultaneous execution of an ASM rule $R$ for each $x$ satisfying a given condition $\phi$:

$$\text{forall } x \text{ with } \phi \text{ do } R$$

A frequently encountered kind of functions whose detailed specification is left open are choice functions, used to abstract from details of static or dynamic scheduling strategies. ASMs support the following concise notation for an abstract specification of such strategies:

$$\text{choose } x \text{ with } \phi \text{ do } R$$

meaning to execute rule $R$ with an arbitrary $x$ chosen among those satisfying the selection property $\phi$. If there exists no such $x$, nothing is done. For *choose* and *forall* rules we also use graphical notations of the following form:

<table>
<thead>
<tr>
<th>choose $x$ with $\phi$</th>
<th>forall $x$ with $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$R$</td>
</tr>
</tbody>
</table>
2 The Java/JVM Modeling and Analysis Problem

The scientific problem to solve was to investigate in which sense and to what extent one can provide a rigorous justification of the claim that Java and the JVM provide a safe and secure, platform independent programming environment for the internet. This claim goes beyond the traditional correctness problem for language compilation and the interpretation of the compiled code on a virtual or real machine, a classical problem which has been studied extensively for other source code languages and compiler target machines, including some work where ASMs are used as modeling device (e.g. [12, 10, 18, 19]). Not only is the problem of trusted (i.e. fully correct) realistic compilation not yet solved (see [16, 17] for a thorough discussion), the case of Java and its implementation on the JVM adds further problems, partly due to the fact that the access to resources by the executed code is controlled not by the operating system, but by the JVM that interprets this code, namely dynamically loaded and verified bytecode. As a result one has at least three new correctness and completeness problems, as illustrated in Fig. 2, namely concerning:

- The loading mechanism which dynamically loads classes; the binary representation of a class is retrieved and installed within the JVM—relying upon some appropriate name space definition to be used by the security manager—and then prepared for execution by the JVM interpreter
- The bytecode verifier, which checks certain code properties at link-time, e.g. conditions on types and on stack bounds which one wants to be satisfied at run-time
- The access right checker, i.e., a security manager which controls the access to the file system, to network addresses, to critical windowing operations, etc.

Fig. 2. Security oriented decomposition of the JVM
The goal of the project was to provide an abstract (read: platform independent), rigorous but transparent, modular definition of Java and the JVM that can be used as a basis for a mathematical and an experimental analysis of the above claim. First of all this modeling work should reflect SUN’s design decisions, it should provide for the two manuals [20, 21, 31] what in [5, 11] has been called a ground model, i.e. a sufficiently rigorous and complete, provably consistent, mathematical model that faithfully represents the given natural language descriptions. Secondly it should offer a correct high-level understanding of

- the source language, to be practically useful for Java programmers,
- the virtual machine, to offer the implementors a rigorous, implementation independent basis for the documentation, the analysis, and the comparison of implementations.

We tried to achieve the goal by constructing stepwise refined ASM models of Java, the JVM (including the loader and the bytecode verifier), and a Java-to-JVM compiler, which are abstract, but nevertheless can in a natural way be turned into executable validatable models, and for which we can prove the following theorem.

**Main Theorem.** Under conditions that are explicitly stated in [34], any well-formed and well-typed Java program, when compiled satisfying the properties listed for the compiler, passes the bytecode verifier and is executed on the JVM. During this execution, none of the run-time checks of the properties that have been analyzed by the verifier is violated, and the generated bytecode is interpreted correctly with respect to the expected source code behavior as defined by the Java ASM.

In the course of proving the theorem, we were led to clarify various ambiguities and inconsistencies we discovered in the Java/JVM manuals and in the implementations, concerning fundamental notions like legal Java program, legal bytecode, verifiable bytecode, etc. Our analysis of the JVM bytecode verifier, which we relate to the static analysis of the Java parser (rules of definite assignment and reachability analysis), led us to define a novel (subroutine call stack free) bytecode verifier which goes beyond previous work in the literature.

In the next section we explain the dependency graph which surveys how we split the proof of the main theorem in subproofs for the JVM components.

3 Decomposition of Java/JVM into Components

To make such a complex modeling and analysis problem tractable one has to split it into a series of manageable subproblems. To this end we construct the ASM for the JVM out of submachines for its security relevant components—the ones which appear in Fig. 2: loader, verifier, preparator, interpreter—and define each component incrementally via a series of submachines, put together by parallel composition and forming a sequence of conservative extensions, which is guided
by the layering of Java and of the set of JVM instructions into increasingly richer sublanguages.

**Components for Language Layers.** Since this language layering is common to all JVM components, we explain it first. We factor the sets of Java and of JVM instructions into five sublanguages, by isolating language features which represent milestones in the evolution of modern programming languages and of the techniques for their compilation, namely imperative (sequential control), procedural (module), object-oriented, exception handling, and concurrency features. This decomposition can be made in such a way that in the resulting sequence of machines, each ASM is a purely incremental—similar to what logicians call a conservative—extension of its predecessor, because each of them provides the semantics of the underlying language, instruction by instruction. The general compilation scheme compile can then be defined between the corresponding submachines by a simple recursion. We illustrate this in Fig. 3.
A related structuring principle, which helped us to keep the size of the models small, consists in grouping similar instructions into one abstract instruction each, coming with appropriate parameters. These parameters become parameters of the corresponding ASM rules describing the semantical effect of those instructions. This goes without leaving out any relevant language feature, given that the specializations can be regained by mere parameter expansion, a refinement step whose correctness is easily controllable instruction-wise.

**Execution Component.** We now turn to explain the vertical components of the ASM model for the JVM. In one component we describe the *trustful execution* of bytecode that is assumed to be successfully loaded and linked (i.e., prepared and verified to satisfy the required link-time constraints). The resulting sequence of stepwise refined trustful VMs, namely $\text{trustfulVM}_I$, $\text{trustfulVM}_C$, $\text{trustfulVM}_O$, $\text{trustfulVM}_E$, and $\text{trustfulVM}_T$, yields a succinct definition of the functionality of JVM execution in terms of language layered submachines $\text{execVM}$ and $\text{switchVM}$ (Fig. 5).

The language layered machine $\text{execVM}$ describes the effect of each single JVM instruction on the current frame, whereas $\text{switchVM}$ is responsible for frame stack manipulations upon method call and return, class initialization and exception capture. This piecemeal description of single JVM instructions can be done similarly for the instructions provided in Java, yielding a succinct definition of the semantics of Java in terms of language layered submachines $\text{Java}_I$, $\text{Java}_C$, $\text{Java}_O$, $\text{Java}_E$, and $\text{Java}_T$. Exploiting the correspondence between these components for the Java/JVM machines yields a simple recursive definition of a compilation scheme for Java programs to JVM code, see Fig. 4, the detailed definition is in Part II of [34]. The conservativity of the component extensions allowed us to incrementally prove this compilation scheme to be correct, as is expressed by the following theorem.
Theorem 1 (Correctness of the compiler). The ASMs for Java and the JVM, running through given Java code and its compilation to JVM code, produce in corresponding method code segments the same values for (local, global, heap) variables and the same results of intermediate calculations, for the current method as well as for the method calls still to be completed.

The proof includes a correctness proof for the handling of Java exceptions in the JVM, a feature which considerably complicates the bytecode verification, in the presence of embedded subroutines, class and object initialization, and concurrently working threads. Obviously, the statement of the theorem as phrased here is vague. In fact, it is part of the modeling and analysis work to provide a precise meaning of this intuitive statement, expressing that runs of the Java machine on a Java program and the corresponding runs of the JVM machine on the compiled program are equivalent. It took us 10 pages to make the underlying notion of corresponding runs and of their equivalence sufficiently precise to be able to carry out a proof for the correctness theorem, see Chapter 14 of [34]. The 83 case distinctions of that 24 pages long proof are not a bizarre effect of our modeling, but directly derive from—it is indeed structured into—the situations which do occur during a Java computation for expression evaluation and statement execution, treated separately for each of the five language layers. This is a strength of the method that by localizing the proof obligations one has a key to modularize the overall proof: each new expression or statement feature will bring with it a clearly identifiable group of new cases to consider for definition (modeling) and proof (verification).

It was crucial for the compiler correctness proof to go through to take into account also some structural static constraints about Java runs, in particular conditions under which it can be proved that well-formed and well-typed Java programs are type safe, including the so called definite assignment rules for variables and the reachability analysis for statements. In fact we were led to
Fig. 6. Decomposing defensive VMs into trustful VMs and checks

correct some inconsistencies in those rules as defined in SUN’s manuals (see below).

Checking Component. The second group of language layered component machines we define are auxiliary machines whose parallel composition constitutes the defensive VM. Their purpose is to define the verifier functionality in run-time terms of trustful VM execution from a language layered component check. Since it is difficult to obtain a well motivated and clear definition of the bytecode verification functionality, we tried to accomplish also that task locally: guided by the language structure that allows to successively refine the checking conditions—from the imperative to the dynamic submachine—we took advantage from knowing for each type of instruction some run-time conditions which can guarantee its safe executability. To be more precise, as the architectural definition in Fig. 6 shows, the defensive VM checks at run-time, before every execution step, the structural constraints which describe the verifier functionality (restrictions on run-time data: argument types, valid return addresses, resource bounds) guaranteeing safe execution. (Note that the static constraints on the well-formedness of the bytecode in Java class files are checked at link-time.) The detailed definition is given in Chapter 15 of [34]. For this new ASM defensive VM, by its construction out of its component trustful VM, one has the following theorem.

Theorem 2 (Correctness of defensive checking). If the defensive VM executes a program P successfully, then so does the trustful VM, with the same semantical effect.

Since we formulate the run-time checking conditions referring to the types of values in registers and on the operand stack, instead of the values themselves, we can lift them to link-time checkable bytecode type assignments, i.e. assignments of certain type frames to code indices of method bodies. When lifting the run-time constraints, we make sure that if a given bytecode has a type assignment, then the code runs on the defensive VM without violating any of the run-time check conditions. For example, at run-time the values of the operands and the values stored in local variables belong to the assigned types; if there is a verify type
assigned to a local variable, then at run-time the local variable contains a value which belongs to that verify type; if the type is a primitive type, then the value is of exactly that type; if the type is a reference type, then the value is a pointer to an object or array which is compatible with that type; the same is true for the verify types assigned to the operand stack, etc. The main difficulty is due to the subroutines, more precisely to the \texttt{Jsr(s)} and \texttt{Ret(x)} instructions which are used in the JVM to implement the \textit{finally} block of Java \texttt{try} statements in the exception handling mechanism of Java. The problem is to correctly capture what is the type of return addresses from subroutines; as a matter of fact concerning this point we have identified in Chapter 16 of \cite{34} a certain number of problems and inconsistencies in current implementations of the bytecode verifier. The outcome of this analysis is the following theorem, whose proof documents for all the cases that can occur for the single instructions in the given run why typable code can be safely executed.

\textbf{Theorem 3 (Soundness of Bytecode Type Assignments).} Typable bytecode satisfies at run-time a set of invariants guaranteeing that when the code is run on the defensive VM, it does not violate any of the dynamic constraints defined in the check component.

The notion of bytecode type assignment also allows us to prove the completeness of the compilation scheme mentioned above. Completeness here means that bytecode which is compiled from a well-formed and well-typed Java program in a way that respects our compilation scheme, can be typed successfully, in the sense that it does have type assignments. More precisely we prove the general statement below, which implies the correctness of our Java-to-JVM compiler. We refine our compiler to a certifying code generator, which issues instructions together with the type information needed for the bytecode verification. Hence, the result of the extended compilation is not only a sequence of bytecode instructions but a sequence of triples \((\text{instr}, \text{regT}, \text{opdT})\), where \((\text{regT}, \text{opdT})\) is what we call a type frame for the instruction \text{instr}. We then prove that the so generated type frames satisfy the conditions for bytecode type assignments. This is yet another example of structuring definition and proof by conservative (purely incremental) extension.

When working on this proof, we detected a not so obvious inconsistency in the design of the Java programming language, namely an incompatibility of the reachability notions for the language and the JVM, related to the treatment of boolean expressions and the rules for the definite assignment of variables. The program in Fig. 7 shows that bytecode verification is not possible the way SUN’s manuals suggest: although valid, the program is rejected by any bytecode verifier we have tried including JDK 1.2, JDK 1.3, Netscape 4.73-4.76, Microsoft VM for Java 5.0 and 5.5 and the Kimera Verifier (\url{http://kimera.cs.washington.edu/}). The problem is that in the eyes of the verifier the variable \(i\) is unusable at the end of the method at the \texttt{return i} instruction, whereas according to 16.2.14 in \cite{21} the variable \(i\) is definitely assigned after the try statement. Our rules of
class Test {
    static int m(boolean b) {
        int i;
        try {
            if (b) return 1;
            i = 2;
        } finally { if (b) i = 3; }
        return i;
    }
}

Fig. 7. A valid Java program rejected by all known verifiers

definite assignment for the try statement are stronger and therefore the program is already rejected by our compiler. In [34] we exhibit another program that illustrates a similar problem for labeled statements. In conclusion, one can avoid this inconsistency by slightly restricting the class of valid programs by sharpening the rules for definite assignment for `finally` and for labeled statements. As a result we could establish the following desirable property for the class of certifying compilers.

Theorem 4 (Compiler Completeness Theorem). The family of type frames generated by the certifying compiler for the body of a method $\mu$ is a bytecode type assignment for $\mu$.

As a corollary, the Java-to-JVM compiler we define is correct since it is extended conservatively by a certifying compiler.

Bytecode Verifier Component. Having distilled the bytecode verifier functionality in the notion of bytecode type assignment, we are ready to extend the `trustfulVM` by a new component, a link-time bytecode verifier. Before `trustfulVM` can run a method in a class that has been loaded, for each method in that class the verifier attempts to compute a—in fact a most specific—bytecode type assignment for the method. The (architecture of the) resulting machine `diligentVM` is defined in Fig. 8.

One has to show that the `verifyVM` component is sound and complete, which is expressed by the following two theorems that we can prove for our novel (subroutine call stack free) bytecode verifier.

Theorem 5 (Bytecode Verifier Soundness). During the computation of the verifier for any given method body, the bytecode type frames computed so far satisfy the conditions for bytecode type assignments. `verifyVM` terminates, either rejecting the code with a type failure detection (in case the method body is not typable) or accepting it and issuing a bytecode type assignment for it.
Theorem 6 (Bytecode Verifier Completeness). If a method body has a bytecode type assignment, then verifyVM accepts the code and during the verification process the type frames computed so far by verifyVM are more specific than that bytecode type assignment.

Components of the Bytecode Verifier. To compute a bytecode type assignment for a given method, verifyVM at each step chooses a still to be verified code index \( pc \), starting at code index 0, to check the type conditions there. Upon successful check, as defined for the defensiveVM, the verifier marks for further verification steps the indices of all successors of \( pc \) that can be reached by the computation, trying to propagate the type frame computed at \( pc \) to each possible immediate successor of \( pc \). This provides the architecture of the machine verifyVM, built out of three components check, propagate, succ as defined in Fig. 9.

At this point it should not any more come as a surprise to the reader that the two new components of verifyVM, namely the ASM propagateVM and the function succ, are language layered similarly to the predicate check defined al-
already above as part of defensive VM. A further reuse of previously defined machines stems from the fact that the submachine propagateVM, together with the function succ, defines a link-time simulation (type version) of the trustfulVM illustrated above.

In a similar way the loading mechanism can be introduced by refining the components execVM and switchVM, see Chapter 18 in [34].

The modular component-based structure of both definitions and proofs explained above for Java and the JVM is reassumed in Fig. 3, showing how the components and the proofs of their basic properties fit together to establish the desired property for the compilation and safe execution of arbitrary Java programs on the dynamicVM, as expressed above in the Main Theorem.

**AsmGofer executable refinements.** The experimentation with the AsmGofer executable refinements of the models outlined above was crucial to get the models and the proofs of our theorems right. AsmGofer is an ASM programming system developed by Joachim Schmid and available at www.tydo.de/AsmGofer. It extends TkGofer to execute ASMs which come with Haskell definable external functions. It provides step-by-step execution and comes with GUIs to support debugging of Java/JVM programs. First of all it allows to execute the Java source code in our Java ASM and to observe that execution—there is no counterpart for this in SUN’s development environment, but similar work has been done independently, using the Centaur system, by Marjorie Russo in her recent PhD thesis [32]. Furthermore one can compile Java programs to bytecode which can be executed either on our ASM for JVM or (using Jasmin for the conversion to binary class format) on SUN’s implementation. More generally, for the executable versions of our machines, the formats for inputting and compiling Java programs are chosen in such a way that the ASMs for the JVM and the compiler can be combined in various ways with current implementations of Java compilers and of the JVM, as illustrated in Fig. 10.

---

**Fig. 10.** Relationship between different machines

```
Fig. 10. Relationship between different machines

ready above as part of defensive VM. A further reuse of previously defined machines stems from the fact that the submachine propagateVM, together with the function succ, defines a link-time simulation (type version) of the trustfulVM illustrated above.

In a similar way the loading mechanism can be introduced by refining the components execVM and switchVM, see Chapter 18 in [34].

The modular component-based structure of both definitions and proofs explained above for Java and the JVM is reassumed in Fig. 3, showing how the components and the proofs of their basic properties fit together to establish the desired property for the compilation and safe execution of arbitrary Java programs on the dynamicVM, as expressed above in the Main Theorem.

**AsmGofer executable refinements.** The experimentation with the AsmGofer executable refinements of the models outlined above was crucial to get the models and the proofs of our theorems right. AsmGofer is an ASM programming system developed by Joachim Schmid and available at www.tydo.de/AsmGofer. It extends TkGofer to execute ASMs which come with Haskell definable external functions. It provides step-by-step execution and comes with GUIs to support debugging of Java/JVM programs. First of all it allows to execute the Java source code in our Java ASM and to observe that execution—there is no counterpart for this in SUN’s development environment, but similar work has been done independently, using the Centaur system, by Marjorie Russo in her recent PhD thesis [32]. Furthermore one can compile Java programs to bytecode which can be executed either on our ASM for JVM or (using Jasmin for the conversion to binary class format) on SUN’s implementation. More generally, for the executable versions of our machines, the formats for inputting and compiling Java programs are chosen in such a way that the ASMs for the JVM and the compiler can be combined in various ways with current implementations of Java compilers and of the JVM, as illustrated in Fig. 10.
```
References


On CAST.FSM Computation of Hierarchical Multi-layer Networks of Automata

Michael Affenzeller, Franz Pichler, and Rudolf Mittelmann

Institute of Systems Science
Systems Theory and Information Technology
Johannes Kepler University
Altenbergerstrasse 69
A-4040 Linz - Austria

Abstract. CAST.FSM denotes a CAST tool which has been developed at the Institute of Systems Science at the University of Linz during the years 1986-1993. The first version of CAST.FSM was implemented in INTERLISP-D and LOOPS for the Siemens-Xerox workstation 5815 ("Dandelion"). CAST.FSM supports the application of the theory of finite state machines for hardware design tasks between the architecture level and the level of gate circuits. The application domain, to get practical experience for CAST.FSM, was the field of VLSI design of ASICS’s where the theory of finite state machines can be applied to improve the testability of such circuits ("design for testability") and to optimise the required silicon area of the circuit ("floor planning"). An overview of CAST as a whole and of CAST.FSM as a CAST tool is given in [11]. In our presentation we want to report on the re-engineering of CAST.FSM and on new types of applications of CAST.FSM which are currently under investigation. In this context we will distinguish between three different problems:

1. the implementation of CAST.FSM in ANSI Common Lisp and the design of a new user interface by Rudolf Mittelmann [5].
2. the search for systemstheoretical concepts in modelling intelligent hierarchical systems based on the past work of Arthur Koestler [3] following the concepts presented by Franz Pichler in [10].
3. the construction of hierarchical formal models (of multi-layer type) to study attributes which are assumed for SOHO-structures (SOHO = Self Organizing Hierarchical Order) of A. Koestler.

The latter problem will deserve the main attention in our presentation. In the present paper we will build such a hierarchical model following the concepts of parallel decomposition of finite state machines (FSMs) and interpret it as a multi-layer type of model.

1 Implementation of CAST.FSM in ANSI Common Lisp

CAST.FSM was implemented during the year 1986 in INTERLISP-D and LOOPS on Siemens 5815 work stations [12]. However, already in 1992 those workstations had to be put out of order. For the continuation of our research
tasks in VLSI design it was necessary to implement parts of CAST.FSM in Common Lisp and Flavors \cite{7}. In addition, new algorithms were developed to speed up computation and to deal with new areas of finite state machine problems. On the basis of this implementation of CAST.FSM (internally called CAST 2), Rudolf Mittelmann made a portation from Flavors to CLOS for Apple Macintosh computers using Procyon Common Lisp. This version of CAST.FSM, called macCASTfsm \cite{6}, has been in use as a CAST tool at our institute until the year 2000. In addition to the previous CAST.FSM implementations, macCASTfsm also offers new implemented methods as the inversion of finite state machines, the representation of finite memory machines in canonical form, and the shift register representation of finite state machines by the method of Böhling as done by Josef Scharinger in \cite{13}. Due to the incompatibility of macCASTfsm with the latter operating systems of the Apple MacIntosh the use of macCASTfsm on appleMacIntosh computers got obsolete. Therefore, after the introduction of the ANSI Common Lisp standard and since associated Common Lisp environments became available, we had the task to realize a portation of macCASTfsm. Rudolf Mittelmann chose Allegro Common Lisp from Franz, Inc. and also did the implementation. This new version of CAST.FSM for Windows PCs, called winCASTfsm, presently replaces macCASTfsm as a rather powerful and complex CAST tool since 2000.

2 Hierarchical Models of A. Koestler (Holarchies)

Arthur Koestler, a well known writer but also an author of science-oriented books, proposed a concept for modeling complex intelligent systems of hierarchical order by his ”holarchical nets” \cite{3}. The nodes (called ”holons ”by Koestler) of such a tree-structured system are assumed to model intelligent components and are equipped with an individual rule base for an autonomous realization of strategies. The introduction of the concept of a ”holarchy” was mainly motivated by his studies of the human brain and the organisational structure of a company. Nowadays, motivated by the concept of agent systems in computer science, it is appropriate to consider a SOHO-structure in the sense of A. Koestler as a special kind of an organized multi-agent system (OMAS) as considered by Ferber \cite{1}. Our ultimate goal is to formalize and to refine A. Koestler’s concept of a ”holarchy” (SOHO-structure) in order to achieve a system-theoretical model which can be classified as ”operational”. Desired properties of such models are ”self-regulation” and ”self-repairment”. For a mathematical approach to holarchies and for biographical notes to A. Koestler we refer to \cite{4}.

3 The Multi-strata and the Multi-layer Representation of a FSM’s Parallel Decomposition

Generally speaking, decomposition of finite state machines (FSMs) deals with the problem of how a machine can be replaced by more than one simpler machines,
i.e. a single FSM is transformed into a network of FSMs. In the case of a parallel decomposition of a FSM, a single machine $M$ is divided into a certain number of submachines whereby the parallel connection of those smaller submachines $M_i$ ($2 \leq i \leq n$) simulates the original finite state machine $M$. Consequently for each of the so obtained submachines $M_i$ a parallel decomposition can be obtained by splitting $M_i$ into $M_{ij}$ ($2 \leq j \leq n$) and so on. The various possibilities of potential parallel decompositions are given by the lattice of the machine or submachine that is actually been taken into account. For further theoretical details the reader may be referred to the well known book of Hartmanis and Sterns [2].

Fig. 1. Multi-strata representation of a FSM parallel decomposition.

Fig. 1 gives an example of a parallel decomposition of a FSM $M$ into three submachines $M_1$, $M_2$ and $M_3$ whereby $M_1$ and $M_3$ further decomposed into $M_{11}$, $M_{12}$, and $M_{13}$ respectively into $M_{31}$, $M_{32}$, and $M_{33}$ and so on. In this multi-strata kind of representation, that is commonly used in the description of FSM parallel decompositions, each level represents the whole system with the restriction that the leaves of each level have to be thought as members of the subjacent levels in order to do so. IN [10] it has been shown, how in general a hierarchical structure (which is required for a holarchy in the sense of A. Koestler) can be derived from a multi-layer representation. In our case of FSMs we have to choose a node, i.e. a submachine, at each layer that will be considered as the direct superior of the other nodes (submachines) being situated in the same layer.

Fig. 2 shows the adaption of the exemplary representation of the FSM’s parallel decomposition given in Fig. 1 to the desired (holarchic) interpretation as an intermediate step where the multi-strata representation is fully included and the holarchic model is represented in the bold highlighted part of the figure.
Fig. 2. Intermediate step between the multi-strata and the multi-layer (holarchic) representation of a FSM’s parallel decomposition.

Fig. 3. Multi-layer representation of a FSM parallel decomposition.

Fig. 3 shows the achieved holarchic multi-layer representation of the introduced multiple parallel decomposition. This is a pure multi-layer representation, i.e. only the components of all layers together model the whole system. As it can be seen in our context, the nodes of the multi-layer model are exactly the leaves of the multi-strata model. Therefore, it is an obvious specification for the design of such a holarchic model that each subtree emerging by multiple parallel decomposition has to be built up in a way that exactly one of the newly emerging submachines will become the direct superior of the other nodes. As a rule for the destination of this superior-machine it is reasonable to choose that submachine that has the least number of states on order to perform leader tasks.
Furthermore, the structure given in Fig. 3 will exactly describe the example of the parallel decomposition of the MCNC benchmark machine which will be shown in the following chapter using CAST.FSM.

4 Realization of a Hierarchical Representation of a FSM by winCASTfsm

In the following we will describe the formation of a finite state machine hierarchy on the example of the bbsse-e MCNC benchmark machine [7] as an application of the latest version of winCASTfsm.

In order to perform a parallel decomposition of a finite state machine it is necessary to compute the lattice of the machine. winCASTfsm supports this feature and the computed lattice is finally displayed as a Hasse-diagram in a new window, a so called lattice browser. Within such a lattice browser, groups of nodes may be selected in order to perform certain lattice browser commands like 'Add Partitions' or 'Multiply Partitions' for showing the supremum respectively the infimum of a selected collection of partitions. According to the theory of FSM decomposition we have to select a group of nodes whose infimum is 0 in order to be able to perform a pure parallel decomposition.

![Lattice of the bbsse-e benchmark FSM](image)

**Fig. 4.** Lattice of the bbsse-e benchmark FSM.

Fig. 4 shows the lattice of the bbsse-e benchmark machine whereby those partitions which were selected for the parallel decomposition are highlighted. The bracket term includes the number of blocks, i.e. the number of states of the corresponding submachine, and the number of states united by the biggest block. For example the partition with number 28 has three blocks, and the biggest block unites ten states (hence the label '28(3b,10m)'). According to Fig. 5 the nodes 3, 24 and 14 correspond to $M_1$, $M_2$ and to $M_3$ respectively. Having performed the synthesis operation 'Parallel Decomposition' on the se-
lected partitions, winCASTfsm shows the result as an expansion of the realization graph allowing a successive implementation of the described methodology. As indicated in Fig. 5, the realization graph of the bbsse-e benchmark FSM, the machine is divided into three submachines each named by its engendering partition (bbsse-e-r24, bbsse-e-r10 and bbsse-e-r13). By an analysis of the lattices (Fig. 6, Fig. 7, Fig. 8) of the submachines it seems reasonable to select the bbsse-e-r24 as the superior of the remaining submachines. Those remaining submachines will recursively be taken into account for further parallel decompositions as illustrated in the realization graph (Fig. 9). According to the holarchic multi-layer representation given in Fig. 5 the lattice given in Fig. 6 denotes the lattice of $M_2$ and the lattices of Fig. 7 and Fig. 8 denote the lattices of $M_1$ and $M_3$ respectively. The latter both are each split (parallel decomposition) into three submachines whereby one of each triple ($M_{12}$ and $M_{32}$ in the notation of Fig. 2 and Fig. 3) is selected as the superior of the others. Subsequently the remaining submachines ($M_{11}, M_{13}, M_{31}, M_{33}$ in the notation of Fig. 2) are each split into three 'sub-sub-submachines' ending in the final holarchic multi-layer model given in Fig. 4.

Depending on the task and on the FSM actually taken into account we would achieve different structures with regard to the width and the depth of the fi-
Fig. 6. Lattice of the submachine derived from partition 24. This submachine is the selected superior and will therefore not be taken into account for further decompositions.

Fig. 7. Lattice of the submachine derived from partition 10. The highlighted partitions indicate the partitions that are selected for a further parallel decomposition.

5 Conclusion

The present paper briefly introduces the new version winCASTfsm as the latest version of the CAST tool CAST.FSM which has been developed at the department ”Systems Theory and Information Technology” at the ”Institute of Systems Science” of the University of Linz. In order to demonstrate the power of winCASTfsm we have performed a multiple parallel decomposition of the bbsse-e...
Fig. 8. Lattice of the submachine derived from partition 3. The highlighted partitions indicate the partitions that are selected for a further parallel decomposition

MCNC benchmark FSM into a hierarchical (multi-layer) representation including the definition of a decision-order.
CAST.FSM is currently in use at the University of Linz as a teaching aid. Furthermore, it is provided as a CAST tool for layout- and the analysis-tasks in application areas such as micro electronics, signal-processing, and coding (development of cryptographic algorithms).

References

A Holarchical Organized Design Assistant for Hierarchical Decompositions

Sebastian Meine
SIEMENS AG, 81739 Munich, Germany

Abstract First the ideas of Arthur Koestler about hierarchical organizations, so called Holarchies are introduced. Then applications of Holarchical Networks are discussed. As an example the decomposition of automata as needed for the design of ASICs is investigated. For this, an OOP framework of classes for easy implementation of Holarchical Networks is developed.
Conclusion: Holarchical Networks can be used to control parallel processes and to simplify design decisions.

1 Holons and Holarchies

Arthur Koestler derived in his work 'The Ghost in the Machine' the following terminology:

- SOHO/OHS (Self-organizing Open Hierarchical Order / Open Hierarchical System)
- Holon (essential part of a SOHO)

A possible form of a SOHO is the Holarchy. Arthur Koestler wrote about Holarchies:
'The organism is to be regarded as a multi-levelled hierarchy of semi-autonomous sub-wholes, branching into sub-wholes of a lower order, and so on. Sub-wholes on any level of the hierarchy are referred to as holons.

...holons are self regulating open systems which display both the autonomous properties of wholes and the dependent properties of parts. This dichotomy is present on every level of every type of hierarchic organization, and is referred to as the Janus Effect...'

A Holarchy therefore is a strong hierarchical ordered network of Holons. A Holon in this context is a half autonomous entity: From each predecessor it gets a task and distributes subtasks to selected successors. From the successors it collects sub-results, combines them to a complete answer and send this back to it predecessor. Therefore, a Holon is told what to do, but not how to do the task. Further communication between Holons in the entire network is not allowed.
1.1 Holarchies

The following is a possible mathematical definition of a Holarchy:
Let $Q$ bet a set of questions and $A$ a set of answers.

**Definition 11**

A finite holarchical network is a pair $N := (H, R)$ where

- $H \neq \emptyset$ is a finite set of Holons
- $R \subseteq H \times H$ is a tree relation

There exists one Holon in the net which is responsible for the communication with the outside world which has a number of Subholons. This Holon is called Main-Holon. All other Holons in the entire network have exactly one predecessor and eventually some successors. Further connections are not allowed.

**Definition 12**

*Continuation of Definition 11.*

In this context a Holon $h$ is defined by:

$$h : Q \to A$$

$$h(q) := C_h(M_h(q), q), \text{ where}$$

- $M_h : Q \to \mathcal{P}(hR)$  
- $Q_h : Q \times hR \to Q$
- $C_h : \mathcal{P}(A) \times Q \to A$

Let $m(N) \in H$ be the Holon with $Rm(N) = \emptyset$ (Main-Holon).

Each other Holon in the net takes a task from its predecessor, delegates a sub-task to each selected successor, collects their answers and builds therefrom the common answer for its predecessor.

Thus the procedure works as follows: Using the function $M_h$ the Holon selects the relevant successors dependent on the given task. Then it defines the subtasks for them using the function $Q_h$. After collecting the answers it forms the answer for the predecessor or the main-answer for the outer world in case of the Main-Holon.

1.2 Extensions

Possible extensions for this theory are:

- learning Holons (adaptation of the function $C_h$)
- adaptation of the connecting paths (of the relation $R$)
– mechanisms for the integration of additional Holons during the process (holons on demand)
– input buffers for Holons for later asynchronous execution of tasks
– additional direct or indirect communicating ways for Holons which are not connected by $R$

**Holons on Demand** As we will make use of this feature, we give a short description of *Holons on Demand*.

**Definition 13**

A finite holarchical network with Holons on Demand is a tuple $N := (H, R, d)$ where

- $H \neq \emptyset$ is a finite set of Holons
- $R \subset H \times H$ is a tree relation
- $d$ is a function on the set of all holarchical networks (in terms of Definition 11)

A Holon $H$ here is a mapping:

$$h : Q \rightarrow A \times \{\kappa | \kappa \text{ is a description of a Holon}\}$$

$h(q) := (C_h(\{g(Q_h(q, g))|g \in M_h(q)\}, q), \zeta_h(q))$, where

- $M_h, Q_h, C_h$ as above
- $\zeta_h : Q \rightarrow \{\kappa | \kappa \text{ is description of a Holon}\} \cup \{\lambda\}$

The result of $\zeta_h$ is processed asynchronously: As soon as $\zeta_h$ is different from $\lambda$ for a Holon $h$ in the net, $d$ produces a copy of the actual holarchical network in which $h$ has an additional successor corresponding to the description $\zeta_h$. Then the complete process is transferred to the new network. $h$ can herein directly use its new successor.

**2 Organization of a Design Assistant as a Holarchy**

Each problem, that can be splitted into more or less independent subproblems is a good input for a Holarchical Design Assistant. For example, formula or automatata decomposition are such problems.

### 2.1 Formula Decomposition

As an easy example we will consider the decomposition of simple formulas, here a mathematical expression in braced prefix notation:

$$(+ (* (- 4 3) 6) (/ (- 37 2) 7))$$
To handle such problems you need a Holon type that can deal with brackets and one for each possible operator. The Main-Holon has to be of bracket type. At the beginning there are not more Holons needed, if we use the Holons on Demand feature. This is a good idea, as the structure of the expression is not known and therefore the alternative is to build a ‘complete’ network, which will have much overhead.

The task of a ‘bracket’-Holon is to check the format of the expression: First there has to be an opening bracket, then an operator, then a list of operands which can be numbers or braced subexpressions and finally the closing bracket. Braced subexpressions must be forwarded to bracket type Subholons. When all subresults have been calculated, the ‘bracket’-Holon gives all operands to a successor of adequate type. The result then has to be returned to its predecessor.

The Task of the Operator-Holons should not need further illustration.

### 2.2 Automata Decomposition

In the following, we will discuss how Holarchies can help to investigate serial and parallel decompositions of automata.

A Holarchy that is able to decompose a finite automaton, may have the following types of holons:

- **R**: Holon that is able to calculate the state-reduced equivalent of a given automaton
- **S**: Holon that is able to calculate a serial decomposition of a given automaton
- **P**: Holon that is able to calculate a parallel decomposition of a given automaton

The Holarchy then should have a form as shown in Figure 1.

Here the Main-Holon (marked with \(M\)) has a successor Holon (\(R\)) that makes a state reduction. Then a Holon (\(P\)) follows, that is able to parallelize the automaton. Next, there are Holons for further state reduction and then serialization Holons (\(S\)). This structure is repeated as often as needed. It is important to check the possibility for parallel decomposition first. The reason is, that if parallelization is possible, serialization also is. But not vice versa.

The Main-Holon gets as input an automaton and gives it to its successor, which is the first reducing Holon. Then the automaton goes through the net being reduced, serialized and parallelized. If a Holon realizes, that there is no further nontrivial splitting possible, it marks the automaton part as not further serializable or parallelizable. If a decomposition Holon gets a marked automaton and is able to split this automaton, it deletes the mark and the process goes on. If not, the process for this part has been finished. The Holon returns this part together with its position to its predecessor. The Main-Holon collects all parts and puts them together in the right order.
For each step in this network, it is necessary to know the decomposition lattice of the actual automaton. There are two possible ways to reach this:

- each Holon has the possibility to calculate the lattice or
- to each Holon an additional successor for this task is added.

As a further extension of the model, we can introduce 'jumping Holons' at this point. As each Holon needs its lattice calculating successor for a short time, only a few Holons of that kind are needed. This Holons than can move through the net on demand.

Note, that the ready calculated lattice can be use by the successors, if a Holon has not changed the actual part.

Each decomposition Holon could also have more than one pair of successors, so that the network is able to process more than one decomposition at a time (if there is more than one decomposition possible). On the way back each decomposition Holon has to select the decomposition which has produced the best fitting automata, according to the given constraints.

At this point we can introduce the 'Holons on Demand' feature: At the beginning only three Holons (M, R and P) exist. Each decomposition Holon orders appropriate successors, if needed.
Such a Holarchy is able to find the best fitting decomposition of an automaton according to given constraints just by calculating all possibilities in parallel. It can make design decisions by itself and does not need the interaction with the user.

2.3 Conclusion

The Holarchical Design Assistant offers a way to describe and control the high amount of parallel processes needed to compute an adequate automaton decomposition. For this special Holons which make design decisions according to given constraints can be designed. Examples: wafer size, the shape of a left place on the wafer, the number of transistors left on a wafer, etc.

Using the Holarchical Design Assistant the calculation time is reduced if real parallel computing is possible. Here no operator is needed to control the process so that even unexperienced users can do the task.

3 Arthur Koestler: Life and Work

Arthur Koestler was born in Budapest on September 5, 1905. His parents were Henrik K. Koestler, an industrialist and inventor, and Adele Koestler, born Jeiteles. Although both parents has been Jewish, he grew up religionless.

1922 he began studying engineer sciences at the Wiener Technische Hochschule. One year later he joined a zionistic student league. This induced his break of study in 1926. He went to Palestine to live in a Kibbutz.

After one year of steeliest work (he earned his sustenance as a field worker), Arthur Koestler became a corespondent for German newspapers. He changed to Paris 1929 and only one year later he was taken in by the Ullstein Verlag. There he went ahead in a very short time. He became science editor of the 'Vossische Zeitung' and external editor of the 'B.Z. am Mittag'.

1931 Arthur Koestler joined the KPD. This was uncovered by his employer, resulting in his dismissal.

1932 - 33 Arthur Koestler made a journey through the Soviet Union. In the meantime his membership in the KPD became passive.

After that he lived in France, working free-lance as a journalist. Among others he worked for the Parisian 'Zukunft', a weekly making propaganda against Hitler and Stalin.

1936 Arthur Koestler became commentator in the Spanish civil war for the 'New Chronicle'. 1937 he was captured by Franco's troops and imprisoned under the
sentence of death. In the prison of Seville he wrote the ‘Spanish Testament’. After four months he was released on the intervention of the British Government and returned to London.

During the war he served with the French Foreign Legion and the British Army and in 1945 became a Special Correspondent for ‘The Times’ in Palestine. In the 1940s and early 1950s he was perhaps the most widely read political novelist of our time.

Arthur Koestler received the Sonning Prize from the University of Copenhagen in 1968 and was awarded a number of honorary doctorates. He was a Fellow of both the Royal Society of Literature and of the Royal Astronomical Society. He was made a CBE in 1972 and a Companion of Literature in 1974, and on three occasions was nominated for the Nobel Prize.

Likewise he was married three times: 1935 to 50 with Dorothy Aher, 50 to 52 with Mamaine Paget and 65 to 83 with Cynthia Jefferies.


For his work ‘Darkness at Noon’, written in 1940, Arthur Koestler earned international fame.

Since the 1950s he worked on science and philosophical themes. 1955 he pronounced his literary-political career finished.


References

Conflict Management in an Intelligent Multiagent Robotic System Based on FSM Approach

Witold Jacak, Karin Pröll, and Stephan Dreiseitl
Dept. of Software Engineering
Polytechnic University of Upper Austria at Hagenberg
Hauptstrasse 117, A 4232 Hagenberg, Austria

Abstract. This paper presents a system-theoretical approach for an intelligent multiagent system which is used to control and coordinate a society of robotic agents acting independently in a partially known environment. We focus our presentation on only one aspect of the coordination of the robotic agent group, namely on the conflict management. We present solutions for detecting and resolving conflicts between two or more robotic agents.

1 Introduction

Within the last few years the paradigm of intelligent software agents has become a mainstream topic in research activities in various fields. A multiagent system can be understood as a society of heterogeneous cooperating entities with varying degrees of intelligence. Such a group of agents is directed to achieve a common goal, while at the same time they have no or only few knowledge about the environment they are acting in. Agents are able to change their behaviour reacting on environmental changes by using their own specific knowledge and by their ability to communicate with one another. An intelligent agent can send and receive information to and from other agents using appropriate protocols, generates multiple objectives, goals and plans for itself and other agents. It processes information received and performs reasoning. These require services that support rich knowledge representation, maintain explicit belief models of itself and other agents and can reason with incomplete, inconsistent and uncertain information. It has a set of capabilities (which can change dynamically) - i.e. the tasks it can perform - and can reason about its own and other agents capabilities and skills (e.g. planning, communication and negotiation). This requires a task specification language, and mechanisms that support learning. It should be able to assume roles and perform tasks and execute physical and non-physical actions - actions can result in events which in turn may trigger other actions and processes. This requires task selection and task execution mechanisms. Agents can be engaged in complex interactions with each other, such as negotiation and task delegation and dynamically join or leave groups or organizations. This requires support for specification of organizational structure, organizational procedures.
Conflict Management in an Intelligent Multiagent Robotic System

An agent-based robotic system consists of a community of independently acting robotic agents. Each of the robotic agents is under control of an intelligent software unit. The intelligence of those software units comes from the ability to solve sub-problems locally and to propose a global solution as a result of the interactions between different agents. Each robotic agent has its own goals and an autonomous behavior which is a result of its observations, its knowledge and its interactions with other agents. The agent group creates the first level of the multiagent system, i.e., the operational (execution) level.

The main goal of a multiagent robotic system is to solve a common task which is split up into several subtasks that are distributed by the highest level of the robotic system to the individual robotic agents. The coordination and synchronization of the realization of subtasks is performed by the second level of the robotic system, i.e., the management level.

2 Multiagent Robotic System

A Multiagent robotic system is a group of cooperating agents acting in a common environment coordinated by two management agents on a higher level, the contract and conflict managers. Cooperation between agents have a different focus. The main task is to solve complex problems by using a communication and cooperation mechanism with other agents or external resources. Cooperation is used when a problem (task) exceeds the capabilities of one individual agent or if an agent exists that already has a solution and whose knowledge can be used by the other agent. Each agent has goal-oriented behavior.

The primary goal of a multiagent robotic system is to solve a common task which is distributed into several individual tasks of the robotic agents. Task distribution on the one side and task performance on the other side require two different agents of the management level of the multiagent system.

The first management agent, called Contract Manager, considers task distribution when a new job enters the system. The Contract Manager has to direct autonomous agents by specifying individual goals (subtasks) for each of them. This can be done in two different ways. The Contract Manager can distribute subtasks to agents in a hierarchical way by simply assigning a subtask to an individual agent, or he can offer subtasks as service requests to the whole agent community. This entails a bidding process between contract manager and robotic agents in a market-like style.

The second agent on this level, called Conflict Manager, indicates cooperation and negotiation while each agent performs its assigned subtask. As the individual...
behavior of all robotic agents involved in task achievement cannot be predicted in advance, the goals of two or more robotic agents can be in direct conflict with one another and the achievement of the whole task is endangered. In order to resolve a conflict situation, a negotiation process among all conflict parties has to take place. The result of the negotiation should be a solution which results in goal achievement for all agents involved. A multiagent robotic system is presented in Fig. 1.

The robotic agent is constructed with many components such as software components and technical (hardware) components. In this article we focus our presentation on a robotic agent which is equipped with an NC-controlled manipulator with effector and different sensors — a hardware component and four software components. The structure of such a robotic agent is presented in Fig. 1.

The behavior of the overall multiagent system can be observed as the set of actions performed by each individual agent. Each agent follows its own specific goal. different dependencies can occur between the goals of agents in a multiagent system. The set of the all goals can change over time. The robotic agents should realize different manipulator’s motions to achieve the common goal of the system, which is the realization of a technological task consisting of many separate jobs. On the management level of a multiagent robotic system, the currently active jobs are decomposed into several motions and the contract manager decides which robotic agent has to realize the movement. The contract manager of the robotic agent group assigns the final position of motion for each agent. These final positions create the set of individual goals of the system in the current time instance. The principal task of the intelligent robotic agent is the execution of the given path of movement so that the robot action does not result in a collision with currently active dynamical objects (such as other robotic agents) in the cell. While the robot is tracking a preplanned path, some extraneous dynamic objects may enter the work space. These dynamic obstacles in the surrounding environment of the robot are supposed to be detected by means of
sensors mounted on the links of the manipulator (a part of the hardware agent). In our case the objective of sensor manipulation is preventive conflict avoidance.

3 Intelligent Robotic Agent

An agent architecture is determined by the agent’s structure and behavior. The agent structure is determined by the set of agent’s units called components, the set of agent input and output relay ports (the interface), the set of connections between agent and other agents in the multiagent system and the knowledge base, the set of models and methods needed to decision making. The behavior of an agent will be specified by its state transition engine, which is the complex state machine over so-called composite states. Each intelligent robotic agent consists of several cooperating and independent components (units) such as knowledge base, components for action planning and action execution, component for safety protection, hardware component, and communication and negotiation component [9, 8, 5].

3.1 Knowledge Base

The knowledge base of an agent builds a formal representation of the agent’s specified data that are needed for decision making. The typical knowledge base contains different models and their formal representations: a world model, a social model, a mental and a self model. The world model represents knowledge about the surrounding environments of an agent, the social model represents knowledge about other agent acting in the system, the mental model represents the knowledge about risk by decision making and their consequences and the self model contains the knowledge about construction, properties and structure agent’s hardware component, for example the robot’s manipulator [8, 10].

World Model: The knowledge represented here is the geometrical model of the robotic agent environment [8, 10]. Many different methods can be used for the geometrical representation of the agent service space. One of them is the triangle approximation, another is cubic approximation. In a model with triangle approximation, the points in the triangle net (lying on service space border) are joined to form triangle walls. The walls represent the data objects of the world model in the knowledge base. This model is modified based on sensor data. The other model describes the service space of the robot manipulator as a cubic approximation [8].

Depending on this knowledge, different self-models can be used to perform the negotiation and coordination process among concurrently acting agents. The knowledge level leads to two different methods of negotiation or synchronization in a conflict situation. In the case of only partial knowledge about the surrounding world, the method is called the negotiation among concurrent acting agents and in case of full knowledge about static surrounding world the method is called the synchronization of concurrent acting agents.
Self-model: Model of the Agent’s Actor The self model contains the knowledge about construction, properties and structure of the hardware component of the agent. Here, knowledge of the kinematical properties of the robotic agent is provided in order to decide about collision avoidance mode and avoidance path. Therefore, the forward and the inverse kinematic models of the robot should be known. These models can be created in different ways using different formal tools as e.g. symbolically computed models, numerical models or a neural network-based model. Depending on the knowledge level, different models of plant can be used for action planning. In case of only partial knowledge about the surrounding environment we can apply the learning-based neural network model of agent actor, but in the case of full knowledge about the static world we can use the finite state machine model of agent actor.

Neural network based model of robotic agent kinematics The planning of the new configuration is based on the computation of robot kinematics and is computationally expensive. Therefore it is attractive to apply a neural network model of robot kinematics, stored in the knowledge base of the agent, which automatically generates safe configurations of the robot. This model uses a multilayer feedforward neural network with hidden units having sinusoidal activation functions. Based on the neural direct kinematic model it is easy to generate the inverse kinematics.

Finite state machine model of robotic agent kinematics The most suitable model of hardware component of robotic agent is a discrete dynamic system defined as:

\[ RA = (Q, U, Y, f, t) \]  

where the set \( Q \) denotes a set of inner states of the agent actor and the state \( q = (q_i | i = 1, .., n)^T \), \( U \) denotes a set of input signals of \( RA \). An output \( Y \) should ensure a possibility of the geometric representation of the robot’s body in 3D base frame (representing the surrounding world frame). For this purpose it is convenient to use a skeleton model of the agent’s manipulator described as the vector \( y = (P_i | i = 0, 1, .., n)^T \), where \( P_i = (x_i, y_i, z_i) \in E_0 \) is the point in the base coordinate frame describing the current position of the \( i \)th joint and \( P_n \) is the position of the effector-end. The function \( f : Q \times U \rightarrow Q \) is the one-step transition function, and \( t : Q \rightarrow Y \) is an output function of the form

\[ y(k) = t(q(k)) = (t_i(q(k)) | i = 0, 1, .., n)^T \]  

where \( t_i = P_i \) represents the Cartesian position of \( i \)th joint. The properties of such a specification of agent kinematics depend on the method of specification of its components, specifically on the input set \( U \).

One of the ways to construct such a model of robot’s kinematics is based on an arbitrary discretization of angle increments of the agent mechanical joints. In order to specify the input set \( U \) of the model \( RA \) the discretization of the agent’s joint space \( Q \) is performed.

\[ q_i = q_i^{min} + j \cdot \delta q_i \quad \text{and} \quad j \in \{0, 1, .., J_i\} \]
Using the fact that all the angles can change only by a predefined increment, we define the input set $U$ of the model $RA$ as:

$$U = \times \{u_i|i = 1,..,n\}$$ (5)

where $u_i = \{-\delta q_i, 0, \delta q_i\}$ is the set of possible (admissible) directions of change of the $i$th joint angle. Having defined the set $U$, it is possible to describe the changes of successive configurations of the agent’s link as a discrete linear system of the form:

$$q(k + 1) = q(k) + \Lambda \cdot u(k)$$ (6)

where $u(k) \in U$ is the vector of increments of angles of the joint and $\Lambda = \text{diag}[\Delta_i]$ is the diagonal $n \times n$ - matrix describing the length of the angle’s step changes at each joint.

In order to make it possible to check the configuration with respect to obstacle locations, it is necessary to create an output function $t$. As we have stated previously, the agent manipulator’s position in the base frame is represented by a skeleton of an agent arm. Recall that the $i$th joint’s position in Cartesian base space, assuming that all the joint variables $q_i$ are known, is described by the Denavit-Hartenberger matrix $T_i$ [10]. The last column of the matrix can be used to determine the output function of the model $RA$ as:

$$t(q(k)) = (P_o, t_i(q(k))|i = 1,..,n)^T$$ (7)

where $t_i(q(k)) = P_i$ is the element of the last column of the matrix $T_i$.

The discrete model of robotic agent kinematics with $n$-degrees of freedom presented above can be simplified when agent has a planar manipulator [8].

### 3.2 Safety Protection Component of $i$–th Agent

The goal of the safety protection component is to provide a collision-free movement of the robot manipulator.

For collision detection and avoidance, we propose the installation of ultrasonic sensors and a sensitive skin on each manipulator link and then use a neural network to estimate the proximity of the objects with the link of question. The resulting distances are compared with the local model of the robot environment to recognize the new obstacle within the radius of the security zone $\rho$ of the agent.

When the obstacle penetrates the security zone, the component calculates the changes of robot configuration to avoid the obstacle in the best possible way. To calculate the safe configuration, the planner uses the distance vector $d$ and combines it in the inverse kinematics computation, performed by the neural network based inverse calculation method from the agent’s knowledge base [8,10].

The obstacle avoidance can be achieved by introducing additional errors for each joint, i.e., the errors between virtual points $p_i$ and the joints positions $t_i(q)$ ($e_i = [p_i - t_i(q)] = -\frac{\alpha \cdot d}{||d||}$) into the inverse calculation method, where the virtual points $p_i$ represent the preferred position of the $i$th link that achieves
collision avoidance. The virtual points are placed on the opposite side of the joint with respect to the obstacle. As a result of the inverse calculation method, the solution of the inverse kinematics with obstacle avoidance is obtained. Because computations are initialized with the desired positions of the manipulators joints (virtual points), the real manipulator’s configuration reflects the correction of motion $\Delta q^i$ which tries to avoid the obstacles and performs the detour action. The correction vector $\Delta q^i$ is added to the currently realized motion by the execution component.

The component for action planning and action execution of the robotic agent will be described in the next section.

4 Action Planning of Multiagent Robotic System

The discretisation of values taken on by the joint angles results in the finiteness of the joint space $Q^i$ of the $i$th agent. The set of configurations $Q^i$ will be taken as the state space of the FSM modeling changes of the agent kinematics. The contract manager on the coordination level of the robotic agent group assigns the current final position of motion $P^f_i$ for the $i$th agent. This final position determinates the set of individual goal states $G^i$ of $RA^i$ in current moment of the time as

$$G^i(t) = Q^i_f = \{q \in Q^i | t^i_n(q) = P^f_i \},$$

where $t^i_n(q)$ is direct kinematic model of agent $RA^i$. Each movement has priority level $\pi^i$ dependent on the priority of the job it belongs to.

Let $M$ be a set of agents acting in a common works space and $M_a$ the subset of agent that currently realize their job (active agents); $M - M_a$ is the set of free agents. Let $\pi^i$ be the priority level of the actions (depended on the priority of the goal) of $i$th robotic agent with $q^i_c$ its current state and $P^f_i$ its final pose, which generates the set of the goal states $Q^i_f$ in its $Q^i$ state set. Each agent has the security zone $\rho^i$ (needed for safety protection component) and a status free or busy.

The problem of finding a safe motion for all active agents appears to be a problem of reachability of the set of goal states from the start state for all agents. Let $QS = X\{Q^i | i = 1, .., M\}$ be the global state space of the robotic agent system with the global state transition function $\lambda : QS \times US \rightarrow QS$, where

$$\lambda((q^i | i = 1, .., M), (u^i | i = 1, .., M)) = (f^i(q^i, u^i) | i = 1, .., M).$$

In the current moment $\tau$ the start state $qs \in QS$ is described by the vector of current configurations of the agents in their state space, i.e., $qs_c(\tau) = (q^i_c(\tau) | i = 1, .., M)$. For robots which currently realize assigned motions (status busy) there exist the goal sets of states $Q^i_f$ and for robotic agent with status free, the goal set is equal to the whole set of states $Q^i$. This allows to define the current goal set of the whole robotic system as

$$QS_f = \{(q^j | j = 1, .., M) | q^j \in Q^j_f \ if \ status_j = \text{busy} \ or \ q^j \in Q^j \ if \ status_j = \text{free}\}$$

(10)
The problem can be expressed as reachability problem of the set $QS_f$ from the current global state $qs_c$ via feasible states, i.e., find the sequence of global states and the sequence of global inputs $qs^* = (qs_c, .., qs(k), .., qs(N))$ and $us^* = (us(0), .., us(k), .., us(N-1))$ such that $qs(k+1) = \lambda(qs(k), us(k))$, and $qs(0) = qs_c$ and $qs(N) \in QS_f$ and all the states $qs(k)$ are feasible, where $qs(k) = (q^i(k)|i = 1, ..., M)$.

4.1 State Feasibility: Static and Dynamic Constraints

Not all configurations of the $i$-th agent $RA_i$ can be feasible. We can define the constraints function $C_i : Q_i \rightarrow \mathbb{R}^n$ that decides if the configuration $q^i$ is feasible. This function is preferred to describing the static constraints such as collision freeness with static obstacle in the agent’s surrounding world.

Static constraints: Collision freeness testing. The typical constraints function describes the collision-freeness conditions between agent and surrounding world. We can say that the configuration $q^i$ is collision free if it does not collide with any static obstacle $OB$ in the world. To test this condition, we should have full knowledge about the geometry of the surrounding static world, i.e., the geometrical model of agents environment should be completely known.

The space occupied by the agent manipulator at configuration $q$ can be approximated by rotary cylinders whose axes are individual links of the skeleton model. In order to check if the manipulator moves in a collision-free way, only on the basis of its skeleton model, let us extend the static obstacles in each direction by the value of the maximal radius of cylinders that approximate links. To obtain fast and fully computerized methods for collision detection, we use an additional geometric representation of each static object on the scene. We introduce the ellipsoidal representation of 3D objects, which uses ellipsoids for filling the volume. Checking for the collision-freeness of the agent configuration can be reduced to the "line-ellipsoid" intersection detection problem, which in this case has an easy analytical solution.

Dynamic constraints: Conflict freeness testing The constraints function presented above does not determine if there occurs a collision between dynamic objects on the work scene. If the distance between two agents is less than the safety zone, this will indicate a conflict between these agents. Additionally to the local configuration collision freeness test for each agent separately, the feasibility of the global state can be expressed by introducing an additional measure of conflict detection between states of different agents defined on the union of the state sets of each agent $d : \cup Q^j \times \cup Q^j \rightarrow \mathbb{R}$. We can say that the global state $qs = (q^j|j = 1, ..., M)$ is conflict free if $(\forall k, j \land k \neq j)(d(q^j, q^k) \geq \rho^j)$, i.e. the dynamic constraints are satisfied. The dynamic constraints are dependent on the current position of the agent during movement and its change over time. The configuration or global state $qs$ will be called feasible when it is collision and conflict free.
4.2 Strategies of Searching of Safe Actions of Robotic Agent Group

In order to solve such complex reachability problem we are going to sequentially apply the graph searching procedure to the state transition graph of the global state set $QS$. Expanding the current state $qs_c$ with function $\lambda$, successors of $qs_c$ etc. ad infinitum, makes explicit the graph that is implicitly defined by the current state and transition function. As evaluation function, we can use the weighted sum

$$e(qs) = \sum_{i=1}^{M} e^i(q^i) \quad (11)$$

of the cost functions $c^i(q^i_c, q^i)$ and a cost estimate function $h^i(q^i, q^i_f)$ for each active agent and distance function for all free agent, i.e. $e^i(q^i) = c(q^i_c, q^i) + h(q^i, q^i_f)$ for agents with status busy and $e^i(q^i) = -\min \{ d(q^i, q^j) | j = 1, \ldots, i - 1, i + 1, \ldots, M \}$ for agents with status free.

Using the standard A* procedure we can find the state trajectory (if it exists) $qs^*$ from the current state to goal state that includes only feasible (collision and conflict free) states.

The development of the search graph will start from the node (configuration) $qs_c$ by the action $\lambda$ for all possible input signals from the set $US$. Each agent’s manipulator has $n$ degrees of freedom. Every configuration of an agent therefore has $3^n - 1$ successors. This implies that every node of the global graph has $(3^n - 1)^M$ successors. Thus, it becomes essential to quickly check for the non-repeatability of the nodes generated, and their feasibility. It is easy to observe that

**Fact 1** If for all global states, the conflict do not occur then the global trajectory $qs^*$ is composed of the $q^*_i$ state trajectories of each active agent.

For each agent separately we can define the problem of achieving a state from its goal state set $Q^f_j$ as the problem of reachability of the goals state set from the agent’s current state. We can also apply the graph searching procedure in the state transition graph of the agent. The way of expanding the graph will depend on the form of the cost function used to evaluate each node. As evaluation function we can use the sum of the cost function $c(q^i_c, q^i)$ and a cost estimate function $h(q^i, q^i_f)$, for example the rectilinear distance between agent position and terminal position. To find the shortest path in the state-transition graph connecting the initial node $q^i_c$ and the final node $q^i_f$, such that $t_n(q^i_f) = P^f_i$, we use the A* algorithm \[8\]. The conflict relation can cause that the goal state set $QS^f$ can be empty. In this case the solution of the reachability problem does not exists.

To solve this problem, we propose the sequential decomposition of the global search problem into sequential search for each agent separately. This means that a solution that cannot be achieved simultaneously will be substituted by a sequence of partial solutions distributed over time. It is easy to proof the following fact.
Fact 2 Let for each agent exist

I the optimal trajectory \(q^*_i\) which achieves the goal state separately, based only on static constraints (collision freeness) and lack of conflict in start position of each agents, and

II an additional track \(q_{ir} = (q_{ic}^i, q_{i(2)}^i, \ldots, q_{ir}^i)\) of configurations that are not in conflict with configurations from the optimal trajectory of other agents and are not in conflict among one another, i.e.,

\[
(\forall i, j \in M \land i \neq j)(\forall q \in q^*_j)(\forall q' \in q_{ir}^i)(\neg q_{ij}^pq^q')
\] (12)

and

\[
(\forall i, j \in M \land i \neq j)(\forall q \in q^*_j)(\forall q' \in q_{ir}^i)(\neg q_{ij}^pq^q').
\] (13)

If assumption (I) and (II) are satisfied then the global reachability problem can be decomposed into \(M\) sequential synchronized problems such that each agent \(i\) realizes its optimal trajectory \(q^*_i\), when all other agents move to the conflicts free configurations \(q_{ir}^i\) sequentially. The global goal will be reached by the sequence of local goal realizations for each agent.

The solution proposed in the fact above is not optimal because it is realized sequentially and it does not force the negotiation between agents for conflict avoidance. It possible that there exist solutions which shorter time of realization. To search for better solutions we propose a new method based on synchronization of agents actions in case of conflict detection.

5 Conflict Management in Multiagent Robotic System

Depending on the level of knowledge, different plant and world models can be used to perform the negotiation and synchronization process among agents acting concurrently.

5.1 N-step Delay - Sequential Synchronization of Actions of Agent Group

Based on fact we show how the finite state machine model of the agent behaviour can be applied to solve the conflict avoidance problem by substitution of the concurrent performed planning and realization of actions of agents group by the sequentially performed one step planning and realization of the action of several agent from agent group (Fig. ??). Such round robin procedure for time synchronization of agent action leads to concurrent realization of actions with N-step delay and to solve the conflict avoidance by using the sequence time synchronized partial solution of goal achievement.

Based on current state of the job queues and messages from agents in the last full activity cycle, the conflict manager prepares the ordered list of agents AGENT and establishes their status, security zones \(\rho\), goals of motions, and the priority
levels of agents actions $\pi$. These priorities will be the basis for the coordination in case the conflict between agents. We will look for the best action of each agent, which does not cause the collisions and leads to goal states set. As we said above we are going to solve the reachability problem separately, by using the graph searching procedure sequentially starting of current configuration of each agent. For this purpose we shall exploit the state transition graph of the $RA_i$ generated implicitly by applying the function $f_i$ as production rules sequentially for each $RA_i$ from list AGENT. Let $RA^m$ be an agent from the list AGENT and is in current state $q^m_c$.

### 5.2 Action Planning and Action Execution Component of $i$–th Agent

The task of the motion planning component is to plan the safe configuration of the robot’s manipulator based on information coming from the conflict manager, the safety protection component, and the world model from knowledge base. The motion planning uses the off-line path planning method based on current knowledge about surrounding environment and the behaviour of agent’s own hardware component. This knowledge is represented as geometrical model of work scene and mathematical model of agent’s manipulator direct and inverse kinematics.

$i$–it Agent Actions Planning. The actions of $i$–th agent is activated by Conflict Manager which sent the message included agent’s goal position $P^i_f$ with priority level of its realization $\pi^i$, its security zone $\rho^i$ and status free or busy.

---

**Fig. 2.** Sequential coordination and synchronization
The action planner of the agent generates the new movement in following steps:

**step 1** The action planner requests the current positions of other agents in common environment,

**step 2** Based on the positions of other agent and its own status and the message from safety protection component, action planner recognizes its own current situation and establishes the parameters for motion search algorithm, such as type of evaluation function and type of feasibility testing function.

**Case: agent is busy and is not in conflict** If the agent current position is not in conflict with other agents and the status is busy then the evaluation function is the sum of cost function and heuristic function. The configuration is feasible if is collision and conflict free.

**Case: agent is busy and is in conflict** If the agent current position is in conflict with one or more of other agents and the status is busy then the evaluation function is in form $e^i(q) = -d^i(q) + \pi^i \cdot h^i(q)$ i.e. is the sum of negative distance function and heuristic function with weight value equal to the priority level $\pi^i \in [0, 1]$. The configuration is feasible if is collision free and only is not in direct conflict with other agents. The direct conflict between agents $i$ and $k$ occurs if $d(q^i, q^k) \leq \epsilon$. The $\epsilon$ is the minimal distance between agents where direct collision not jet occurs.

**Case: agent is free** If the agent status is free then the evaluation function is always in form of the negative distance function. The configuration is feasible if is collision free and is not in direct conflict with other agents.

**step 3** Action planner starts the state-graph searching algorithm $A^*$ from its current position $q^i_c$ with previously established evaluation and feasibility testing functions. The searching stops if the goal position is reached or if the OPEN set is empty or if the OPEN set has more as $N$ elements. To calculate the successors set planner uses the $RA^i$ FSM - model of the agent’s hardware component.

**step 4** The temporary path of motion is calculated i.e. $path^i_t = q^i_c \rightarrow q^i_b$ where $q^i_b$ is the best configuration in CLOSE set. Depending of the length and conflict freeness of this paths the new configuration of motion is chosen and realized.

**step 5** The new current state (configuration) is analysed and the adequate message is send to conflict manager. The massage include the information if agent has achieved its goal and/or if the conflict occur and/or if the agent is in deadlock position.

With sending the message agent ends his activity and wait for new command from Conflict Manager. The flow of action planning is presented in Fig. 2.

### 5.3 Conflict Avoidance Algorithm

The Contract Manager assigns jobs to agents and sent the jobs into the job queue of each agent. Based on current state of the job queues and messages from agents in the last full activity cycle, the conflict manager prepares the new ordered list
of agents AGENT and establishes their status, security zones $\rho$, goals of motions, and the priority levels of agents actions $\pi$. This ordered list will determines the sequence of agents in the next flow of full cycle of their actions.

For first to last agents in the AGENT list, conflict manager activates each agent sequentially. He sent the activated message to agent and wait on answer message. After fully cycle manager prepares new values of priorities which are used to order the new list AGENT. The coordination flow is shown in Fig. 2.

**Synchronization strategy** In order to establish the strategy for allocation the agent in the list AGENT the following parameters have to be calculated. The priority of each agent’s action should be calculated separately for each agent. It is based on the following factors: the current distance to finish position, the rest duration of the motion $\tau_r$, the current delay of the motion $\tau_d$. Let agent i has priority $\pi_i$ and status busy. Let $MaxD = distance(q_{start}, q_{final})$ and $D = distance(q_c, q_{final})$ then $\pi_i^{new} = (1 - \frac{D}{MaxD}) + \pi_i - (1 - \frac{D}{MaxD}) \cdot \pi_i$. If $\pi_i^{new} \leq \epsilon$ then $\pi_i^{new}$ is temporally equal to zero. If agent has status free then $\pi_i^{new}$ is allways equal to zero. Based on such new priorities conflict manager establish the security zones $\rho_i^{new}$ for each agent.

$$
\rho_i^{new} = \begin{cases} 
\max\{\rho_i^{min}, \rho_i^{new}\} & \text{if } \pi_i^{new} \neq 0 \\
\rho_i^{max} & \text{if } \pi_i^{new} = 0 
\end{cases}
$$

(14)

The new list AGENT is ordered with growing priority of each agent.

6 An Example

Let the agent group contains two agent acting in common workspace. In case as shown in fig. 3 does not exist the global solution of goal achievement. The agents do not achieve the own final positions. We can started with different

Fig. 3. Agent group: Global action planning
start priorities. If we assume that agent 2 has the greater priority as agent 1 then the conflict manager allows the second agent to realize its motion and after achievement its goal position the first agent obtain the greater priority and achieves its goal. Contrary to such behaviour is action plan if the first agent has the greater start priority. In both cases the agent achieve their goals as shown in fig. The goal solution is realized in sequentially way.

Time, energy and length of motion trajectories for both cases are presented bellow.

<table>
<thead>
<tr>
<th>CASE</th>
<th>PRIORITY</th>
<th>AGENT</th>
<th>LENGTH</th>
<th>TIME</th>
<th>ENERGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optimal traj.</td>
<td>Agent 1</td>
<td>1,8m</td>
<td>2,1sec</td>
<td>119,7J</td>
</tr>
<tr>
<td>1</td>
<td>Optimal traj.</td>
<td>Agent 2</td>
<td>2,1m</td>
<td>1,9sec</td>
<td>206,8J</td>
</tr>
<tr>
<td>2</td>
<td>$\pi_1 \leq \pi_2$</td>
<td>Agent 1</td>
<td>5,1m</td>
<td>6,3sec</td>
<td>327,2J</td>
</tr>
<tr>
<td>2</td>
<td>$\pi_1 \leq \pi_2$</td>
<td>Agent 2</td>
<td>2,2m</td>
<td>2,4sec</td>
<td>210,1J</td>
</tr>
</tbody>
</table>

Fig. 4. Agent group: Sequential action planning

Case 1 presented parameters of motion for optimal trajectories, calculated separately for both agent and Case 2 - agent 1 subordinated shown the parameters of trajectories from start position to the goal position for both agent.

References

Computer-Aided Simulations of Gaussian Processes and Related Asymptotic Properties

Elvira Di Nardo1, Amelia G. Nobile2, Enrica Pirozzi3, and Luigi M. Ricciardi4

1 Dipartimento di Matematica, Contrada Macchia Romana, Potenza, Italia
dinardo@unibas.it,
2 Dipartimento di Matematica e Informatica, Via S. Allende, Salerno, Italia
nobile@unisa.it,
3 Dipartimento di Informatica, Matematica, Elettronica e Trasporti, Via Graziella,
Reggio Calabria, Italia
pirozzi@ns.ing.unirc.it,
4 Dipartimento di Matematica e Applicazioni, Via Cintia, Napoli, Italia
Luigi.Ricciardi@unina.it

Abstract. A parallel algorithm is implemented to simulate sample paths of stationary normal processes possessing a Butterworth-type covariance, in order to investigate asymptotic properties of the first passage time probability densities for time-varying boundaries. After a self-contained outline of the simulation procedure, computational results are included to show that for large times and for large boundaries the first passage time probability density through an asymptotically periodic boundary is exponentially distributed to an excellent degree of approximation.

1 Introduction

It has often been pointed out that first-passage-time (FPT) probability density functions (pdf’s) through generally time-dependent boundaries play an essential role for the stochastic description of the behavior of various biological systems (see, for instance, [5], [6] and references therein). Investigations reported in the literature have essentially proceeded along three main directions:

(i) to search for closed-form solutions under suitable assumptions on the involved stochastic processes and on the boundaries;
(ii) to devise efficient algorithms to reconstruct FPT densities;
(iii) to analyze the asymptotic behavior of the FPT densities as boundaries or time grow larger.

The present paper, that falls within category (iii), is the natural extension of some investigations carried out for the Ornstein-Uhlenbeck (OU) process [4] and successively extended to the class of one-dimensional diffusion processes

* This work has been performed within a joint cooperation agreement between Japan Science and Technology Corporation (JST) and Università di Napoli Federico II, under partial support by National Research Council (CNR) and by Ministry of University and of Scientific and Technological Research (MURST).
admitting steady state densities in the presence of single asymptotically constant boundaries or of single asymptotically periodic boundaries (2 and 3). There, computational as well as analytical results have indicated that the FPT pdf through an asymptotically periodic boundary is approximately exponentially distributed for large times and for large boundaries, i.e.

\[ \tilde{g}(t) \sim \alpha(t) \exp\left\{ -\int_{0}^{t} \alpha(\tau)d\tau \right\}, \]

where \( \alpha(t) \) is a positive periodic function.

However, if one deals with problems involving processes characterized by memory effects, or evolving on a time scale which is comparable with that of measurements or observations, the customarily assumed Markov property does not hold any longer, and hence one is forced to face FPT problems for correlated processes.

As is well known, for such processes no manageable equation holds for the conditioned FPT pdf: to our knowledge, only an excessively cumbersome series expansion is available for time-dependent boundaries when the process is Gaussian, stationary and mean square differentiable 7. Due to the outrageous complexity of the numerical evaluation of the involved partial sums on accounts of the form of its terms, here a completely different approach is discussed in order to gain some insight on the asymptotic behavior of the FPT densities for correlated normal processes. This consists of resorting to a simulation procedure implemented on a Cray T3E parallel supercomputer to generate sample paths of a preassigned normal process to estimate the corresponding FPT densities 1. The results of our computations have shown that for certain periodic boundaries of the form

\[ S(t) = S_0 + A \sin\left(\frac{2\pi t}{Q}\right), \quad S_0, A, Q > 0 \]

not very distant from the initial value of the process, the FTP pdf soon exhibits damped oscillations having the same period of the boundary. Furthermore, starting from quite small times, the estimated FPT densities \( \tilde{g}(t) \) appears to be representable in the form

\[ \tilde{g}(t) \simeq \tilde{Z}(t) e^{-\tilde{\lambda}t}, \]

where \( \tilde{\lambda} \) is a constant that can be estimated by the least squares methods and \( \tilde{Z}(t) \) is a periodic boundary of the period \( T \). The goodness of the exponential approximation increases as the boundary is progressively moved farther apart from the starting point of the process.

In Section 2 a sketch of the simulation procedure to estimate the FPT pdf is provided, while in Section 3 a quantitative approximation of the FPT pdf by an exponential pdf is considered.

2 The Simulation Procedure

Let \( \{X(t), t \geq 0\} \) be a one-dimensional non-singular stationary Gaussian process with mean \( E[X(t)] = 0 \) and with an oscillatory covariance of the Butterworth type:
\[ \gamma(t) := E[X(t + \tau)X(\tau)] = \sqrt{2}e^{-t} \cos\left(t - \frac{\pi}{4}\right) \quad t \geq 0. \]  

Furthermore, let \( S(t) \in C^1[0, +\infty) \) be an arbitrary function such that \( X(0) = 0 < S(0) \). Then, 
\[ T = \inf_{t \geq 0} \{ t : X(t) > S(t) \}, \quad X(0) = 0 \]  
is the FPT random variable and 
\[ g(t) = \frac{\partial}{\partial t} P(T < t) \]  
is the FPT pdf of \( X(t) \) through \( S(t) \) conditional upon \( X(0) = 0 \).

Since from (4), \( \gamma(0) = 1, \dot{\gamma}(0) = 0 \) and \( \ddot{\gamma}(0) < 0 \), the derivative of \( X(t) \) with respect to \( t \), \( \dot{X}(t) \), exists in the mean-square sense and the FPT pdf of \( X(t) \) through \( S(t) \) can be expressed as the following convergent Rice-like series \([7]\):

\[ g(t) = W_1(t) + \sum_{i=1}^{\infty} (-1)^i \int_0^t dt_1 \int_{t_1}^t dt_2 \cdots \int_{t_{i-1}}^t dt_i W_{i+1}(t_1, \ldots, t_i, t), \]

with \( 0 < t_1 < \ldots < t_i, \forall i \in \mathbb{N} \) and where \( W_i(t_1, \ldots, t_i) dt_1, \ldots, dt_i \) denotes the joint probability that \( X(t) \) crosses \( S(t) \) from below in the intervals \((t_1, t_1 + dt_1), \ldots, (t_i, t_i + dt_i)\) given that \( X(0) = 0 \). Because of the computational complexity of the involved integrals, the evaluation of the partial sums in (7) appears is much too cumbersome to be hopefully accomplishable. A complementary approach consists of obtaining a computationally evaluated FPT density by implementing a suitable simulation procedure \([1]\). This procedure consists of constructing sample paths of the stochastic process and of recording their first crossing instants through the assigned boundary. The underlying idea can be applied to any Gaussian process having spectral densities of a rational type with the degree of the denominator greater than that of numerator. In particular, it could be applied to any Gaussian processes with covariance function \([4]\), since its spectral density is

\[ \Gamma(\omega) := \int_{-\infty}^{\infty} \gamma(t) e^{-i\omega t} \, dt = \frac{8}{\omega^4 + 4}. \]

Indeed, the stochastic process \( X(t) \) could be viewed as the output of a linear filter, i.e.:

\[ X(t) = \int_0^\infty h(s) W(t - s) \, ds, \]  

where \( h(t) \) is the impulse response function and \( W(t) \) is the input signal. By Fourier transforming, equation (8) yields \( \Gamma_X(\omega) = |H(\omega)|^2 \Gamma_W(\omega) \), where \( \Gamma_W(\omega) \) and \( \Gamma_X(\omega) \) are respectively the spectral densities of input \( W(t) \) and output \( X(t) \), and where \( H(\omega) \) denotes the Fourier transform of \( h(t) \). If the Gaussian process \( X(t) \) has a preassigned spectral density \( \Gamma_X(\omega) = \Gamma(\omega) \), in (8) \( W(t) \) is identified with a white noise having spectral density \( \Gamma_W(\omega) = 1 \), and \( h(t) \) is selected in such a way that \( |H(\omega)|^2 = \Gamma(\omega) \). If \( \Gamma(\omega) \) is of rational type, it is then possible...
to give a procedure in order to construct $X(k \Delta \vartheta)$ with $k \in \mathbb{N}$ and $\Delta \vartheta > 0$ by using a sequence of two-dimensional standard normal pseudo-random numbers (cf. [1], for the details).

The simulation procedure is structured in such a way that first one specifies the form of the boundary, and then the choice of the involved parameters is made. Input variables are the total number of sample paths to be simulated ($10^6$ in our simulations), the number of sample paths to be simulated in parallel, and the time discretization step $\Delta t$ for the representation of the estimated FPT pdf. As soon as a sample path crosses the preassigned boundary, the instant when such a crossing takes place is recorded. Histograms are then constructed to estimate the FPT pdf $\tilde{g}(t_i)$ at the instant $t_i = i \Delta t$, $i = 1, 2, ..., N$. Let us observe that the sample paths of the process $X(t)$ are constructed at $t = k \Delta \vartheta$, $k = 1, 2, ...$ with $\Delta \vartheta < \Delta t$; however, the simulation step $\Delta \vartheta$ is not an input parameter for this procedure. Indeed, there exists a choice of $\Delta \vartheta$ that is the best possible one for each specified computational time of the simulations and for the assigned degree of goodness of the approximation. After a large number of simulations, the conclusion has been reached that the best choice is $\Delta \vartheta = \Delta t/10$.

By making use of this simulation procedure, extensive computations have been performed to gain some insight on the asymptotic behavior of FPT pdf through varying boundaries of the form (2). The results are discussed in Section 3.
3 The Asymptotic Behavior of Estimated FPT Density

In [6] extensive computations have been indicated, which had been performed in order to gain some insight on the behavior of FPT densities for the OU process characterized by drift $A_1(x) = -x$ and infinitesimal variance $A_2(x) = 2$ through single periodic boundary of the form (2). The FPT pdf $g(t)$ through the boundary $S(t)$ is solution of the second-kind non singular Volterra integral equation

$$g(t) = -2 \Psi[S(t), t, 0, 0] + 2 \int_0^t g(\tau) \Psi[S(t), t|S(\tau), \tau] \, d\tau \quad (S(0) > 0)$$

with

$$\Psi[S(t), t, y, \tau] = \frac{S'(t) + S(t)}{2} f[S(t), t|y, \tau] + \frac{\partial}{\partial x} f(x, t|y, \tau) \bigg|_{x = S(t)},$$

where $f(x, t|y, \tau)$ denotes the transition pdf of the OU process. In [6] it is shown that for periodic boundaries of the form (2), not very distant from the initial value of the process, the numerical evaluation $\tilde{g}(t_i)$ of $g(t_i)$ ($i = 1, 2, \ldots, N$) exhibits damped oscillations having period $Q$ equal to that of the boundary. Furthermore, even for rather small times, $g(t)$ can be represented in the form (3), with $\tilde{\lambda}$ a constant and $\tilde{Z}(t)$ a periodic function of period $Q$. To compute
Fig. 3. The simulated FPT density $\tilde{g}(t)$ for the periodic boundary $S(t) = 2 + 0.1 \sin(2\pi t/3)$ is compared with the exponential density $\tilde{\lambda} e^{-\tilde{\lambda} t}$, with $\tilde{\lambda} = 0.030386$.

$\tilde{Z}(t)$, the following procedure has been devised. After numerically computing $\tilde{g}(t_i)$, the parameter $\tilde{\lambda}$ has been determined via (3) as

$$\tilde{\lambda} = -\frac{1}{nQ} \ln \left[ \frac{\tilde{g}(t_i + nQ)}{\tilde{g}(t_i)} \right]$$  

$(n = 1, 2, \ldots)$.  

(9)

It turns out that $\tilde{\lambda}$ is independent of $n$ and of $t$, even for rather small times. With such a value of $\tilde{\lambda}$, the periodic function $\tilde{Z}(t)$ has been finally computed at times $t = t_i$ $(i = 1, 2, \ldots, N)$ via (3) as

$$\tilde{Z}(t_i) = \tilde{g}(t_i) e^{\tilde{\lambda} t_i}$$  

(10)

These results are clearly suggestive of the opportunity to look for the possibility that similar asymptotic behaviors be exhibited also by Gaussian processes, at least under suitable specific assumptions. However, now a different strategy has appeared to be necessary. Indeed, since for Gaussian processes the FPT pdf $g(t)$ is estimated uniquely by the simulation procedure, presumably $\tilde{g}(t_i) \simeq 0$ for some $i$. Hence, in order to estimate $\tilde{\lambda}$ a different approach must necessarily be pursued.

To this end, let us consider the simulated FPT distribution function $\tilde{G}(t_i)$ for $i = 1, 2, \ldots, N$ evaluated by the Simpson quadrature rule applied to the simulated values $\tilde{g}(t_i)$:
The function $\tilde{Z}(t) = \tilde{g}(t) \exp\{\lambda t\}$ is plotted for the case of Figure 3.

$$\tilde{G}(t_i) = \left[ \frac{\tilde{g}(t_0)}{2} + \sum_{k=1}^{i-1} \tilde{g}(t_k) + \frac{\tilde{g}(t_i)}{2} \right] p$$

with $\tilde{g}(t_0) = 0$ and $p = t_i - t_{i-1}$ a constant for $i = 1, 2, \ldots, N$. We have then used the method of least squares to fit the simulated FPT distribution function (11) with an exponential distribution probability function

$$G(t) = 1 - e^{-\lambda t}.$$ 

To this aim, after setting $y_i = \ln[1 - \tilde{G}(t_i)]$, we have evaluated the minimum with respect to $\lambda$ of the function

$$\sum_{i=1}^{N} \left\{ \ln[1 - G(t_i)] - \ln[1 - \tilde{G}(t_i)] \right\}^2 = \sum_{i=1}^{N} (y_i + \lambda t_i)^2,$$

which is equivalent to solving equation

$$\sum_{i=1}^{N} y_i t_i + \lambda \sum_{i=1}^{N} t_i^2 = 0.$$
Fig. 5. Plot of the simulated FPT density $\tilde{g}(t)$ for the periodic boundary $S(t) = 2 + 0.5 \sin(2\pi t/3)$ and of the exponential density $\hat{\lambda} e^{-\hat{\lambda} t}$, with $\hat{\lambda} = 0.041516$.

with respect to $\lambda$. Hence, the least squares estimate of $\lambda$ can be determined as

$$\tilde{\lambda} = - \frac{\sum_{i=1}^{N} t_i \ln[1 - \tilde{G}(t_i)]}{\sum_{i=1}^{N} t_i^2} \quad \text{(12)}$$

Figures 1-3-5-7 show the FPT densities $\tilde{g}(t)$ obtained via the simulation procedure, as well as the exponential densities $\hat{\lambda} e^{-\hat{\lambda} t}$ in which the parameter $\hat{\lambda}$ has been obtained from (12).

Via equation (10) it is possible to plot the function $\tilde{Z}(t)$ and to analyze its behavior as $t$ increases. The results of numerous simulations have shown that the function $\tilde{Z}(t)$ exhibits a periodic behavior having the same period of the boundary, starting from quite small times (see Figures 2-4-6-8). In conclusion, to a high degree of accuracy one has $\tilde{g}(t) \sim \tilde{Z}(t) e^{-\tilde{\lambda} t}$, i.e. the estimated FPT pdf $\tilde{g}(t)$ is susceptible of the exponential approximation (11) for periodic boundaries, as far as these are not too close to the initial position of the process.

The relevance and the validity of such a numerical results has been confirmed by rigorously proving that the exponential approximation (11) holds for a wide class of Gaussian processes in the presence of any boundary that either possesses a horizontal asymptote or is asymptotically periodic (paper in preparation).
Fig. 6. The function $\tilde{Z}(t) = \tilde{g}(t) \exp\{\tilde{\lambda} t\}$ is plotted for the case of Figure 5.

Here, we limit ourselves to a sketch of the main results. Let us consider separately the following two cases:

(i) the threshold possesses an asymptote;
(ii) the threshold is asymptotically periodic.

Case (i) We consider the FPT problem in the presence of the asymptotically constant boundary

$$S(t) = S_0 + \varrho(t), \quad t \geq 0, \quad (13)$$

with $S_0 \in \mathbb{R}$ and where $\varrho(t) \in C^1[0, +\infty)$ is a bounded function independent of $S_0$ and such that

$$\lim_{t \to +\infty} \varrho(t) = 0 \quad \text{and} \quad \lim_{t \to +\infty} \dot{\varrho}(t) = 0. \quad (14)$$

If

$$\lim_{t \to +\infty} \gamma(t) = 0, \quad \lim_{t \to +\infty} \dot{\gamma}(t) = 0, \quad \lim_{t \to +\infty} \ddot{\gamma}(t) = 0 \quad (15)$$

and

$$\lim_{S_0 \to +\infty} \frac{\varrho \left( \frac{t}{R(S_0)} \right)}{S_0} = 0, \quad (16)$$

then for large values of $S_0$ one has approximately:

$$g(t) \sim R(S_0) \exp\{-R(S_0) t\}, \quad \forall t > 0 \quad (17)$$
Fig. 7. The simulated FPT density $\tilde{g}(t)$ for the periodic boundary $S(t) = 2.5 + 0.1 \sin(2\pi t)$ is compared with the exponential density $\lambda e^{-\lambda t}$ with $\lambda = 0.0096462$.

where

$$R(S_0) = \frac{\sqrt{-\tilde{\gamma}(0)}}{2\pi} \exp \left\{ -\frac{S_0^2}{2} \right\}.$$  \hfill (18)

Case (ii) We consider the FPT problem in the case of an asymptotically periodic boundary \([13]\), where $S_0 \in \mathbb{R}$ and $g(t) \in C^1[0, +\infty)$ is a bounded function independent of $S_0$ and such that

$$\lim_{k \to \infty} g(t + kQ) = V(t), \quad \lim_{k \to \infty} \dot{g}(t + kQ) = \dot{V}(t),$$  \hfill (19)

where $V(t)$ is a periodic function of period $Q > 0$ satisfying

$$\int_{0}^{Q} V(\tau) \, d\tau = 0.$$  \hfill (20)

If the covariance function $\gamma(t)$ satisfies \([13]\) and if

$$\lim_{S_0 \to +\infty} \frac{g \left( \varphi \left( \frac{t}{\alpha} \right) \right)}{S_0 + V \left( \varphi \left( \frac{t}{\alpha} \right) \right)} = 0,$$  \hfill (21)
Fig. 8. The function $\tilde{Z}(t) = \tilde{g}(t) \exp\{\tilde{\lambda} t\}$ is plotted for the case of Figure 7

with $\alpha$ and $\varphi(t)$ such that

$$\alpha = \frac{1}{Q} \int_0^Q R[V(\tau)] \, d\tau, \quad \frac{d}{dt} \varphi\left(\frac{t}{\alpha}\right) = \frac{1}{R[V\left(\varphi\left(\frac{t}{\alpha}\right)\right)]}, \quad (22)$$

then for large values of $S_0$ one has approximately:

$$g(t) \sim R[V(t)] \exp\left\{\int_0^t R[V(\tau)] \, d\tau\right\}, \quad (23)$$

where

$$R[V(t)] = \frac{\sqrt{-\gamma(0)}}{2\pi} \exp\left\{-\frac{[S_0 + V(t)]^2}{2}\right\}$$

and

$$\times \left[ \exp\left(\frac{-[\dot{V}(t)]^2}{2[-\gamma(0)]}\right) - \sqrt{\frac{\pi}{2[-\gamma(0)]}} \dot{V}(t) \text{Erfc}\left(\frac{\dot{V}(t)}{\sqrt{2[-\gamma(0)]}}\right) \right], \quad (24)$$

with $\text{Erfc}(z) = 1 - \text{Erf}(z)$. Note that (23) can also be written as:

$$g(t) \sim \beta(t) e^{-\alpha t}, \quad (25)$$

where $\beta(t)$ is a periodic function of period $Q$ given by

$$\beta(t) = R[V(t)] \exp\left\{\alpha t - \int_0^t R[V(\tau)] \, d\tau\right\}, \quad (26)$$
with $\alpha$ defined in the first of (22). Indeed, since $R[V(t)]$ is a periodic function of period $Q$, due to (20) and (25), one has $\beta(t + nQ) = \beta(t)$.

We have thus obtained in (25) an asymptotic expression of $g(t)$ of the same form as in (3).

The numerical results obtained via the simulation procedure are thus in full agreement with the those mathematically proved in the above cases (i) and (ii).

References


Hybrid Systems’ Properties - Classification and Relation to Computer Science*

Thomas Stauner

Institut für Informatik, TU München, D-80290 München, Germany
stauner@in.tum.de, http://www4.in.tum.de/~stauner/

Abstract. In order to promote a deeper understanding of hybrid, i.e. mixed discrete and continuous, systems, we introduce a set of important properties of such systems and classify them. For the properties of stability and attraction which are central for continuous systems we discuss their relationship to discrete systems usually studied in computer science. An essential result is that the meaning of these properties for discrete systems vitally depends on the used topologies. Based on the classification we discuss the utility of a refinement notion based on trace inclusion. Furthermore, for proofs of stability the role of Lyapunov functions as abstractions is emphasized by identifying conditions under which they define Galois connections.

1 Introduction

The development of hybrid, i.e. mixed discrete and continuous, systems occurring for example in robotics and process engineering is an interdisciplinary task. It mainly involves engineers from computer science and control theory. Developing such systems as well as designing formal methods, such as validation and refinement techniques for them requires a deeper understanding of their essential properties.

Regarding the work on hybrid systems one notices that properties of such systems which are examined in case studies often reflect the background of the research groups working on them. Computer scientists often focus on safety properties, whereas people from control theory often put their emphasis on stability properties. While these distinct focuses are sometimes due to the specific characteristics of the regarded systems, they often also result from a lack of knowledge of the respective other domain. To alleviate this shortcoming we define and classify a set of important properties of control systems within a general framework which should be familiar for computer scientists. For the properties of stability and attraction the paper identifies topologies where stability is a safety property and attraction is a persistence property in computer science terminology.

The properties the paper considers result from the evaluation of hybrid systems case studies and of text books on control theory. The classification of the

* This work was supported with funds of the DFG under reference number Br 887/9 within the priority program Design and design methodology of embedded systems.
properties and the case study evaluation serve as reference for judging the utility of a notion of refinement for hybrid systems which is based on trace inclusion and used in [15]. The result here is that the essential classes of properties are preserved by the intended refinement notion.

Finally, the paper proposes a general proof method for stability which results from adapting a Liapunov like proof method from general systems theory to our framework [10]. Conditions are identified under which a Liapunov function for a stability proof defines a Galois connection. This formalizes the claim that Liapunov functions define abstractions in computer science terminology.

A more detailed presentation of the results in this paper as well as further proof methods and examples are given in [15], and in a revised form, in [16].

**Overview.** Section 2 defines the underlying system model which is the basis for the rest of the paper. Section 3 lists and defines the properties which resulted from the evaluation of case studies and text books. Furthermore, it contains the classification of the properties and examines the utility of trace inclusion as refinement notion for hybrid systems. In Section 4 a proof concept for stability is introduced and a parallel to computer science is drawn. Section 5 discusses the paper’s contribution, compares it with related work, and outlines future work. The appendix introduces some concepts of topology.

## 2 Systems under Consideration

The properties we will present all assume a system structure as the one depicted in Fig. 1. Its basic elements are a controlling device (Controller), the physical environment (Plant) and a feedback loop. Such systems are called feedback control systems.

On a very abstract level we can regard a system as a nondeterministic function mapping a cause to a set of possible effects:

\[
Sys \in C \rightarrow \wp(E)
\]

where \( C \) denotes the set of causes, \( E \) denotes the set of effects and \( \wp(\alpha) \) denotes the power set of \( \alpha \), \( \wp(\alpha) = \{\beta \mid \beta \subseteq \alpha\} \). Pairs \((c, e)\) with \( e \in Sys(c) \) are called behaviors of the system.

---

1 Although control theory usually focuses on deterministic systems, we employ a more general nondeterministic approach which is closer to models in the field of computer science.
Black-box behavior. We use two specializations of this system model. The first one describes the black-box behavior (or I/O behavior) of a system as a function:
\[ Sys_{IO} \in I^R+ \rightarrow \varphi(O^R+) \] (2)
where \( I \) is the input domain and \( O \) is the output domain, i.e. \( C \) has been instantiated to \( I^R+ \) and \( E \) to \( O^R+ \), where \( \alpha^R+ \) denotes the set of functions from the non-negative real numbers \( R_+ \) to \( \alpha \). Elements of \( \alpha^R+ \) are called hybrid streams, hybrid traces or trajectories and functions on streams, like \( Sys_{IO} \), are also called stream processing functions \[4\]. We require \( Sys_{IO} \) to be total in its input, i.e. \( Sys_{IO}(\nu) \neq \emptyset \) for all \( \nu \in I^R+ \). Furthermore, to model realistic systems \( Sys_{IO} \) must be (weakly) time guarded, i.e. its current output may not depend on future inputs:
\[ \nu_1|_{[0,t]} = \nu_2|_{[0,t]} \Rightarrow Sys_{IO}(\nu_1)|_{[0,t]} = Sys_{IO}(\nu_2)|_{[0,t]} \] (3)
where \( \xi|_M \) denotes the restriction of function \( \xi \) to arguments in the set \( M \) and is extended to sets of functions in a pointwise manner. Throughout the paper variable \( t \) is used to denote a point in time, \( t \in R_+ \).

White-box behavior. Some of the properties we will examine require a state-based (or glass-box) system description, or come in two variants, one depending only on the interface behavior and the other depending on the system state. We formalize state-based systems as follows:
\[ Sys_S \in S \rightarrow (I^R+ \rightarrow \varphi(S^R+)) \]
\[ Out \in S \rightarrow O \] (4)
Again \( I \) is the input domain, \( S \) is the state space. Depending on the initial state in \( S \), \( Sys_S \) maps an input trajectory to a set of possible state trajectories. Thus, \( C \) is instantiated to \( S \times I^R+ \) and \( E \) to \( S^R+ \), here. All state trajectories are required to start with the prescribed initial state, \( \forall \sigma \in Sys_S(s, \nu). \sigma(0) = s \). Function \( Out \) maps the current state to the current output in \( O \). Thus, it performs some state-to-output conversion. It is extended in time by pointwise extension, \( Out^\dagger(\sigma)(t) = Out(\sigma(t)) \) for all \( t \in R_+ \). The black box behavior of a state based system with initial state \( s \) is defined by the sequential composition of \( Sys_S(s) \) and the time extension of \( Out \):
\[ Sys_{IO} = Sys_S(s); Out^\dagger \] (5)
where sequential composition of a nondeterministic system \( A \) and a deterministic system \( B \) is defined as \( A; B(\alpha) = \{ \gamma \mid \exists \beta. \beta \in A(\alpha) \land \gamma = B(\beta) \} \). For a set of initial states \( S \subseteq S \) we define \( Sys_S(S) \) by pointwise extension. Like for black-box descriptions, we also require that state-based system descriptions \( Sys_S(s) \) be total in the input and time-guarded for any start state \( s \).

Furthermore, we demand that \( Sys_S \) is time invariant, meaning that the system does not depend on absolute timing. Instead, a system’s state trajectories are completely determined by the initial state and the input trajectory. Formally, we demand that a left shift of the input results in the same left shift of the state trajectory for the shifted start state:
\[ \sigma \in Sys_S(s, \nu) \Rightarrow \sigma^{-u} \in Sys_S(\sigma(u), \nu^{-u}) \] (6)
for any \( u \geq 0 \), where \( \varphi^{-u} \) is the left shift of \( \varphi \) by \( u \), \( \varphi^{-u}(t) = \varphi(u+t) \). Because of time invariance it is sensible to regard a disturbance of a system, i.e. an unforeseen change in its state, as reinitialization of the system. Hence, the behavior of
system $Sys_S$ resulting from disturbing it by setting its state to $s'$ at time $t$ is defined by $Sys_{dis}(s_0, \iota, s', t) = \{ \sigma | \exists \sigma_1 \in Sys_S(s_0, \iota). \exists \sigma_2 \in Sys_S(s', \iota^{-t}). \sigma|_{[0,t]} = \sigma_1|_{[0,t]} \land \sigma^{-t}|_{[0,\infty)} = \sigma_2 \}$. When we consider a system’s reaction to disturbances, it therefore suffices to consider disturbances of the initial state. System behavior after later disturbances can be inferred from its reaction to initial disturbances.

3 Properties

Based on the system model introduced above a number of important properties of control systems are defined and classified in this section. Relative to this classification we then discuss the utility of trace inclusion as refinement notion. The properties have been extracted from the evaluation of nine hybrid systems case studies, which were taken from papers and from student projects performed within various departments of the TU München, and from text books on control theory. For an overview of this background material see [15]. Using case studies as one source was done to be able to estimate the practical relevance of the properties. Unless otherwise mentioned all the following definitions assume that systems are given as nondeterministic functions from a cause to a set of effects. Thus, they apply to I/O based as well as to state-based system descriptions.

3.1 Robustness

An important property of control systems is that they satisfy some required properties in spite of deviations between the model of the system and the real system. Deviations may range from inaccurate parameters, e.g. due to aging effects, to structural mistakes, like an incorrect model of the plant.

Hence, robustness of a system depends on the system itself, the regarded property, and on the regarded deviations. Let $D$ be the set of systems deviating from $Sys$ in the way that is supposed to be considered. For instance, $D$ may be defined relying upon some metric on systems. We demand that $Sys \in D$, i.e. no deviation from the perfect system is also allowed. Furthermore, let $valid_\Psi(Sys)$ be an evaluation function which is true iff system $Sys$ satisfies property $\Psi$. We define robustness as follows.

**Definition 1 (Robustness).** A system $Sys$ with deviations $D$, $Sys \in D$, robustly satisfies property $\Psi$ iff $\forall Sys' \in D. valid_\Psi(Sys')$ holds.

Thus, robustness is parameterized with the notion of validity of the considered object property $\Psi$.

3.2 Optimality

Apart from finding a solution to a given problem, control theory is interested in identifying the optimal solution for the problem w.r.t. some cost function. Possible aims e.g. are minimizing energy consumption or maximizing throughput of a plant. For a given system $Sys$, a set of alternative systems $A$, and a cost function $c$ from the set of systems to a linearly ordered set with order $<$ we define optimality as follows.
Definition 2 (Optimality). System \( Sys \) is optimal w.r.t. alternatives \( A \) and cost function \( c \) iff \( \forall Sys' \in A. c(Sys) \leq c(Sys') \).

Here, \( \leq \) denotes the reflexive closure of \( < \). The set of alternative systems \( A \) may e.g. be defined as the set of all those systems which satisfy a given abstract specification. In practice cost functions are often defined based on the system’s output.

3.3 Stability

The general idea of stability is based on the desire that small disturbances in the causes should only cause small disturbances in the effects. To formalize closeness of one cause (or effect) to another one we use the notion of neighborhood which is induced by the topologies considered for causes and effects (see the Appendix for basic concepts of topology). We write \( N(\alpha) \) for the set of all neighborhoods of \( \alpha \).

Definition 3 ((General) Stability). For a system \( Sys \in \mathcal{C} \rightarrow \varphi(\mathcal{E}) \) between topological spaces \( (\mathcal{C}, \mathcal{O}_C) \) and \( (\mathcal{E}, \mathcal{O}_E) \), the tuple of sets of causes and effects \( (\mathcal{C}, \mathcal{E}) \), with \( \mathcal{C} \subseteq \mathcal{C} \) and \( \mathcal{E} \subseteq \mathcal{E} \), is stable w.r.t. these spaces iff \( \forall \alpha \in N(\mathcal{E}). \exists \beta \in N(\mathcal{C}). \forall b \in \beta. Sys(b) \subseteq \alpha. \)

The definition requires that for any neighborhood of the effects \( E \) there is a neighborhood of the causes \( C \) such that the effects resulting from any of these causes are in the considered neighborhood of \( E \). Note that disturbances and their “size” are a vague concept. The stability definition tries to grasp the notion of “small disturbances of causes” resulting in “small disturbances of effects” by universally quantifying over the neighborhoods \( N(\mathcal{E}) \). For a “small” neighborhood \( \alpha \in N(\mathcal{E}) \), i.e. one that does not contain much more elements than \( E \), stability provides that there is a \( \beta \in N(\mathcal{C}) \), which may also be small, such that disturbed causes in \( \beta \) result in disturbances of the effects which are in \( \alpha \). Hence, stability allows to conclude that there exists a neighborhood of \( C \) in which disturbances of causes must be in order to ensure that the resulting disturbances of the effects remain within a desired neighborhood of \( E \).

Depending on the instantiation of causes and effects in the general definition of stability and depending on employing an I/O-based or a state-based system model, we obtain a number of different notions of stability. In the next paragraph we consider the state-based stability of (sets of) points in more detail. This notion of stability is encountered most frequently in applications. Other variants, including stability of trajectories, are discussed in [15].

Stability of points. Informally, stability of (sets of) points expresses that the effects of a system always are in the neighborhood of some desired points for small disturbances in the system’s causes. In contrast to stability of trajectories which regards traces over time, stability of points views the effects of a system in a pointwise manner, namely for each point in time. For state-based systems, stability of (sets of) points is defined as follows.

Definition 4 (State-based stability of points). For a state-based system \( Sys_S \in S \rightarrow (\mathbb{R}^+ \rightarrow \varphi(S\mathbb{R}^+)) \) a set \( A \subseteq S \) is stable w.r.t. the topology \( \mathcal{O}_S \) on \( S \)
and the inputs $I \subseteq \mathbb{T}^{\mathbb{R}^+}$ if $\forall \alpha \in N(A). \exists \beta \in N(A). \forall \iota \in I. \forall s \in \beta. \text{Sys}_{SS}(s, \iota) \subseteq \alpha^{\mathbb{R}^+}$.

Note that the definition is parameterized with the set of inputs $I$ for which stability must hold. This differs from usual definitions like those in [10, 11], which completely disregard external input for state-based system descriptions. Sets $C$ and $E$ from the general definition are instantiated to $A \subseteq S$, here.

Fig. 2, left, visualizes this notion of stability along an example trajectory: For neighborhood $\alpha$ of $A$ there is neighborhood $\beta$ such that the trajectory starting in $\beta$ never leaves $\alpha$.

While the definition only considers disturbances in a system’s initial state explicitly, we can infer that disturbances in the state at any point in time cause the same (shifted) behavior as a likewise disturbed initial state. This is due to our definition of disturbed systems which in turn is motivated by time invariance of the considered systems (Sect. 2).

Liapunov stability as defined e.g. in [9] is a special case of the definition given here for deterministic systems and the natural metric on the real line. Typically Liapunov stability of a point $x \in \mathbb{R}$ is defined as follows: $\forall \epsilon > 0. \exists \delta > 0. \forall x_0. |x - x_0| < \delta \Rightarrow \forall t > 0. |x - \text{Sys}_{SS}(x_0)(t)| < \epsilon$, where $\text{Sys}_{SS}$ is deterministic and gets no external input. If we take $\{x\}$ as the considered set of causes and effects and if we employ the notion of neighborhood that is induced by taking, as usually, the sets $B_\delta(y) = \{y' \mid |y - y'| < \delta\}$ as base for the topology on $\mathbb{R}$, the equivalence to our definition of stability is straightforward.

The stability definitions in [10, 11] additionally require that the regarded set is invariant, i.e. that it is never left again once it has been entered:

**Definition 5 (Invariance).** A set $A \subseteq S$ is invariant for system $\text{Sys}_{SS}$ and the inputs $I \subseteq \mathbb{T}^{\mathbb{R}^+}$ if $\forall s \in A. \forall \iota \in I. \exists \gamma \in \gamma_{SS}(s, \iota) \subseteq A^{\mathbb{R}^+}$.

In topologies where $A$ is open, invariance of $A$ is equivalent to the stability of $A$ (see [15] for a proof). When associating disturbances with neighborhoods this can informally be explained as follows. In topologies of this kind there is a neighborhood of $A$ (i.e. a disturbance of $A$) which remains within $A$. Hence, the smallest possible disturbance is having no disturbance at all. Or, more informally, there are no “small” disturbances which leave $A$. This suits well to discrete systems where all disturbances can be regarded as equally big, i.e. for discrete systems the only small disturbance also is having no disturbance at all. From a computer science perspective invariance of $A$ is a safety property [5]. Hence, in topologies where $A$ is open stability of $A$ is a safety property.
3.4 Attraction

Often we do not only want that disturbances only have a limited effect (stability), but also that a system returns to some desired trajectory or state after a disturbance. Informally, attraction of a set means that despite of some initial disturbance the set is always reached after a finite or infinite amount of time. Like for stability, a number of variants of attraction result depending on whether we regard attraction of trajectories or of points for I/O-based or for state-based systems (see [15]). Here, we regard state-based attraction of sets of points.

Attraction of points. Attraction of (a set of) points denotes that the effects must reach every neighborhood of the considered points as time progresses. It does not constrain the evolution of the output within this set. For state-based systems attraction of points is defined as follows.

Definition 6 (Attractive set). The set $A \subseteq S$ is attractive w.r.t. the topology $\Omega_S$ on $S$ and the inputs $I \subseteq I^{\mathbb{R}^+}$ iff $\exists \alpha \in N(A). \forall i \in I. \forall s \in \alpha. \forall \beta \in N(A). \forall \sigma \in \mathcal{S}_{SS}(s, i). \exists t. \forall t' \geq t. \sigma(t') \in \beta$.

Set $A$ is globally attractive iff $\forall i \in I. \forall s \in S. \forall \alpha \in N(A). \forall \sigma \in \mathcal{S}_{SS}(s, i). \exists t. \forall t' \geq t. \sigma(t') \in \alpha$. Global attraction means that each system behavior remains inside any neighborhood of $A$ eventually for any starting state. Hence, it expresses a kind of convergence to set $A$. (Normal) attraction only requires that there is a neighborhood of $A$ such that system behaviors starting in it exhibit this convergence to $A$. Again we have parameterized our definition of attraction with a set $I$ of allowed external inputs. Fig. 2, right, visualizes global attraction. Note that the depicted trajectory leaves $\alpha$ before reentering it and not leaving it again thereafter. This does not violate attraction.

In topologies where $A$ is open global attraction is equivalent to the property that $A$ is already reached in finite time ($t \in \mathbb{R}_+$) and not left again, i.e. to $\forall i \in I. \forall s \in S. \forall \sigma \in \mathcal{S}_{SS}(s, i). \exists t. \forall t' \geq t. \sigma(t') \in A$. Hence, the notion of asymptotically approaching $A$ expressed via the neighborhoods in attraction is replaced by truly reaching $A$ eventually in this property (see [15] for a proof). Again, this suits well to discrete systems. The idea of attraction is that we are ensured to get arbitrarily close to the attractive set $A$. For discrete systems all states different from $A$ can be regarded as being far away from it. The only states close to $A$ are those inside it. Thus, for topologies where $A$ is open, i.e. when there exists a neighborhood of $A$ which only contains elements of $A$, attraction is a persistence property [5] when regarded from a computer science point of view. Persistence expresses that “eventually the system always remains in $A$”. Persistence properties contain a safety as well as a liveness part.

Note that in our general setting attraction does not imply stability. The reason is that attraction is interested in system behavior as time goes to infinity, whereas stability considers the whole time axis. However, for most control systems studied in practice attraction implies stability. Therefore, many control theory text books do not introduce the notion of attraction for its own, but only introduce asymptotic stability:
Definition 7 (Asymptotic stability). A is called asymptotically stable iff it is stable and attractive. A is called asymptotically stable in the large iff it is stable and globally attractive.

Asymptotic stability in the sense of Liapunov is a special case of the definition given here for deterministic systems and the natural metric on the real line. For a stable point $x$ asymptotic stability usually requires that the following holds: $\exists \delta > 0. \forall x_0. |x - x_0| < \delta \Rightarrow \lim_{t \to \infty} Sys_S(x_0)(t) = x$, where $Sys_S$ is deterministic and gets no external input. To see that this is a special case of Definition 6 we need to take $\{x\}$ as the considered set of causes and effects, and employ the natural topology on the real line as outlined in Sect. 3.3. As $\lim_{t \to \infty} Sys_S(x_0)(t) = x$ is equivalent to $\forall \epsilon > 0. \exists t'. \forall t \geq t'. |Sys_S(x_0)(t) - x| < \epsilon$, the correspondence to our definition is straightforward.

3.5 Further Properties

Whereas the properties above are rather general and meaningful for most systems, there are a lot of further, more detailed requirements on individual systems. Classes of such properties will be discussed in the following.

Universal properties. A characteristic of many properties is that they constrain all possible behaviors of a system. Formally, such properties can be written as $\forall c \in C. \forall e \in Sys(c). (c, e) \in \Phi$, where $\Phi \subseteq C \times E$. In computer science, properties of this kind are often formalized with linear time temporal logic (LTL) [7].

A very important example for such properties are invariants which demand that a certain time-independent constraint be always satisfied [5]. Typically invariants constrain the value of variables. An example is the property that the temperature in a room always is within prescribed limits [1]. Further examples for properties expressable in LTL are bounded response requirements, and safety and liveness formulas in general [5].

Existential properties. Existential properties can be divided into two relevant groups. First, we have properties which demand that certain behavior exists. Such properties involve existential quantification over causes and effects (see (1) below). Second, there are properties which require that causes exist which are guaranteed to lead to some desired effect. For these properties existential quantification over the causes and universal quantification over the effects is involved (see (2) below).

(1) Existence of behavior. The only property of this kind which was regarded in a few of the evaluated case studies is the existence of periodic behavior when there are no external disturbances. For state-based system descriptions the existence of periodic behavior can formally be written as follows

$$\exists s_0. \exists \sigma \in Sys_S(s_0, 0). \exists t. \exists \tau \in O^{[0, t)} \cdot \sigma = \tau^\infty$$

where $0 \in T^{\mathbb{R}^+}$ denotes a neutral input, i.e. no external disturbances, $O^{[0, t)} = [0, t) \to O$, and $\tau^\infty$ denotes the trajectory resulting from the infinite repetition of $\tau$. Such periodic trajectories in reaction to neutral input are called limit cycles.
In an analogy to computer science the existential path quantifier of CTL can be used to express similar existential properties.

(2) Existence of causes. In the regarded case studies no property of this type was examined. However, the properties of controllability and observability, which are important for state based controller design, are of this kind. In contrast to all the properties regarded before they only refer to the plant, not to the whole control system, and they presuppose a state-based model of the plant. Thus, the term system refers to the plant in the remainder of this paragraph.

Controllability means that there is an input trajectory such that a certain system state can be reached from the initial state within finite time. In an adaption from [14] to nondeterministic systems, we can define controllability from state set $S_1$ to state set $S_2$ as follows:

$$\exists t. \exists \iota. \forall s \in S_1. Sys_S(s, \iota)(t) \subseteq S_2 \tag{8}$$

Note that the input trajectory after time $t$ is irrelevant for controllability, since we are dealing with time guarded systems. A system is called fully controllable iff any system state can be controlled to any other system state.

Observability denotes that for any two distinct states there is an input such that a finite observation of the system output suffices to detect the distinctness of the states. Like in our treatment of controllability we define observability (or distinguishability) of sets of system states first. Two disjoint sets of states $S_1$ and $S_2$ are distinguishable iff $\forall s_1 \in S_1. \forall s_2 \in S_2. \exists t. \exists \iota. (Sys_S(s_1); Out^\dagger(\iota))(t) \cap (Sys_S(s_2); Out^\dagger(\iota))(t) = \emptyset$.

This means that for any two differing start states $s_1$ and $s_2$ from $S_1$ and $S_2$ there exists an input such that the observable output of the system starting in $s_1$ is disjoint from that of the system starting in $s_2$ after some finite time. Note that disjointness is required to ensure that all nondeterministic choices $Sys_S$ can make for the two start states are different. Due to time invariance observability provides that states from $S_1$ and $S_2$ can also be distinguished at any later time instant, not only when they are initial states. Finally, a system is called fully observable if every two distinct states can be distinguished, formally:

$$\forall s_1, s_2 \in S. \exists t. \exists \iota. \exists s_1 \neq s_2 \Rightarrow (Sys_S(s_1); Out^\dagger(\iota))(t) \cap (Sys_S(s_2); Out^\dagger(\iota))(t) = \emptyset \tag{9}$$

If a plant is not controllable or not observable we cannot design controllers which are able to drive it into every desired state, because either the desired state is unreachable from the current state of the plant, or we cannot determine the current state due to lacking observability, or both. In this sense unobservability can be seen as a lack of (appropriate) sensor information, and uncontrollability can be interpreted as a lack of (appropriate) actuators.

In computer science properties of this kind can be expressed with alternating-time temporal logic (ATL) as defined in [3]. In contrast to CTL, ATL allows us to distinguish between the system and its environment in path quantification. Thus, we can express that for all possible moves of the system the environment can select moves (or inputs in our context) which ensure that a certain property holds.
3.6 Classification of the Properties and Its Consequences

The classification of properties we propose is based on the semantic models relative to which the validity of the properties is defined.

The validity of robustness and optimality of a system must be determined relative to a set of systems (Fig. 3 left branch). For robustness, an evaluation function for the regarded object property is necessary additionally. Optimality instead requires a cost function. For the other properties in Sect. 3 no reference systems are needed to determine their validity (Fig. 3 right branch). Determining stability or attraction requires that topologies for the input and output space (or the state space, respectively) of the regarded system are given (Fig. 3 left branch at the second level). For the properties of Sect. 3.5 no topologies are necessary to determine their validity (Fig. 3 right branch at the second level). The properties of Sect. 3.5 are all evaluated in the same domain, namely w.r.t. a given system. As already indicated in the previous section, we partition this subclass further into the set of properties which constrain all behaviors of a system (universal properties) and the set of properties which demand existence of some specific behavior (existential properties). The existential properties can furthermore be divided into those demanding the existence of some behavior and those demanding the existence of causes enforcing certain effects (see Sect. 3.5).

Consequences for refinement. Let us consider which properties are maintained under refinement. The notion of refinement we employ is based on set inclusion, i.e. we say that relation $A$ is a refinement of relation $B$, written as $A \preceq B$, iff $A \subseteq B$. Consequently, for I/O based system descriptions this means that $Sys'_{IO} \preceq Sys_{IO}$ iff $\forall \iota \in I. Sys'_{IO}(\iota) \subseteq Sys_{IO}(\iota)$. For state-based system descriptions we have $Sys'_S \preceq Sys_S$ iff $\forall s \in S, \iota \in I. Sys'_S(s, \iota) \subseteq Sys_S(s, \iota)$. This expresses that the traces of the original system include those of the refined system, while we require that both systems be total in their input and start states (Sect. 2). Note that this notion of refinement is common in computer science, see e.g. [4].

From the definition of universal properties it is obvious that they are preserved under refinement. For stability and attraction it is also easy to see that refinement maintains them. Similarly, controllability and observability are preserved under refinement. Existential properties requiring the existence of effects, like e.g. the existence of periodic behavior or limit cycles (Sect. 3.5) are not maintained by refinement. The situation is more difficult for
the properties robustness and optimality, because we also have to consider sets of
reference systems, the object property and the cost function, respectively, here.
[15] discusses this and also explains the above results in more detail.

As stability and universal properties were in the center of by far most of the
regarded case studies, this stresses that trace inclusion is a useful refinement
notion in most applications.

4 Some Proof Concepts

Because of the importance of stability we present a general proof method for
state-based stability of (sets of) points in the following. As a formal evidence
for the relation of this proof method to abstraction in computer science, we
identify circumstances under which the method defines a Galois connection. A
proof method for attraction, specializations and examples are provided in [15].

4.1 State-Based Stability via Liapunov Functions

Proofs of stability in control theory usually consist of finding a continuous mono-
tonously decreasing function from the system states to some assessment space,
usually the real numbers, which has a unique minimum for the state whose
stability must be shown. According to the work of Liapunov existence of such a
Liapunov function implies the stability of the unique minimum. From an abstract
point of view, the Liapunov function can be seen as an abstraction that maps
the real system to a qualitatively equivalent system [11].

The following theorem may be seen as a generalization of the classical Lia-
punov theory. It is adapted from [10]. In the theorem we write \( \langle v \rangle \) for \( \{ v' \mid v' \subseteq v \} \) for partial order \( \subseteq \) and \( f^{-1}(y) \) for the inverse image of \( y \) under function \( f \). \( f^{-1} \)
is extended to sets by pointwise extension. As the theorem is rather technical
by part, it is explained in the following paragraph.

**Theorem 1.** The set \( A \) is stable w.r.t. system \( \text{ Sys } S \in \mathcal{S} \to (\mathbb{R}^+ \to \wp(\mathbb{R}^+)) \),
topology \( \mathcal{O}_S \), and inputs in \( I \) if there exists a function \( L \in \mathcal{S} \to V \) with:

1. \( V \) is a partially ordered set with partial order \( \subseteq \). \( V^+ \subseteq V \) is the subset
   of elements \( v \in V \) for which there is a neighborhood of \( A \) such that the
   inverse image under \( L \) of all elements \( v' \subseteq v \) is not a proper subset of the
   neighborhood, formally \( V^+ = \{ v \in V \mid \exists \alpha \in N(A). \neg((L^{-1}(\langle v \rangle) \subseteq \alpha)) \} \).
2. \( L \) is monotonously decreasing along the traces of \( \text{ Sys } S \), formally
   \( \forall s \in \mathcal{S}, i \in I. \forall (s, i, \sigma) \in \text{ Sys } S. \forall t, t' \in \mathbb{R}_+. t' \geq t \Rightarrow L(\sigma(t')) \subseteq L(\sigma(t)) \).
3. \( \forall v \in V^+. \exists \alpha \in N(A). \forall x \in \alpha. L(x) \subseteq v \)
4. \( \forall \alpha \in N(A). \exists v \in V^+. \forall x. L(x) \subseteq v \Rightarrow x \in \alpha \)

Based on the application of the theorem’s requirements 4 and 3 the proof is
straightforward and given in [15].

If existing, such a function \( L \) is called a Liapunov function. Informally the
combination of the last two requirements expresses that for any neighborhood
of \( A \) there exists a smaller neighborhood whose \( L \)-image is bounded from above
by some \( v \in V^+ \). The set \( V^+ \) eliminates all those elements from \( V \) which are not helpful in the proof of stability, because they constrain the considered sets of points too much, namely to the inside of all neighborhoods of \( A \). \( V^+ \) is needed to simplify the application of the theorem. Namely, specializations of the theorem usually use sets \( V \) with bottom element \( \perp \) and mappings \( L \) with \( L^{-1}(\perp) = A \). If requirement three quantified over all \( v \in V \), it would not be satisfiable for these specializations in topologies on \( S \) in which \( A \) is a closed set.

In the next section we consider a specialization which is helpful for applications and which makes the parallel to abstraction in computer science explicit.

### 4.2 Liapunov Functions and Galois Connections

The last two requirements in Theorem 1 suggest that there are two total mappings from \( V^+ \) to the neighborhoods of \( A \) (Requirement 3) and back again (Requirement 4). Such mappings are similar to abstraction via Galois connections [6], which is a common technique in computer science. Informally a Galois connection is a pair of monotonous mappings, an abstraction function and a concretization function, from a “concrete” partially ordered set to an “abstract” partially ordered set and back again. The compositions of these mappings are required to loose information in a way consistent with the regarded partial orders. We will make the similarity between Liapunov functions and Galois connections explicit now. This idea leads to a specialization of Theorem 1 which is developed in this section. Note that a detailed understanding of the following presentation requires some advanced concepts from topology. The interested reader who is not familiar with these concepts is referred to [15].

**Assumptions.** Let \((V, \sqsubseteq)\) be a complete partial order which furthermore is densely ordered by the strict version \(\subset\) of \(\sqsubseteq\) and has a least element \(\perp\). Let \(L\) be a mapping from \(S\) to \(V\) such that the given topology on \(S\) is the coarsest topology which makes \(L\) continuous w.r.t. the interval topology induced by \(\sqsubseteq\) on \(V\). \(L\) is required to be onto and the \(L\) image of all elements in \(A\) must be \(\perp\). Furthermore, \(V\) and \(L\) must be chosen such that \(V^+ = V \setminus \{\perp\}\), where \(V^+\) is defined as in Theorem 1. We define function \(abs \in N(A) \rightarrow V^+\) from the neighborhoods of \(A\) to \(V^+\) by \(abs(\alpha) = \sup\{v \in V^+ | \alpha \supseteq L^{-1}(<v>)\}\), where \(<v>\) is the set of all those \(v'\) which are strictly “less than” \(v\), \(<v> = \{v' | v' \subset v\}\).

Function \(conc \in V^+ \rightarrow N(A)\) is given by \(conc(v) = L^{-1}(<v>\).

**Theorem 2.** Functions \(abs\) and \(conc\) are a Galois connection between the spaces \((N(A), \supseteq)\) and \((V^+, \sqsubseteq)\), where \(\sqsubseteq\) is given by \(v \sqsubseteq v' :\Leftrightarrow v' \subset v\).

By the definition of Galois connections [6] this means that monotonicity of \(abs\) and \(conc\), and \(\alpha \supseteq conc(abs(\alpha))\) (extensivity) and \(abs(conc(v)) \supseteq v\) (reductivity)\(^2\)

\(^2\) An example for a densely ordered set with the interval topology are the non-negative real numbers with the usual less-or-equal order. There, the interval topology coincides with the natural topology on the real line.

\(^3\) This can be achieved by choosing \(L\) such that \(L^{-1}(\perp) = A\).

\(^4\) Actually \(abs\) and \(conc\) make up a dual Galois connection [6], because in the usual terminology our \(abs\) is the concretization and \(conc\) the abstraction.
must be proven. The proof is provided in [15], where well-definedness of \( \text{abs} \) and \( \text{conc} \) is also shown.

**Specialization for stability.** The given definition of \( \text{abs} \) and \( \text{conc} \) allows us to derive stability of \( A \) from the monotonicity of \( L \) w.r.t. the system traces of \( \text{Sys}_S \). This claim is proven in [15]. This results in a specialization of Theorem II because an \( L \) function with the properties described in the assumptions above also satisfies the requirements of that theorem. Namely, \( \text{abs} \) and \( \text{conc} \), which are defined depending on \( L \), immediately help to satisfy Requirements 3 and 4 of Theorem II.

Note that monotonicity of \( L \) can also be interpreted as stability of \( \bot \) w.r.t. the \( L \)-image of \( \text{Sys}_S \) in \( V \). This is considered in more detail in [15].

Using the interval topology on \( V \) together with the other requirements we stated for \( V \) and \( L \) implies that \( L^{-1}(\bot) \) is a closed set in \( S \). If we choose \( L \) such that \( L^{-1}(\bot) = A \), this suits well to standard control theory books. There, Liapunov functions usually use \( V = \mathbb{R}_+ \), the considered set \( A \) is a singleton, and therefore closed w.r.t. the natural topology on the Euclidean space, and \( L(x) \) has its unique minimum for \( \{x\} = A \).

Thinking in terms of abstraction and concretization functions can help us to find Liapunov functions. Namely, they lead our intuition to looking for equivalence classes in \( S \) whose elements are then defined to produce the same \( L \)-value. In [16] the above specialization is applied to prove stability of an example system.

## 5 Discussion and Further Work

**Contribution.** Based on an abstract system model which is suitable for hybrid systems and close to models in computer science we have formalized important properties of control systems. The properties have been extracted from the evaluation of nine case studies and a number of text books on control theory. The properties have been classified w.r.t. the semantic models relative to which they are defined. The classification has revealed that a refinement notion based on trace inclusion preserves the properties which were regarded most frequently in the case studies.

For the properties of stability and attraction the vital role topology plays in their definition was made obvious. Topologies were identified under which stability and attraction are equivalent to invariance and persistence, respectively, which are important classes of properties in computer science. Furthermore, a general Liapunov-like proof method for stability was adapted from [10] to our model of hybrid systems. This allowed us to make parallels between the Liapunov theory and abstraction in computer science explicit. Namely, circumstances have been identified under which Liapunov functions define Galois connections.

**Related work.** [11] studies similar properties from a very general system theory point of view. In particular [11] also introduces an extension of the classical Liapunov theory (see e.g. [14]) for their systems. We build on this result by extending it to nondeterministic systems, which play an important role in the early development phases of discrete (computer science) systems. [13] also puts
the Liapunov theory into a more general framework and regards it as one way of defining abstractions, i.e. qualitatively equivalent comparison systems to the system under study. This work, however, remains limited to systems operating on metric spaces as it assumes metric spaces as basis for the abstraction mappings it defines.

[12] mentions that there is a correspondence between invariance in control theory and safety in computer science, and between attraction in control theory and liveness in computer science without going into further detail. With our formal definitions of control systems’ properties we make these correspondences precise by identifying classes of topologies where they become apparent.

[13] regards dynamical systems from an abstract, systems theory point of view, based on predicate transformers. The authors define the concepts of invariance, fullness of invariants (i.e. invariants are not empty) and atomicity of invariants (invariants are singletons) and (finite time) attraction. Furthermore, invariants and attractors are identified as necessary or potential, corresponding to universal or existential path quantification over a system’s computations. This classification into necessary and potential properties is similar to our partitioning of properties in universal and existential properties. However, for existential properties we furthermore distinguish between system inputs and the system output or state, which is in spirit of alternating-time temporal logic (ATL) [3]. This is necessary to classify the control theory properties of controllability and observability.

[2] develops the topological foundations of general dynamical systems starting from iterated relations. Although invariance and attraction, also in the context of the Liapunov theory, are considered there, the theory does not seem to be immediately useful for the application to hybrid systems. It rather supports a deeper understanding of dynamical systems in general.

**Further work.** Based on the results on refinement obtained here, [16] tackles the refinement of hybrid systems. In particular the transition from abstract, hybrid models capturing system requirements to implementation oriented discrete-time models is considered there.

To be immediately useful in practice specializations of the general proof method developed here are necessary. Such specializations should be driven by deficits encountered and experience gained in the practical development of hybrid system.

**Acknowledgment.** We thank Wolfgang Prenninger and Alexander Pretschner for their valuable feedback on draft versions of this paper. Further thanks go to Michel Sintzoff for his hints on related work in particular concerning dynamical systems theory.

**References**

A Some Topology

Definition 8 (Topological space \( \mathbb{S} \)). For a set \( X \) and a family of subsets \( \mathcal{O} \subseteq \mathcal{P}(X) \), \((X, \mathcal{O})\) is a topological space iff (1) \( \emptyset \in \mathcal{O} \) and \( X \in \mathcal{O} \), (2) for \( U_1 \in \mathcal{O} \) and \( U_2 \in \mathcal{O} \), \( U_1 \cap U_2 \in \mathcal{O} \), and (3) for \( \mathcal{A} \subseteq \mathcal{O} \), \( \bigcup \mathcal{A} \in \mathcal{O} \).

\( X \) is also called a space, \( \mathcal{O} \) is called a topology on \( X \), and the elements of \( \mathcal{O} \) are called open sets w.r.t. the topology. A set \( U \subseteq X \) is called closed if its complement \( X \setminus U \) is an open set.

Definition 9 (Neighborhood of a point or set). For \( x \in X \), a set \( U \in \mathcal{O} \) is a neighborhood of \( x \) iff \( x \in U \) \( \mathbb{S} \). We define \( N(x) \) to denote the set of all neighborhoods of \( x \), formally \( N(x) = \{ O \in \mathcal{O} \mid x \in O \} \).

The notion of neighborhood is extended so sets in a pointwise manner: \( U \in \mathcal{O} \) is a neighborhood of \( Y \subseteq X \) iff \( U \) is a neighborhood for every element of \( Y \), formally \( Y \subseteq U \). The set of all neighborhoods of \( Y \) is defined as \( \mathcal{N}(Y) = \{ O \in \mathcal{O} \mid Y \subseteq O \} \).

Note that this notion of neighborhood implies that every open set is a neighborhood of itself: \( Y \in \mathcal{O} \Rightarrow Y \in \mathcal{N}(Y) \).
Assertion-Based Analysis of Hybrid Systems with PVS*

Erika Ábrahá-m-Mumm1, Ulrich Hannemann2, and Martin Steffen1

1 Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Preußerstraße 1–9, D-24105 Kiel, Deutschland
{eab|ms}@informatik.uni-kiel.de
2 Computing Science Department, University of Nijmegen
P.O. Box 9010, NL - 6500 GL Nijmegen, The Netherlands
ulrichh@cs.kun.nl

Abstract. Hybrid systems are a well-established mathematical model for embedded systems. Such systems, which combine discrete and continuous behavior, are increasingly used in safety-critical applications. To guarantee safe functioning, formal verification techniques are crucial. While research in this area concentrates on model checking, deductive techniques attracted less attention.

In this paper we use the general purpose theorem prover PVS for the rigorous formalization and analysis of hybrid systems. To allow for machine-assisted proofs, we implement a deductive assertional proof method within PVS. The sound and complete proof system allows modular proofs in that it comprises a proof rule for the parallel composition. Besides hybrid systems and the proof system, a number of examples are formalized within PVS.

Keywords: hybrid systems, deductive methods, machine-assisted verification.

1 Introduction

Embedded systems interacting with the physical environment pervade everyday’s life and are increasingly used in safety-critical applications, for instance for automotive control, avionics, telematics, chemical process control systems, etc. To guarantee safe functioning, rigorous, i.e., formal arguments are crucial. Their formal analysis is challenging, as well, since such systems are notoriously complex. Capturing the discrete finite-state behavior of the digital device as well as the continuous, infinite-state behavior of the physical environment, hybrid systems [2] provide an appropriate and well-studied formal model. To deal with the complexity of the verification task and to ensure the necessary rigor for fail-safe arguments, computer support is in demand, where two major approaches — enumerative and deductive — of complementary strengths and weaknesses can be distinguished.

Enumerative techniques like model checking promise fully automatic system verification. Based on state-exploration, however, they are limited by the size of the model,

* The work was supported by the Dutch Technology Foundation STW, project EIF 3959, “Formal Design of Industrial Safety-Critical Systems” and further by the German Research Council (DFG) within the special program KONDISK under grant LA 1012/5-1.
especially when dealing with the parallel composition of systems. This phenomenon is known as the notorious state-explosion problem. Furthermore, for hybrid systems as a priori infinite-state models, one has to face the fact that they cannot be dealt with automatically in their full generality. Already the computational properties of timed-automata [3], an important subclass of hybrid systems, are undecidable. Therefore, in the field of model checking, research concentrates on identifying tractable subclasses, for instance linear hybrid systems [2] as the most prominent subclass (cf. for instance [21, 16, 28]). Instead of restricting the class of models, one can also resort to approximative techniques at the expense of information loss (cf. for instance [15, 14]).

In contrast, deductive methods do not support fully automatic verification, but are applicable to the full model of hybrid systems. While there are some theoretical investigations on deductive proof methods for hybrid systems (cf. for instance [22] for an overview), work on computer assistance is scarce. See the concluding section for further discussion of related work in this field.

Classical deductive verification techniques use induction over the system’s computation steps to prove invariance of properties. First introduced for sequential programs, these assertional methods have been extended for more complex models of computation, especially for various forms of parallel and communicating programs (cf. [9] for an extensive monograph on the topic).

In this paper we describe an assertion-based deductive proof method for hybrid systems. To assure rigorous formal reasoning, we employ the interactive theorem prover PVS [25]. PVS is based on higher-order logic, includes extensive libraries of data-structures and theories, offers powerful strategies to assist in routine verification tasks, as well as modularization facilities. We furthermore use PVS to rigorously reason about different examples.

The remainder of the paper is organized as follows. We start in Section 2 briefly surveying the pertinent features of PVS and highlighting the use of the tool for our formalization. In Section 3 we review the definition of hybrid systems, their transition semantics, and their parallel composition. Section 4 describes the proof method for verifying safety properties of hybrid systems, based on assertion networks: After introducing the basic definitions in Section 4.1, we extend them in Section 4.2 to deal with the parallel composition of hybrid systems. After describing in more detail the PVS formalization of hybrid systems including one of the treated examples in Section 5, we conclude in Section 6 with a discussion of related and future work. The library of PVS-theories formalizing the hybrid system model, together with the proof methods and the examples is available via http://www.informatik.uni-kiel.de/~eab.

2 The PVS Theorem Prover

Theorem provers offer mechanized support for logical reasoning in general and for program verification in particular. Unlike verification systems for fully automated reasoning such as model checkers [6], theorem provers provide machine-assistance, i.e., an interactive proof environment. Interactive means that the user is requested to organize the proof, for instance to come up with an induction hypothesis, to split the proof in appropriate lemmas, etc. While doing so, the verification environment takes care of
tedious details like matching and unifying lemmas with the proof goals and assists in the proof organization by keeping track of open proof goals, the collected lemmas and properties. Last but not least it offers a range of automatic decision or semi-decision procedures in special cases. Well-known examples of theorem provers are Isabelle [27], Coq [7], PVS [25, 26] and HOL [11].

To formalize hybrid systems and their theories, we use the theorem prover PVS (Prototype Verification System) developed at SRI International Computer Science Laboratory. PVS is written in Common Lisp and has been used for a wide range of applications; cf. [29] for an extensive bibliography.

PVS’s built-in specification language is a typed higher-order logic. Type declarations, their operations and properties are bundled together into so-called theories which can be organized hierarchically using the IMPORTING-construct. Theories may contain declarations, definitions, axioms, lemmas, and theorems, and can be parameterized with type or value parameters. PVS has a extensive prelude with many predefined types such as lists, sets, natural numbers, integers, reals, relations, functions, etc., and associated lemmas about their properties. Type construction mechanisms are available for building complex types, e.g., lists, function types, records, and recursively defined abstract data types. Being based on a typed logic, PVS automatically performs type-checking to ensure consistency of the specification and the proof-in-progress. Furthermore, the type checking mechanism generates new proof obligations, so-called Type-Correctness Conditions, which are often very useful for an early detection of inconsistencies.

Besides the typed internal logic, the PVS-environment supports the interactive verification by predefined and user-definable proof strategies. It offers facilities for proof maintenance, such as editing and rerunning (partial) proofs, easy reuse of already existing proofs, and the like. PVS notation will be introduced when used in the examples; for a complete description of PVS we refer to the PVS manual [26]. In the sequel, the typewriter-font indicates formalization in the PVS language.

3 Hybrid Systems

3.1 Basic Definitions

Hybrid systems [2] are a well-known formal model for discrete systems acting in a continuous environment. The system’s discrete part is represented as a finite set of locations or modes Loc, connected by discrete transitions or edges. The continuous part is given by a finite set $Var \subseteq Var_g$ of variables ranging over the real numbers $\mathbb{R}$, where $Var_g$ is a countably-infinite variable set. A mapping $\nu : Var \rightarrow \mathbb{R}$ of variables to real values is called a valuation; the set of all valuations is denoted by $V$. A location-valuation pair $\sigma = (l, \nu) \in Loc \times V$ constitutes a state of a hybrid system. Let $\Sigma = Loc \times V$ denote the set of all states. A state set $Ini \subseteq \Sigma$ characterizes the initial states of the system.

As states consist of a discrete and a continuous part, so do the transitions of a hybrid system. A discrete state change is captured by the edges of the graph: leading from one location to another, a transition changes the discrete part of the state; besides that, in going from one location to the next, it may alter non-deterministically the values of the variables. To cater for synchronization between parallel components, the edges come
decorated with a synchronization label from a finite label set \( Lab \). The set of labels contains a specific stutter label \( \tau \) denoting internal moves, not eligible for synchronization. Each location \( l \) is assumed to be able to perform a stutter transition labeled by \( \tau \). Such a transition stands, as usual, for a “do-nothing” step and denotes that other hybrid systems involved in the parallel composition take some discrete transitions. To distinguish between variables the component has under its control in a stutter transition and those it cannot actively influence, the variable set is split into control and non-control variables. The distinction is drawn per location by a function \( Con : Loc \rightarrow 2^{Var} \). Stutter transitions leave the valuations for control variables of the given location unchanged, while putting no restriction on the effect concerning the non-control variables, as they are considered as being influenced solely by the outside.

For the continuous part, the values of the variables may evolve over time, where the corresponding behavior is described, per location, by a set of activities. An activity is a continuous function, describing the variables’ change starting from the moment the location is entered. Since the specific entrance point in time should not influence the behavior relative to that moment, the set of activities for a location is required to be insensitive against shift in time, or time-invariant. Let \( F \) denote the set of all continuous functions in \( \mathbb{R}_{\geq 0} \rightarrow V \). A set \( F \subseteq F \) of activities is called time-invariant, if for all \( f \in F \) and \( t \in \mathbb{R}_{\geq 0} \), also \( f + t \in F \), where \( f + t \) denotes the function which assigns to each \( t' \in \mathbb{R}_{\geq 0} \) the value \( f(t + t') \). An invariant finally is attributed to each location, i.e., a predicate over the set of valuations \( V \), where the system is allowed to enter or stay in a location only as long as the invariant evaluates to true.

Before giving the formal definition of a hybrid system, let us fix some notations. We write \( f|_{A'} : A' \rightarrow B \) for the restriction of a function \( f : A \rightarrow B \) to a sub-domain \( A' \subseteq A \); the same notation is used for the extension of the restriction operator to sets of functions, as well. For \( f \in \mathbb{R}_{\geq 0} \rightarrow V \) and \( x \in Var \), we denote by \( f^x \) the function in \( \mathbb{R}_{\geq 0} \rightarrow \mathbb{R} \) such that \( f^x(t) = f(t)(x) \) for all \( t \in \mathbb{R}_{\geq 0} \). We call a function \( f \in \mathbb{R}_{\geq 0} \rightarrow V \) continuous, if for all \( x \in Var \), \( f^x \) is continuous. The following definition corresponds to the one encoded in PVS; to avoid overly baroque notation, we allowed ourselves to elide type declarations present in PVS within the definitions in the paper, in case the type can unambiguously be inferred from the context. This convention applies to all the following definitions.

**Definition 1 (Hybrid system).** A hybrid system \( H \) is a tuple \((Loc, Var, Con, Ini, Lab, Edg, Act, Inv)\), where \( Loc \) is a finite, non-empty set of locations and \( Var \) a finite, non-empty set of variables. The function \( Con : Loc \rightarrow 2^{Var} \) defines the control variables in each state, the set \( Ini \subseteq \Sigma = Loc \times \mathbb{R}_{\geq 0} \) the initial states. The transitions are given by \( Edg \subseteq Loc \times Lab \times (2^{\mathbb{R}_{\geq 0}} \times (V \rightarrow 2^V)) \times Loc \), where \( Lab \) denotes a finite set of labels containing the stutter label \( \tau \). For all \( l \in Loc \) there is a stutter transition \((l, \tau, (\phi, h), l) \in Edg \) such that \( \phi = V \) and \( h(\nu) = \{ \nu' \mid \nu'|_{Con(l)} = \nu'|_{Con(l)} \} \). The activities are given by \( Act : Loc \rightarrow 2^F \) such that \( Act(l) \) is time-invariant for each location \( l \in Loc \). The function \( Inv : Loc \rightarrow 2^V \) specifies the invariants.

For a discrete transition \((l_1, \alpha, (\phi, h), l_2) \in Edg, \phi \subseteq V\) is called the guard and \( h : V \rightarrow 2^V \) its effect. Depending on various restrictions on the form of the invariants, the guards, the activities etc., a score of variants and simplifications of this model
have been investigated, especially to obtain decidable and automatically checkable sub-
classes of the general definition (cf. for instance [2,3,21,16,28]). As in this paper we
are concerned with formulating a proof method within a deductive framework, we will
stick to the general definition.

Representing the above definition in PVS is straightforward. The hybrid system
tuple is represented by the type hys, a record type, i.e., a product type with named fields
written as [# ... #] (the record value is denoted by (# ... #)). The PVS theory of
the same name hys, partly shown below, contains the type definition of hybrid systems.

PVS

hys: THEORY
BEGIN
... IMPORTING invariant_func

hys: TYPE = [#
  Loc: location_set,
  Vari: variable_set,
  Lab: label_set,
  Cont: control_variable_func[Loc,Vari],
  Ini: state_set[Loc,Vari],
  Edg: edge_set[Loc,Vari,Lab],
  Act: activity_func[Loc,Vari],
  Inv: invariant_func[Loc,Vari] #]
END hys

The component types of the above PVS-definition are implemented and grouped
into separate theories and imported into hys by the IMPORTING-construct. For exam-
ple, the type of an invariant function, which assigns to each location an invariant (i.e., a
valuation set), is implemented as a parameterized theory, since its type depends on the
location and the variable sets:

PVS

invariant_func[(IMPORTING location) Loc :location_set,
  (IMPORTING variable) Vari:variable_set]: THEORY
BEGIN
  IMPORTING valuation[Vari]
invariant_func : TYPE = [(Loc)->valuation_set]
END invariant_func

3.2 Semantics

As mentioned before, a system’s state can change in two ways: either by discrete transi-
tions or by time delay. Hence there are two kinds of transitions between states: an
instantaneous, discrete step, written $\rightarrow^\alpha$, follows an edge $(l_1, \alpha, (\phi, h), l_2)$ of the sys-

\[
\begin{align*}
\phi(\nu_1) = \text{true} \quad & \nu_2 \in h(\nu_1) \quad & \nu_1 \in \text{Inv}(l_1) \quad & \nu_2 \in \text{Inv}(l_2) \quad & (l_1, \alpha, (\phi, h), l_2) \in \text{Edg} \\
(l_1, \nu_1) \rightarrow^\alpha (l_2, \nu_2)
\end{align*}
\]
**Assertion-Based Analysis of Hybrid Systems with PVS**

*Time steps*, written \( \rightarrow^{f,t} \), describe the evolution of the values of the variables in a given location and according to an activity in that location:

\[
\begin{align*}
  f(0) = \nu_1 \\
  f(t) = \nu_2 \\
  \forall 0 \leq t' \leq t. \ f(t') \in \text{Inv}(l) \\
  t \in \mathbb{R}^{\geq 0} \\
  f \in \text{Act}(l)
\end{align*}
\]

\[
(l, \nu_1) \rightarrow^{f,t} (l, \nu_2)
\]

For both relations, control may stay in a location (i.e., time can progress in a location), resp. enter a location through a discrete state change, only if the invariant is not violated.

The *one-step* relation \( \rightarrow \) is defined by \( \rightarrow^{\alpha} \cup \rightarrow^{f,t} \). A *run* of the hybrid system \( H \) is a (finite or infinite) sequence \( \rho = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \), with \( \sigma_0 = (l_0, \nu_0) \in \text{Ini} \) and \( \nu_0 \in \text{Inv}(l_0) \). We denote the set of runs of \( H \) by \( [H] \). A state \( \sigma \in \Sigma \) is *reachable* in \( H \), if there exists a run \( \rho = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \) of \( H \) with \( \sigma_n = \sigma \). We write \( R(H) \) for the set of all reachable states of \( H \).

We use \( \rightarrow^n \), \( \rightarrow^* \), and \( \rightarrow^+ \) to denote respectively the \( n \)-step relation, the reflexive-transitive closure, and the transitive closure of the one-step relation.

The semantics of hybrid systems is defined in PVS as a parameterized theory semantics. We list the core of this theory containing the definition of initial states, and discrete and time step relations, but elide ancillary definitions which should be clear from the context (for the full definitions we have to refer to the web resources):

```pvs
semantics[ IMPORTING hys] H: hys]: THEORY
BEGIN
  ini_step(sigma: state[Loc(H), Vari(H)]) : boolean = 
  Ini(H)(sigma) AND Inv(H)(loc(sigma))(val(sigma))

  disc_step(sigma1, sigma2: state[Loc(H), Vari(H)], 
            e: edge[Loc(H), Vari(H), Lab(H)]) : boolean = 
  trrel(e)((# pre := val(sigma1), 
            post := val(sigma2) #)) AND 
  Inv(H)(sourceloc(e))(val(sigma1)) AND 
  Inv(H)(targetloc(e))(val(sigma2)) AND 
  Edg(H)(e) AND 
  sourceloc(e) = loc(sigma1) AND 
  targetloc(e) = loc(sigma2)

  time_step(sigma1, sigma2: state[Loc(H), Vari(H)], 
            f: activity[Vari(H)], t: nonneg_real) : boolean = 
  f(0) = val(sigma1) AND 
  f(t) = val(sigma2) AND 
  {FORALL (t1: {t1: nonneg_real|t1 <= t}): 
    Inv(H)(loc(sigma1))(f(t1))} AND 
  Inv(H)(loc(sigma1))(val(sigma2)) AND 
  Act(H)(loc(sigma1))(f) AND 
  loc(sigma2) = loc(sigma1)

  ...
END semantics
```

Before giving an example, let us fix some conventions to specify the components of the hybrid system. The standard way to describe the activities is as solutions of differential equations \( \dot{x} = g(x) \) resp. differential inclusions \( \dot{x} \in g(x) \), where \( x = (x_1, \ldots, x_n) \) is a vector of variables from \( Var \) and \( g \) a function from \( V \) to \( V \), resp. from \( V \) to \( 2^V \). We will write subsets of valuations, like the invariants of the locations,
in form of boolean predicates $\varphi : V \rightarrow \text{Bool}$. In such formulas we write short $x$ for the evaluation $\nu(x)$. In a transition relation $(\phi, h)$, the effect $h$ will be written in the form of a simultaneous, non-deterministic assignment $x_1, \ldots, x_n := g_1, \ldots, g_n$, where $x_1, \ldots, x_n \in \text{Var}$, and $g_1, \ldots, g_n$ are set-valued functions from $V$ to $2^\mathbb{R}$. The relation $h$ is then defined as the set of all valuation pairs $(\nu, \nu') \in V^2$ such that $\nu'(x_i) \in g_i(\nu)$ for all $i = 1, \ldots, n$, and $\nu(y) = \nu'(y)$ for all $y \in \text{Var} \backslash \{x_1, \ldots, x_n\}$.

Let’s illustrate the definitions so far on a simple, well-tried example, the thermostat.

Example 2 (Thermostat). The temperature $x$ of a room is controlled by a thermostat, which continuously senses the temperature and turns a heater on and off if the threshold values $x_{\text{min}}$ and $x_{\text{max}}$ are reached, where $x_{\text{min}} < x_{\text{max}}$ and $x_{\text{min}}, x_{\text{max}} \in \mathbb{R} > 0$. When the heater is off, the temperature decreases according to the function $x(t) = x_0 e^{-Kt}$, where $x_0$ is the initial temperature, $t$ the time, and $K \in \mathbb{R} > 0$ a room constant. With the heater turned on, the temperature follows the function $x(t) = (x_0 - h)e^{-Kt} + h$, where $h > x_{\text{min}} + x_{\text{max}}$ is a real-valued constant which depends on the power of the heater. The initial temperature is $x_{\text{max}}$ degrees and the heater is off initially. Two variables $y$ and $z$ serve to record the duration of time spent in the heating and the non-heating mode. The resulting hybrid system is shown in Fig. 1. By convention, trivial components of an edge $(l, \alpha, (\phi, h), l')$, i.e., $\alpha = \tau$, $\phi = \text{true}$, or $f = \text{Id}$ are not shown, and neither are stutter transitions. The same simplification is done for trivial invariants in locations.

The thermostat example is implemented by the theory thermostat:

```plaintext
thermostat: THEORY
BEGIN
... Loc: setof[location] = LAMBDA (l:location): l = l_off OR l = l_on
Vari: setof[variable] = LAMBDA (k:variable): k=x OR k=y OR k=z
Lab: setof[label] = LAMBDA (la:label): la=tau
Inv: invariant_func[Loc,Vari] = LAMBDA (l:(Loc)): LAMBDA (nu:valuation[Vari]): IF l=l_off THEN x_min <= nu(x) ELSE nu(x) <= x_max ENDIF
```

Fig. 1. Thermostat
3.3 The Parallel Composition of Hybrid Systems

Complex systems are often built from smaller components working in parallel. The parallel composition of two hybrid systems \( H_1 \) and \( H_2 \) is given by a standard product construction and written as \( H_1 \times H_2 \). Locations are paired and the set of variables combined. The two partners can take a common discrete step, either by synchronizing on the same label, or in that one of the contributors performs a discrete non-synchronizing transition while its partner stutters. Besides synchronizing on the label in a common discrete step, the conjunction of the actions on the variables is taken, i.e., a common step is possible only if both guards are true and if the outcome on the variables coincides. On variables it does not control, a component cannot block non-synchronizing transitions of its partner, since stutter transitions, available at each location, don’t restrict the behavior of non-controlled variables. On control variables, on the other hand, stuttering is allowed only without changing the variables’ values. Time transitions of the composed systems are time transitions in both systems, i.e., the activities of the composed system, restricted to the local variable sets, are activities of the component systems. Invariants of the composition finally are conjunctions of the component invariants.

**Definition 3 (Parallel composition).** Let \( H_1 \) and \( H_2 \) be two hybrid systems of the forms \( (\text{Loc}_i, \text{Var}_i, \text{Con}_i, \text{Ini}_i, \text{Lab}_i, \text{Edg}_i, \text{Act}_i, \text{Inv}_i) \). The product \( H_1 \times H_2 \) is the hybrid system \( H = (\text{Loc}_1 \times \text{Loc}_2, \text{Var}_1 \cup \text{Var}_2, \text{Con}, \text{Ini}, \text{Lab}_1 \cup \text{Lab}_2, \text{Edg}, \text{Act}, \text{Inv}) \) such that for all \( l_1, l'_1 \in \text{Loc}_1, \ l_2, l'_2 \in \text{Loc}_2, \alpha \in \text{Lab}, \phi \subseteq V, \ h : V \rightarrow 2^V \) and \( f \in F:

1. \( (l_1, l_2), \nu) \in \text{Ini} \iff (l_i, \nu|_{\text{Var}_i}) \in \text{Ini}_i, \text{for } i = 1, 2; \)
2. \( \text{Con}((l_1, l_2)) = \text{Con}_1(l_1) \cup \text{Con}_2(l_2); \)
3. \( (l_1, l_2) \longrightarrow^\alpha_{(\phi, h)} (l'_1, l'_2) \in \text{Edg} \iff \text{there exist } l_i \longrightarrow^\alpha_{(\phi_i, h_i)} l'_i \in \text{Edg}_i, \text{such that } \)
   (a) \( \alpha = \alpha_1 = \alpha_2, \text{or } \alpha_1 = \alpha \notin \text{Lab}_2 \text{ and } \alpha_2 = \tau, \text{or } \alpha_1 = \tau \text{ and } \alpha = \alpha_2 \notin \text{Lab}_1, \)
   (b) \( \nu = \phi_1(\nu|_{\text{Var}_1}) \land \phi_2(\nu|_{\text{Var}_2}), \text{and } \)
   (c) \( \nu' \in h(\nu), \text{iff. } \nu'|_{\text{Var}_1} \in h_1(\nu|_{\text{Var}_1}) \text{ and } \nu'|_{\text{Var}_2} \in h_2(\nu|_{\text{Var}_2}); \)
4. \( f \in \text{Act}((l_1, l_2)); \text{iff. for both } i = 1 \text{ and } i = 2, \text{there exist } f_i \in \text{Act}_i(l_i), \text{such that } \)
   \( \lambda t. f(t)|_{\text{Var}_i} = f_i; \)
5. \( \text{Inv}((l_1, l_2))|_{\text{Var}_i} = \text{Inv}_i(l_i) \text{ for } i = 1, 2. \)

Note that by construction the set of activities \( \text{Act}((l_1, l_2)) \) for a composed location is time invariant, since \( \text{Act}_1(l_1) \) and \( \text{Act}_2(l_2) \) are. It is routine albeit tedious to show that parallel composition is associative and commutative. For a parallel composition \( H_1 \times \ldots \times H_n \) with \( n > 0 \) and \( j \in \{1, \ldots, n\} \), we call the composition system \( H_j \).
without $H_j$ the context of $H_j$. Let $\Sigma_H$ denote the state space of $H$. For the product system $H = H_1 \times H_2$, and a state $\sigma = ((l_1, l_2), \nu)$ of $H$, we write $\sigma \downarrow H_1 = (l_1, \nu|_{\text{Var}_1})$ and $\sigma \downarrow H_2 = (l_2, \nu|_{\text{Var}_2})$ for the projections on the respective components; we will use the same notation for sets of states, and analogously for runs. A basic property of the product system is that all runs of the product projected to one of the component systems are runs of that component system:

**Lemma 4.** Let $H = H_1 \times H_2$ be the parallel composition of two hybrid systems $H_1$ and $H_2$. Then $[H] \downarrow H_i \subseteq [H_i]$ and $R(H) \downarrow H_i \subseteq R(H_i)$, for $i = 1, 2$.

### 4 Proof System

Our approach and formalization to analyze the behavior of hybrid systems is based on Floyd’s *inductive assertion method* [10]. In this classical state-based verification method one associates an assertion, i.e., a predicate over the current values of variables, with each control location of the underlying program. This gives a finite number of verification conditions to check for proving the given correctness criteria of that program. While originally developed in the context of sequential programs, the inductive assertion method serves also as fundamental technique in the analysis of concurrent programs [9]. We extend the inductive assertion method to hybrid systems.

#### 4.1 Inductive Assertional Method

Let ($\text{Loc}$, $\text{Var}$, $\text{Con}$, $\text{Ini}$, $\text{Lab}$, $\text{Edg}$, $\text{Act}$, $\text{Inv}$) be a hybrid system. An assertion on a location $l$ is a boolean predicate over $V$, and an assertion network a predicate over $\Sigma$. For a given assertion network $Q$ of $H$ and a location $l$, let the assertion $Q_l \subseteq V$ be defined by $Q_l = \{\nu \mid (l, \nu) \in Q\}$, i.e., $\nu \in Q_l$ iff. $(l, \nu) \in Q$. Considering subsets of states as predicates on or properties of the states, we say $Q_l$ holds for a valuation $\nu$, in case $\nu \in Q_l$, and correspondingly for states and assertion networks. By the same token, we will speak of an assertion network implying a property etc. In connection with the system’s transition semantics, an assertion network $Q$ is *invariant*, if it holds for all reachable states, it is called *inductive*, if it holds for all initial states and is preserved under the transition relation, i.e., if for all states $\sigma$ and $\sigma'$ from $\Sigma$:

$$\{(l, \nu) \in \text{Ini} \mid \nu \in \text{Inv}(l)\} \subseteq Q$$

(1)

if $\sigma \in Q$ and $\sigma \rightarrow^\alpha \sigma'$, then $\sigma' \in Q$, and

(2)

if $\sigma \in Q$ and $\sigma \rightarrow^{f,t} \sigma'$, then $\sigma' \in Q$.

(3)

Obviously, each inductive network is invariant, while the converse will, in general, not hold. The definitions of inductiveness and invariance and their connection are represented as follows:

Pvs

```plaintext
verification[IMPORTING hys] H:hys]: THEORY
BEGIN
  IMPORTING semantics[H]
```
To verify a property $\varphi$ for all reachable states, one can do so by finding a stronger invariant, i.e., an inductive assertion network $Q$ which implies $\varphi$. This proof principle, known as inductive assertion method, is summarized in the following rule:

$$
\begin{align*}
Q & \rightarrow \varphi \\
Q \text{ inductive for } H & \implies R(H) \rightarrow \varphi
\end{align*}
$$

In PVS, the proof method looks as follows:

```
PVS verification_methods : THEORY
BEGIN
IMPORTING ...
simple_method: LEMMA
    FORALL (H:hys,Q,phi:assertion_network[H]):
        induct[H](Q) AND
        (FORALL (l:location): Loc(H)(l) IMPLIES
        (FORALL (nu:valuation[Vari(H)]): Q(l)(nu) IMPLIES phi(l)(nu)))
        IMPLIES invariant[H](phi)
    ...
END verification_methods
```

It is standard to show that the rule IND is sound and complete. Note that its PVS representation contains the corresponding soundness proof as the proof of the LEMMA simple_method. We have to refer to the technical report [1] for the soundness and completeness proofs.

### 4.2 Inductive Assertional Proofs for Parallel Composition

When analyzing the parallel composition of hybrid systems, it is always possible to apply the inductive assertion method of the previous section directly on the composed system. Neglecting the structure of the composed system, however, often leads to a proliferation of the verification conditions, which reflects the state-explosion problem.

The basic idea for an improvement over the plain product of assertions for the classical programming concepts is a two-level approach, where first local assertion networks are checked for local consistency, and then some global consistency test (the
interference freedom and the cooperation test) relates these local networks, reducing the amount of verification conditions (cf. to [9] for an exhaustive treatment).

In contrast to most applications of assertional methods, which are based on a discrete, interleaving model of concurrency, our method has to capture the continuous system evolution as well as the synchronous nature of hybrid systems’ composition: the context of a single component cannot act independently from that component in the synchronous model, local assertions need not be shown invariant under context actions (i.e., an interference freedom test is redundant). As hybrid systems do not communicate via message passing, no cooperation test is needed, either.

An important technique, commonly called augmentation, which allows to speak about the peer processes of a component, is the extension of the system by fresh, otherwise unused auxiliary variables. As auxiliary variables are added for the sole purpose of verification, their addition must not influence the system’s behavior in any way.

In the following, we will write $H' \geq H$, when $H'$ is an augmentation of $H$ (see [11] for detailed description). For a state set $Q'$ of the augmented system we define the projection $Q' \downarrow H' = \{(l, \nu) \in \Sigma_H | \exists (l, \nu') \in Q', \nu = \nu' | \text{Var}\}$. As the control flow and the activities for variables of $H$ are not influenced by the auxiliary variables, the set of reachable states of $H'$ restricted to the original variable set $\text{Var}$ in the valuation component equals the reachable states of the original system, i.e., $R(H') \downarrow H = R(H)$. Thus, a property whose satisfaction does not depend on the values for the auxiliary variables, holds for all reachable states of $H'$, iff. it holds for all reachable states of $H$.

Let $H_1$ and $H_2$ be hybrid systems, $H = H_1 \times H_2$ with variable set $\text{Var}$ their parallel composition, and $Q_1$ and $Q_2$ assertion networks for $H_1$ and $H_2$, respectively. We define the composition of the local assertion networks as $Q_1 \times Q_2 = \{\sigma \in \Sigma_H | \sigma \downarrow H_1 \in Q_1 \land \sigma \downarrow H_2 \in Q_2\}$. Note that $Q_1 \times Q_2$ is an assertion network of $H$. Now let $\varphi \subseteq \Sigma_H$ a predicate on the set of $H$’s states. Then $\varphi$ is an invariant of $H$ if and only if there exists an auxiliary variable set $\text{Var}_{aux}$, hybrid systems $H'_1$ and $H'_2$, such that $H' = H'_1 \times H'_2$ is an augmentation of $H$ with $\text{Var}_{aux}$, and inductive assertion networks $Q'_1$ and $Q'_2$ of $H'_1$ and $H'_2$, respectively, such that $(Q'_1 \times Q'_2) \downarrow H \subseteq \varphi$. With these conventions, we can formulate the proof rule to deal with the parallel composition of systems.

\[
\begin{align*}
H'_1 \times H'_2 & \geq H_1 \times H_2 \\
Q'_1 \text{ inductive for } H'_1 & \quad Q'_2 \text{ inductive for } H'_2 \\
(Q'_1 \times Q'_2) \downarrow H_1 \times H_2 & \rightarrow \varphi \\
R(H) & \rightarrow \varphi
\end{align*}
\]

**Proposition 5.** The proof rule (COMP) is sound and complete.

For the proof of soundness and completeness we refer to the technical report [11].

5 Verification in PVS

Next we sketch the hierarchical structure of the main theories in the PVS implementation of our proof methods and give an overview of the examples verified within PVS.
5.1 Structure of the Proof System in PVS

In general, the dependencies of the modules mirror the order of definitions and lemmas as presented in the previous sections (or rather the paper follows the structure of the PVS-theories). Fig. 2 gives an overview of the main components.

The basis of the formalization are the theories containing the definition of hybrid systems and their parallel composition. These modules are imported into the definition for the semantics, both for hybrid systems and their parallel composition. The semantics of one instance of a hybrid system is defined as a separate theory parameterized in this instance (cf. the code fragment in Section 3). The theories defining the proof rules for hybrid systems and their parallel composition import the above basic definitions.

5.2 Example

Besides formalizing the proof rules in PVS, we applied the method to a number of examples, e.g., non-linear variations of the water level monitor \([2]\), or a modified clock synchronization of the MPEG4 standard. The PVS formalization of these examples and the verified properties are available on the web-site and in [1]. In the following, we describe in more detail a simple example of a non-linear, composed hybrid system, which computes a linear approximation of a non-linear function.

The approximator is the parallel composition of two hybrid systems, \(H_{\text{input}}\) and \(H_{\text{approx}}\). The first one changes the value of a variable \(x\) according to activities with derivation in \([-\alpha, \alpha]\), where \(\alpha > 0\). The second one reads the value of \(x\) periodically after each time interval \(dT\) and approximates the value of \(x\) linearly, based on the last two received values. The approximated value is represented by the variable \(y\). Variables \(x_0\) and \(y_0\) store the value of \(x\) and \(y\) respectively at the time instance of the last synchronization point. For measuring the time we introduce the clock variable \(z\). Fig. 3 shows the resulting hybrid system.

The property we are interested in is that the approximation error does not exceed a certain margin, i.e., for each state the invariant \(|x - y| \leq \alpha \cdot dT\) holds.

In order to verify this property we define an assertion (network) for \(H_{\text{input}}\), which expresses some boundaries for the value of \(x\):
\[ Q_{\text{input}}(l_0) = x_0 - \alpha z \leq x \leq x_0 + \alpha z. \]

With \( H_{\text{approx}} \), we associate a predicate which expresses that the invariant was valid for the last synchronization point and defines an upper boundary for the actual approximation error:

\[ Q_{\text{approx}}(l_0) = y = y_0 + \frac{x_0 - y_0}{dT} z \land 
| x_0 - y_0 | \leq \alpha \cdot dT \land 
| x - y | \leq \frac{|x_0 - y_0|}{dT} (dT - z) + \alpha z. \]

6 Conclusion

As the main line of research on hybrid systems focuses on model checking techniques for appropriately restricted subclasses, there are less investigations on deductive methods for their verification. In this paper we present an assertional deductive proof method for the verification of hybrid systems. Especially for the verification of composed systems, we give a complete proof rule to reduce the complexity introduced by the parallel composition. To facilitate the tedious verification of those system without restricting the model artificially, we embedded the proof system into the PVS theorem prover. Beside offering the full power of higher-order logic, a further advantage of such a deductive verification environment is that it allows a straightforward rigorous formalization of the mathematical definitions, without the need to resort to any specific logic. Furthermore, PVS comes equipped with a wide range of automated proof-strategies and heuristics.

Related Work Closest in spirit to our work is [5], which embed timed automata into PVS and apply their approach to the steam boiler example. The same example is treated in [33], with the goal of deriving an implementation of a real-time program in a number of refinement steps [19]. The PVS theorem prover is also used in [17] in combination with model checking using HyTech [4] for the reachability analysis for various classes of linear hybrid automata. For the verification of safety properties of hybrid systems, [20] employ hybrid temporal logic HTL, an extension of interval temporal logic. They
give a number of proof-rules which they prove sound. Likewise building upon temporal logic, [24] use the Stanford theorem prover STeP as proof environment. Besides the verification of safety and liveness properties, [31] contains a deeper discussion of the connection of hybrid systems and the field of control theory and presents proof concepts for stability and attraction properties of hybrid systems (cf. also the contribution [32] in this volume). [22] surveys various deductive and algorithmic approaches for the verification of hybrid systems.

Future Work As for future work, we intend to apply our method to larger case studies, especially to extend the control example based on MPEG4 of [8], and further a laser steering system for mass spectroscopy. To improve the specification structure of hybrid systems, the interface information can be extended, for instance separating the variable set into input and output variables like in [23]. Such a cleaner separation is a necessary prerequisite for the development of an assume-guarantee reasoning scheme [30]. Furthermore, we expect that the verification will benefit from an alternative semantics allowing for compositional proofs [13].

References


Algebraic Description of Physical Systems

Hermann von Issendorff

Institut für Netzwerkprogrammierung
D-21745 Hemmoor, Hauptstrasse 40
hviss@issendorff.de

Abstract. The formal methods for the description of discrete systems which have been developed so far are more or less incomplete. In particular, they do not or only partially describe the physical structure of a system. Here, we introduce a formal method, called Akton algebra, which for the first time covers the structural and the operational properties of discrete systems. This makes it possible to completely describe electronic systems including the layout as well as to formally describe biomolecular structures. Moreover, any of the incomplete formal methods may be converted to Akton algebra by assuming a default structure as may Akton algebra be abstracted into these formal methods.

1 Introduction

The notion of system is widely used but does not have a unique definition. A system may be dynamic or static, discrete or continuous, and it may be as complex as a computer or an industrial plant or as simple as a digital gate. Normally, the notion of system is used in an abstract sense covering only certain operational, functional or relational properties.

Here, we make use of the notion of a physical system as the most comprehensive discrete system from which a large variety of other discrete systems can be deduced by abstraction. We define a physical system as a finite directed cyclic network of material components which consume and/or produce a finite number of discrete items, i.e. material or data. Each component and each item occupy unique areas in 4-dimensional space-time.

Based on this definition, we introduce a formal method for the complete description of physical systems, called Akton Algebra and abbreviated to AA. The word „description“ is used in its literal sense meaning „representation by a string of symbols“. The term „complete description“ means to include all operational as well as structural properties of a physical system. AA makes use of a set of specially defined components which are designated by the word „akton“. Essentially, AA can be viewed as a programming language with a largely extended semantics. Actually, it is a many-sorted algebra.

Conventional formal methods for the description of systems only cover partial aspects of a physical system. These formal methods include programming languages and algebras as for instance classical arithmetic. Since AA is complete every feature of the conventional methods can be represented by AA. Features which are not covered by the conventional method can be supplied by default. Thus in principle, every programming language and every algebra can be converted to and from AA. This gives AA the status of a unifying frame. AA may therefore be used as a mediator

© Springer-Verlag Berlin Heidelberg 2001
between different programming languages and algebras. Moreover, since an AA-description is complete, total correctness of a system can just be proved by refining the AA-description down to the level of Boolean algebra.

Among others there are three main fields of application. First, AA may first be applied to discrete technical systems, in particular to electronic circuits. This even includes the layout. Second, AA may be used as a common basis for software as indicated. Thirdly and most exciting, since AA has the capability of describing dynamic structures, i.e. structures which are first generated as a string and then contract into a more-dimensional structure, AA has the potential for describing proteins and DNA sequences.

AA is a novel approach, and we do not know about related work of other authors. To our best knowledge there is at least nothing closely related. However, it may make sense to mention two approaches which can be considered as loosely related. One of them is the Algebra of Flownomials [3], which aims at the formal description of flowcharts, i.e. of plans formally representing control structures. This approach was extended to a Network Algebra, which aims at the formal description of dataflow structures [1]. Both approaches do not refer to physical space although control structures as well as dataflow structures are essentially 3-dimensional. As a consequence, they had to hide the spatial properties in a large variety of special annotated constructs which severely limits their applicability.

This paper is organized as follows. The next section first defines physical space as seen by an observer. This relative space provides the spatial semantics for the elements of AA which are introduced thereafter. The relations between the 3-dimensional space and its 1-dimensional description are studied in more detail in section 3. Section 4 presents a physical interpretation of the elements of AA followed by the representation of 3-dimensional structures which are describable by AA. As will be shown even helical structures can be described precisely. Section 5 expands the scope of AA to the description of processing structures. The fact that combinational circuits as well as feedback circuits can be described shows that AA is also computationally complete. Section 6 deals with the semantical relations between AA on the one side and conventional programming languages as well as classical algebras on the other. Since the latter can be considered as subsystems, they can be converted to and from AA. This is demonstrated in section 7. Section 8 finally mentions a variety of important applications of AA.

2 Akton Algebra

AA describes a physical system as it is recognized by an observer who distinguishes between left and right, above and below, before and behind. The system components are supposed to be directed, i.e. as having the input and the output at opposite sides. This is not a real restriction because components with intermixed inputs and outputs can always be unfolded into several directed components. Furthermore, undirected components can be made directed by arbitrarily defining a direction. The observer always takes a position so that adjacent components which are connected appear to proceed from left to right and adjacent components which are not connected appear above and below each other. Underpasses as appearing in crossings and loops are specially treated by cutting them and marking the ends as being behind.
As a result, the observer sees the physical system stretched out on a plane and oriented from left to right. The original system structure can either be regained by recording the different relative positions of the components, or by assuming physical or chemical attracting forces wherever the system has been cut and stretched.

AA is a many-sorted term algebra. The elements of AA are called akton. An akton represents a physical system component as a black box, i.e. as a box which does not show its internal structure. An akton has an ordered input and an ordered output. Both the input and the output may be empty or contain an arbitrary number of elements. The elements are identified and ordered by their spatial positions. If there are input elements and output elements they are usually related by a processing network. An output element may be related to several input elements and vice versa. The production of an output element from one or more input elements takes an individual amount of time. This means that the production of the output elements of an akton is usually not synchronized.

A term represents a partial physical system as a white box, i.e. as a box which shows its internal structure. It has an ordered input and an ordered output like an akton. A term contains a single akton or any akton structure which can be constructed by means of two binary operators, which will be defined hereafter. A white box can always be abstracted to a black box. This way aktons of arbitrary size and complexity can be generated.

Two spatially adjacent terms \( x \) and \( y \), where the input of \( y \) and the output of \( x \) have the same order and the same content, are written \( x:y \). The colon is a binary operator called \( \text{Next} \) and assumed to define a direction from left to right.

Two spatially adjacent terms \( x \) and \( y \) which are not input/output related are written \( x/y \) if \( x \) lies above \( y \). Both terms \( x \) and \( y \) share the input of the term \( x/y \) so that \( x \) gets its share from the upper end and \( y \) gets its share from the lower end of the common input. If both shares overlap the overlapping subset is accordingly forked. The slash is a binary operator called \( \text{Juxta} \).

The left and right defined by \( \text{Next} \) and the above and below defined by \( \text{Juxta} \) span the plane of observation.

The infix notation of both operators as used here only serves the purpose to increase readability. As a consequence, parentheses have to be introduced in order to designate the terms. The amount of parentheses is reduced by assuming left-first parsing and by giving \( \text{Juxta} \) more binding strength than \( \text{Next} \). For machine processing the less redundant form of Polish notation would be preferable.

There are 6 types of structural aktons each representing a specific basic system structure: The first two types are complementary aktons called \( \text{Entry} (*) \) and \( \text{Exit} (*)' \). An \( \text{Exit} \) exports items from the system to the visible environment and an \( \text{Entry} \) imports items from the visible environment to the system. The \( \text{Entry} \) is only supplied with an output and the \( \text{Exit} \) only with an input. If an \( \text{Entry} \) imports the same items as exported by an \( \text{Exit} \), and if both aktons are planary adjacent they are called matching. Since a matching \( \text{Entry} \) and \( \text{Exit} \) can always uniquely be recognized by their adjacency the matching does not need to be designated by a common name. In contrast, \( \text{Entries} \) and \( \text{Exits} \) which do not match need to be distinguished by different names.

The next two types are complementary aktons called \( \text{Up} (o) \) and \( \text{Down} (o') \). They also serve to import items to or export items from the system. An \( \text{Up} \) has the properties of an \( \text{Entry} \) and a \( \text{Down} \) of an \( \text{Exit} \) except that the environment they are
connected to is considered as hidden behind the plane of observation. Consequently, a matching Down/Up-pair is not planary but spatially adjacent. A matching Down/Up-pair therefore makes it possible to precisely describe a crossover. The description distinguishes between a forward or backward crossing and clearly indicates which line underpasses the other.

A type called Link ($\$\$) serves to separate a pair of immediately connected aktons. A Link has an ordered input and an ordered output which are in a one-to-one relation. Thus, a Link acts like a flat interconnection cable.

A type called Gap (#) only serves to provide space. A Gap has no input and no output and neither imports nor exports anything. It may represent a piece of matter as well as empty space.

The structural aktons are assembled in table 1.

<table>
<thead>
<tr>
<th>Table 1. The structural aktons of AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

In addition to the structural aktons, there is another type of aktons which may be called functional. A functional akton has an ordered input and an ordered output like a Link. However, in contrast to a Link a functional akton usually produces an output which differs from the input. In principle, a single Nand- or Nor-gate suffices to realize every digital function. However, the functional type is not restricted to a single basic element but may include any number of functional aktons of any kind and complexity. Accordingly, different functional aktons need to be distinguished by names.

A particular feature is that functional aktons may be controlled by a three-state-condition. The three-state-condition is a partial input relation which applies conjunctively to all output elements. The condition can be made explicit by enclosing it in brackets and appending it to the akton name. Thus, if $a$ is the name of the functional akton and $\alpha$ is the condition the akton may be designated by $a[\alpha]$. A conditional Link, i.e. $[\alpha]$, is a functional akton by definition. The condition $\alpha$ may have the values true, false and undefined. Normally, $\alpha$ is controlled by a two-valued Boolean variable $p$ which only assumes the values true or false. In order to also get the value undefined we introduce a special function $s$ which changes the value false to undefined. Thus, $p \in \{\text{true, false}\}$ and $s(p) \in \{\text{true, undefined}\}$.

Throughout this paper we will use the letters $a,b,c,d,e,f$ to denominate functional aktons and the letters $u,v,w,x,y,z$ to denominate terms.

3 The Two Mappings from Space to AA

The description of physical systems by AA is actually based on two well-defined successive mappings. The first mapping reduces the three dimensions of the physical system to the two dimensions of a planar representation, and the second mapping reduces the two dimensions of the planar representation to the one dimension of a string of
symbols. Both mappings can be visualized by means of a rubber band model, where the rubber bands designate the input/output relations between the components. However, it should be kept in mind that the rubber bands are virtual, and in particular do not have a physical extension.

The rubber band model assumes that all connections between the aktons can be stretched or bent. The first mapping begins by orientating all substructures from left to right. In particular topological loops and feedback cycles are aligned this way. The system is then stretched in the left/right- and the above/below-direction so that each akton becomes fully visible to the observer. This results in a system which is planar with respect to the aktons but may still contain three-dimensional local structures, i.e. crossovers which originate from the passing of independent lines as well as from topological and feedback loops. The three-dimensional local structures can now be reduced into two-dimensional structures by formally cutting each underpass and replacing it by a Down/Up-pair. The final result of this mapping is an orientated, partially ordered planar structure. The second mapping further stretches the planar representation into a string. This is done by selecting the uppermost Entry or Up at the left side and the lowermost Exit or Down at the right side and pulling them apart. The pulling has the effect of interleaving all independent substructures, so that the upper one of two adjacent independent substructures comes first and the lower one comes second. However, there may be two adjacent substructures which are traversed by a third substructure so that they are not independent anymore. Such a structure cannot be interleaved in a unique manner. The problem is illustrated in figure 1.

The akton structures which are shown there and in other figures of this paper are represented in a symbolic way which deserves some words of explanation. Functional aktons are depicted by named boxes and structural aktons by their specific symbols. A solid line between the aktons does not have any physical meaning but indicates just a dependence relation. A relation between a matching Exit/Entry-pair may be indicated by a broken line, and a relation between a matching Down/Up-pair by a dotted line.

Fig. 1. Non-interleavable akton structure, interleavable akton structure and AA-string

The planar structure at the left side of figure 1 consists of an upper substructure \(x,y\) and a lower one \(u,v\) with a traversing substructure \(w\) in between. There, \(w\) is as well lower to \(x\) as upper to \(v\), and can thus not be interleaved. The solution is to cut the connection of the substructure \(w\) on either side and replace the cut by an Exit/Entry-pair. In the planar akton structure at the right side of figure 1 the connection between \(w\) and \(y\) has been cut. This structure is uniquely described by the AA-string below.
4 Physical Structures

Up to now the elements of AA are only provided with a rather poor semantics covering only some formal functional and spatial relations. However, in order to establish a clear relation between AA and a physical system the elements of AA need to be provided with some more properties. Depending on the kind of the physical system described by AA, there are two major interpretations.

1. In the biplanar setting of electronic circuits Next is interpreted as a flexible joint which connects two substructures so that they always touch each other. The joint does not occupy any area as long as both substructures are aligned. The area usually increases if the substructures are bent against each other. In order to minimize the area of the joint the substructures may be shifted. Juxta is interpreted as a vertical alignment. A matching Up/Down-pair is interpreted as two sets of vias which are interconnected on a back-plane. Matching Exit/Entry-pairs are interpreted as a connection with zero extension. Nonmatching Exit/Entry-pairs are interpreted as plugs. Links of appropriate size are inserted whereever dependent aktons need to be connected which are not physically adjacent.

2. In a general 3-dimensional setting Next is also interpreted as a flexible joint. Juxta is interpreted as a bond which holds both substructures in position as long as there are no stronger external forces pulling them apart. Matching Exit/Entry-pairs and Down/Up-pairs are interpreted as real physical objects. The aktons of the Exit/Entry-pair and of the Down/Up-pair are viewed as two objects which attract each other by means of physical or chemical forces. Nonmatching elements of both pairs are interpreted as objects which are ready to couple with another system. With this, AA describes a linear physical system which has the capability to transform into a spatial physical system in a well-defined manner. This feature closely resembles a fundamental feature of life, i.e. the formation of proteins from chains of amino acids.

Fig. 2. Physical structure, akton structure and AA-expression of complementary crossovers

There are two basic structures which can be found in every spatial system, the crossover and the topological loop. They are discussed in more detail in this section. A crossover consists of two independent strands which cross each other. This gives the crossover a 3-dimensional structure, where one dimension serves to underpass one of the strands. Depending on which strand underpasses the other one there are two complementary structures. In AA they are clearly distinguished by the Down/Up-pair describing the underpass. Thus, AA has the capability to express the chirality of structures. Figure 2 depicts the two complementary crossovers, their akton structures and the akton expressions. The upper structure exhibits a left-handed chirality and the
lower one a right-handed chirality. The black points indicate the Up and Down and the dotted line the relation between them. In a biplanar setting Up and Down are vias to a back-plane, where they are connected. In a general 3-dimensional setting Up and Down are either only symbolic or representing objects which attract each other.

Fig. 3. Forward and backward types of a left-handed crossover

Actually, this first example only shows one type of a crossover, which may be called a forward-type. In addition, there is a second type, which accordingly may be called a backward-type. A third type which exhibits an orthogonal crossover is by definition assumed to also represent a forward-type. Figure 3 depicts the three types of a left-handed crossover.

Fig. 4. Two essential structures: Fork and shuffle (S)

There is an important topological difference between the forward- and the backward-type of a crossover: The underpassing connection of the forward-type keeps the line ordering during the passage while the line ordering is reversed in the backward-type.

The forward-type crossover is an essential part of two basic structures which in particular are extensively used in all data processing systems. These basic structures are commonly called fork and shuffle. A fork splits a connection into two connections having the same line ordering, and a shuffle merges two ordered connections of the same cardinality into a single one with ordered pairs of lines. Figure 4 depicts the physical representation of a fork structure and a shuffle structure together with their
AA descriptions. Of course there is a complementary structure for each of them. Usually, the fork structure does not explicitly show up in AA because it is normally assumed to be an integral part of the Juxta-operator. However, it can always be made explicit if necessary. On the other hand, a shuffle represents an explicit akton, which accordingly needs to be named. The shuffle akton is designated by the symbols $S^+$, where the '+' indicates that it may have as many pairs of lines as required. In data processing, a shuffle is usually followed by an akton consisting of a set of Or- or And-gates. Each of these gates reduce an adjacent pair of lines to a single line. Since these aktons contain as many gates as required they are accordingly designated by $O^+$ and $A^+$. The shuffle akton and a multiple gate akton may be composed into an akton designated by $SO^+$ or $SA^+$.

Fig. 5. Physical structure, AA-structure and -description of complementary open loops

A topological loop is purely structural as opposed to a feedback loop which always contains some functionality. The feedback loop will be treated in the next section. A topological loop may either be open or closed. The structure of an open topological loop is closely related to the crossover structure. While the crossover deals with the crossing of two independent strands, the loop deals with a strand which crosses itself. By necessity, this can only be achieved by a backward crossing. The structure of an open topological loop therefore includes a backward crossover. Accordingly, an open topological loop has a chirality like the crossovers. Both, a left-handed and a right-handed open loop are depicted in figure 5 together with the corresponding akton structure and the AA description. As before, the underpassing connection is indicated by a dotted line. Its special form does not have any relevance. It has only been chosen to clearly show the underpassing.

The Exit/Entry-pair and the Down/Up-pair can both be used to describe closed loops. In the first case the closed loop would lie in the plane of observation, and orthogonal to it in the second case. Both cases are covered by AA as shown in figure 6. A closed loop may be directed or not. If it is not directed a direction can be introduced by default. In order to become describable a closed loop has to be cut somewhere. A closed loop lying in the plane is described by means of the Exit/Entry-pair, while an orthogonal structure is described by an Down/Up-pair. However, while the orthogonal representation is unique, the planar representation is not. This deficiency, however, can be removed by placing a Gap above or below the akton which represents the loop. Because the Gap occupies some space the distance between the Exit and the Entry is then shorter at the other side.
We will now demonstrate the description of open and closed 3-dimensional structures by means of two examples. The first example is a helix, i.e. one of the most important structures which are used by nature in DNA and proteins. The helix shown in figure 7 is a rather primitive one. It consists of two loops of three aktons each. Nevertheless it shows the principle. In order to describe the helix the two loops are cut as indicated by a small black plane. The helix can then be defolded into a flat structure as shown by the akton structure in the middle. The akton structure can be described uniquely by the AA-expression at the right side. It should be pointed out that the denomination of the 6 aktons is not necessary as long as only the structure is described. Structurally, the aktons are uniquely discriminated by their relative spatial position. Here, the denominators have only been introduced to elucidate the relations between the physical structure, the akton structure and the akton expression.

In nature, the structural and the functional properties of a protein are described by a chain of amino acids. There are 20 different types of amino acids by which the chain is built up. However, up to now it is widely unknown how the structural and the functional properties of the protein are coded across the amino acids. Studies which try to unveil the folding process from the chain of amino acids to the protein are still in a rather early state [6].

In comparison, in AA, the structural and the functional properties of a data processing system also are described by a chain of symbols. The set of 9 symbols used by AA, i.e. 2 binary operators, 6 structural aktons and at least one functional akton, is smaller than the set of amino acids. This gives rise to the conjecture that there may be a relation between the language of amino acids and AA.

The closed structure shown in figure 8 is a tetrahedron. The edges are designated by $e$. In order to get a planar representation it needs to be cut first by a Down/Up-pair. The cut is indicated by a black plane. The emerging planar structure needs to be further cut by an Exit/Entry-pair. With these two cuts the tetrahedron can uniquely be

**Fig. 6.** Two views of a closed topological loop

**Fig. 7.** AA-representation of a left-handed two-loop helix
described by AA. This means that the tetrahedron could be completely reconstructed
from the AA-expression. There is no need to index the edges because each of them
has a unique position in space. The AA-expression even states that the tetrahedron has
been defolded from the backside.

![Fig. 8. AA-representation of a tetrahedron](image)

5 Data Processing Systems

In the previous section we demonstrated how a variety of physical structures can be
described by AA. In this section we will show how AA can be applied to two basic
data processing structures, i.e. the combinational network and a feedback network.
These two networks have been chosen because they suffice to build up every data
processing system. Since AA also describes every composition of these networks the
conclusion is that AA completely describes every data processing system at every
level of abstraction. The composition rules will be discussed in the next section.

![Fig. 9. AA-representation of a half-adder](image)

It should be noted that up to now there is no formalism by which combinational or
feedback networks can be described analytically. Combinational networks need to be
split up in a set of Boolean functions, i.e. one function for each exit. Feedback net-
works even cannot be described at all. They therefore need to be encapsulated and
treated as a black box.

We take a half-adder as an example for a combinational network. The principle
akton structure and its akton expression is shown in figure 9. The Entries \(*_a/\) provide the input, Exit \(*'_c\) provides the carry and Exit \(*'_r\), the result. The letters \(A\), \(O\),
\(N\) designate an And-gate, an Or-gate and a Not-gate. Please remember that the solid
lines only show a dependence relation as defined in section 3. The structure and the
function of the half-adder is precisely described by the akton expression at the right
side.

In comparison, the Boolean functions \(a \land b = c\) and \((a \lor b) \land \neg(a \land b) = r\) only define
the functionality. Of course, there are many other structures aside from the one shown
in figure 9. Other structures could for instance be achieved by turning it upside down or by using a separate And-gate for the carry.

\[
\text{Fig. 10. A (left-handed) feedback-loop controlled by akton } LC
\]

A feedback loop is an active open topological loop, where the loop input and output are commonly controlled. This gives the loop a structure as shown in figure 10. There the akton \( LC \) does the loop control. Term \( x \) designates the activities inside the loop and term \( y \) those at the loop exit. Both are controlled by \( LC \). This feedback structure can virtually be found in all loop constructs of data processing no matter if they are software or hardware.

A primitive feedback just contains a single And-gate or a Nand-gate. The And-gate provides a positive feedback and the Nand-gate a negative one. This means that if activated the And-gate turns the feedback into a stable state and the Nand-gate into an oscillation. This behaviour endures as long as the systems stay activated. While the Nand-network represents a clock, the And-network as such is not useful. However, a combination of them leads to a bistable network, i.e. to a storage device called flipflop. The most primitive flipflop, i.e. an RS-flipflop, is depicted in figure 10. The structure of the RS-flipflop is the same which is generally used for representation.

\[
\text{Fig. 11. AA-structure and -expression of an RS-flipflop}
\]

However, the structure of the RS-flipflop could be modified in many ways. The \( \text{Exit/Entry} \)-pair could for instance be eliminated by shifting the lower part of the structure to the right. The structure shown here corresponds to the standard representation.

6 Semantical Distance to Conventional Systems

Any encapsulation, abstraction or restriction reduces the general system to a subsystem. There are numerous formal methods and models for the representation of systems. An important class of systems is that of the general purpose programming languages.

The derivation of these systems from a general system requires at least three formal steps. The first step abstracts from the spatial properties of the components, i.e. their position and physical connections. This step reduces the general system to an
operational network of functional components, where the data are expressed by variables and the components are expressed by statements.

The second step encapsulates the dynamic network structure in special control constructs like the assignment for storage, the while statement for repetitions and the if statement for alternatives.

The third step restricts the operational network. Restriction to a lattice structure results in the system of concurrent programming languages, restriction to a linear structure in the system of classical sequential programming languages.

Finally, total abstraction from any operational network reduces the system to a set of partially related functional components, the domain of functional programming languages. A prominent member of these languages is ASM [2], which was created as a tool for the design of data processing systems.

Further excluding the control constructs of assignment, feedback and alternatives from the last system results in the well-known system of classical algebras, e.g. Boolean algebra and arithmetic.

In contrast, there are only a few formal methods dealing with the algebraic description of hardware structures. Two of them are CADIC [4] and Ruby [5]. Their systems are derived from the general system by abstracting from any control, i.e. from feedback cycles and alternatives, and by mapping the 3-dimensional network onto two dimensions. This creates the problem of incompletely defined crossings which can only be solved by abstraction, i.e. by concealing the crossings in special constructs. The system described by Ruby is directed, acyclic and planar. Ruby is thus suited to describe the planar structure of combinational circuits. CADIC is restricted to a layout system which is obtained by further abstracting from the functions of the components.

The systems mentioned above are not the only type of systems which are covered by the general system. For example scheduling systems, workflow systems and the like can be treated as well.

7 Conversions to and from Programming Languages and Algebras

The considerations of the previous section reveal that AA is a kind of a superlanguage which covers the properties of a broad variety of productive or constructive formal methods. It should therefore be possible to formally reduce AA to the properties of the restricted methods or to raise these methods to AA by adding the missing properties by default. The main question is, however, whether this requires a complex procedure or not. Surprisingly, the procedure turns out to be simple and regular. It only requires a set of conversion rules for either direction, which often can even be identical.

Here, we present the conversion rules for two methods, i.e. for Boolean algebra and for a simple programming language. Table 2 represents the conversion rules for Boolean algebra.

The table contains several term replacement rules, each consisting of an upper and a lower term. The double-pointing arrow means that either term can be replaced by the other. If the replacement is restricted, the restriction is put in square brackets next to the arrow. The Greek letter ρ designates a general conversion function which maps the upper term, here an AA-term, to the lower term, here a Boolean term. The letters ρ
and \( q \) designate data. Since in general an AA-term includes some physical information, \( \rho \) can be said to abstract from the physical properties.

**Tab. 2.** Conversion rules to and from Boolean algebra

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>( \frac{\rho((x/y:A))}{(\rho(x) \land \rho(y))} )</td>
<td>b.</td>
</tr>
<tr>
<td>d.</td>
<td>( \frac{\rho(*_p)}{p} )</td>
<td>[Out(_p) = {p}]</td>
</tr>
</tbody>
</table>

The reverse conversion, i.e. replacing the lower term by the upper term can accordingly be done by a conversion function \( \rho^{-1} \). Here, the reverse function \( \rho^{-1} \) can be said to add the missing physical information by default.

As an example we convert the Boolean expressions defining a half-adder. The input variables of the half-adder are designated by \( a, b \), the result by variable \( r \) and the carry by variable \( c \). The Boolean equations are

\[
(a \lor b) \land \neg (a \land b) = r \quad \text{and} \quad a \land b = c
\]

The conversion begins with the replacements of the Boolean variables in the leftmost term \( (a \lor b) \) by means of rule d, which results in \( (\rho(*_a) \lor \rho(*_b)) \). This term is then replaced by rule b, resulting in \( (*_a / *_b:O) \). The conversion is continued this way until the conversion is completed. The corresponding AA-expressions are

\[
(*_a / *_b:O)/(A:N:*_c) \quad \text{and} \quad *_a / *_b:A:*_c
\]

The two akton expressions above are actually describing two systems, the first producing the carry \( c \) and the latter the result \( r \). They can be combined to a single expression describing a single system, e.g. to

\[
\]

This can be achieved by a set of transformation rules which are not shown here.

**Tab. 3.** De Morgan-rules of Boolean algebra written in AA.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>( (A:N) )</td>
<td>b.</td>
</tr>
<tr>
<td>( (N:N:O) )</td>
<td>( (N:N:A) )</td>
<td></td>
</tr>
</tbody>
</table>

It may be interesting to see the De Morgan-rules written in AA, as presented in table 3. It should be noted that in AA the upper and the lower term of the two rules cannot be equated because they are representing systems with different structures, different functions and thus a different behaviour, which are only functionally equivalent.

Next, we are presenting the conversion rules for a Pascal-like programming language, as represented in table 4. The conversion rules are built up in the same way as before. However, the rules representing alternatives and feedback control are
considerably more complex. The *if*-statement the *if-then-else*-statement are covered by the rules g and h, and the *while*-statement by rule i. Please note that in AA the *if*-statements are controlled by the Boolean value $p_i$, whereas the *while*-statement is controlled by the special function $s(p_i)$, which has been introduced at the end of section 2. This reflects the fact that the two conditional branches of both *if*-statements are commonly evaluated and then merged by $SO^+$, in contrast to the *while*-statement, where the loop-exit is shut off as long as the loop is running.

Table 4. Conversion rules to and from a Pascal-like programming language.

<table>
<thead>
<tr>
<th>a.</th>
<th>$\rho(akt_j$/s)</th>
<th>b.</th>
<th>$\rho(*j$/s)</th>
<th>c.</th>
<th>$\rho(*/$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d.</td>
<td>$\rho((x:y)/$)</td>
<td>e.</td>
<td>$\rho((x)/$)</td>
<td>f.</td>
<td>$\rho(*/$)</td>
</tr>
<tr>
<td>g.</td>
<td>$\rho($(($</td>
<td>p_i</td>
<td>:x)/$</td>
<td>$</td>
<td>\neg p_i</td>
</tr>
<tr>
<td>i.</td>
<td>$\rho($(($</td>
<td>p_i</td>
<td>:x)/$</td>
<td>$\neg p_i</td>
<td>:SO^+$/)$)</td>
</tr>
</tbody>
</table>

The conversion between the programming language and AA is slightly complicated. The problem is that the programming language does not explicitly show the data relations between the statements. In order to solve this problem, every AA-term in the conversion rules except for the complete term in rule f is supplied with a Link by default. The Links which are not needed, i.e. empty Links, can then be eliminated in a subsequent run. Vice versa, if using the same conversion rules, all terms of the AA-expression except the one regarding rule f have to be provided with a Link before the conversion can be done.

An application of this conversion table will be demonstrated by a trivial program consisting of three statements only. The conversion is depicted in figure 12. The conversion proceeds as indicated by the arrows. Each term is converted by the rule shown in front. The conversion is complete after the application of rule f. An optimizing run would reduce the AA-expression to

$*begin:*_1$/s$*akt_2$/s$*end.$

The remaining Links serve for passing an activation signal from $*begin$ to $*end$. 

begin;
read p;
q:=p/2;
write q;
end.

Fig. 12. Conversion of the program left to the AA-expression down right
8 Final Remarks and Future Work

This paper presents a first and rather short introduction to AA. Many of the topics could have been treated in more detail. In particular, because of lack of space, two important topics had to be omitted completely. These are
- Composition and Function Preserving Transformation and
- AA as a layout-language for electronic circuits
In addition, there are a large variety of different applications of AA which still have to be elaborated. Some of them should at least be mentioned. AA could be applied to
- automated proving of total formal correctness
- automated casting of software into hardware
- software recycling by extracting valuable components from old software
and in particular
- exploring the code of life, i.e. the structural and functional properties of the set of the essential amino acids.

References

Abstract. Apart from purely theoretical approaches (mainly in microeconomics), there seem to be not a lot of studies which employ computational environment to examine/explain sociological/ethical process of building informal institution in a society, or statement like “people should behave moralistically”. In this study I introduce an “agent based model” and then present an idea to examine/explain the sociological/ethical processes and statements using it. I make a trial to set a situation in which the agents can be defined to be “moralistic”, using the notion of games and information structure. Some meaningful results on the formalisation are shown.

1 Introduction

The purpose of this study is to construct an agent-based simulation system derived from game theory. To do that, formulation/formalisation of required/supposed situation is in order. In this paper, I demonstrate only the introductory part of the study, namely, the formal theory of agents who can solve a ‘game’ based on the ‘information structures’ they own.

There seems to be quite a few researches which are based on some model of “complicated agent” which is, for example, a “description of reasoning” of human being. I use “complicated” to mean some property of agent which include transition mechanism of state-transition function. (So, it would be something different from automata in usual sense.) The reference [1], for example, studies the simulation of one significant condition whose notion is based on “one-shot game” and its iteration. I would say the situation described in the experiment of “iterated prisoner’s dilemma” [1] or [1] may be too much simplified to be claimed that they are the description of “virtual society”. For the reason, the context I introduce in this paper could have the significance which is similar to the claim in [1], [1] and so on.

I recognise this type of issue is the matter for the area of sociology, economics, or even ethics. But I’d mention the fact that, if we could describe some aspect of complicated property of our object (including members of the society) in some proper way, a simulation based on the the model which employs the description should have some significant, not only from the view of ‘computational social
science, but also from the viewpoint of the systems theory. As we can see the recent paradigm called “complex systems” ([5]. As is well-known, this book is famous about its manifest of “complex systems” as a new systems-theoretical paradigm), a result of simulation is not necessarily required to have the direct application of technology or to be one reproduction of some natural phenomenon. Rather, according to the opinions in the paradigm “complex systems”, one result on a simulation system should be considered to be a pattern which could have been possible in the given situation.

This study does not contain some kind of critical view against existing sociological/economical studies which is based on computer simulation.

The main stream of this study proceeds as:

– Formal theory of “agent”, based on “game theory and theory of information structure”,
– Several ideas for “simulation system” and “the system which results proper rules” for agents not to die out,
– Result of the simulation (Unfortunately, so far, this part is under construction).

I should mention briefly here to the reason for letting an agent have an information structure. In the simulation system, games are repeated and the results should also be one feedback of the agent. To implement this kind of condition, the notion of information structure can be considered to be relatively useful and reasonable.

2 The Agent ; as a Player in Games

Firstly we review and re-formulate a game and agents on it. As will be referred later, the word “player” is used instead, following the vocabulary of game theory.

I generalise Aumann’s formulation ([4], in order to make it easy for the notions to be introduced in some computational environment.

2.1 Game and Its Model

Let a pair of characters {<, >} be given.

**Definition 1 (Game Tree, Sub-game tree and its depth).** A game tree is a structure defined recursively as follows:

1) $<>$ is a game tree.

2) $x_1, \cdots, x_n$ are game trees, then so is $<x_1 \cdots x_n>$.

3) Nothing else is a game tree.

Let us call the game tree of the form $<>$ a germ.

On a game tree $\Gamma$, the sub-game trees on $\Gamma$ are defined as:

1) $\Gamma$ itself is sub-game tree on $\Gamma$ of the depth 0.

2) If $\Gamma =$ $< x_1x_2 \cdots x_n >$ then $x_i(i \in \{1, \ldots, n\})$ is a sub-game tree of the depth 1.
3) Every sub-game trees on a sub-game tree $\Gamma'$ on $\Gamma$ are sub-game trees on $\Gamma$, where if the depth of the sub-game tree is $k$, then all of the depths of the sub-game trees of $\Gamma'$ are $k+1$ on $\Gamma$.

4) No exception exists.

Obviously a sub-game tree of some game tree is also a game tree.

**Definition 2 (Set of sub-game trees).** Let $\Gamma$ be a game tree. Let the set $\text{Sub}_k(\Gamma)$ be indices of sub-game trees of the depth $k$ on $\Gamma$. Namely, the procedure to construct $\text{Sub}_k(\Gamma)$ is, fining out all sub-game trees of the depth $k$ on $\Gamma$, attaching indices to each on it (from left side through right side), and picking up all of the indices. Let us write $\text{Sub}(\Gamma)$ to denote $\bigcup_{k \in \omega}\text{Sub}_k(\Gamma)$. The largest value of the depth of sub-game trees on the game tree is called the depth of the game tree.

![Fig. 1. An example of Game Tree](image)

text:

**example:**
Consider the example $\Gamma = <<<><><><><><><><><>$. Intuitively this can be represented by the “tree” shown in the Figure 1. $\text{Sub}_1(\Gamma)$ is $\{10, 11, 12\}$ and so on.

Given a game tree $\Gamma$, let us denote all of the indices of germ on $\Gamma$ by $G(\Gamma)$.

**Definition 3 (Players).** Let a game tree $\Gamma$ be given. Fix an index set $\mathcal{I}$ and also fix a mapping $f$ on the set $\text{Sub}(\Gamma) - G(\Gamma)$ into $\mathcal{I}$. Under this condition, we call the elements of the indices the players on the game tree $\Gamma$.

**Definition 4 (Game).** Let a game tree $\Gamma$ be given. Let the players set on $\Gamma$ be $\mathcal{I}$ (adjointed with $f : \text{Sub}(\Gamma) - G(\Gamma) \to \mathcal{I}$). A game is a triple $\langle \Gamma, v, f \rangle$, where $v$ is a mapping $G(\Gamma) \to \mathbb{R}^m$.

Given a game $\langle \Gamma, v, f \rangle$, $\{f^{-1}(i)\}_{i \in \mathcal{I}}$ is a partition on $\text{Sub}(\Gamma)$. I will call the element $\Gamma' \in f^{-1}(i)$ a stage for the player $i$. Also on the game, $v(x)$ for some germ $x$ on $\Gamma$ is thought to be a finite sequence of values: $(v_1, \ldots, v_m)$. I adapt the

\footnote{In this paper $\mathbb{R}$ always denotes the natural numbers.}
notation $v_i(x)$ to represent value of projection $P^i((v_i)_{i\in I})$. Following the manner in game theory, I will call the value $v_i(x)$ the payoff of the player $i(\in I)$ in the game $(\Gamma, v, f)$.

The following fact clearly holds. I will use it later.

**Fact 5.** In a game $(\Gamma, v, f)$, if $\Gamma' \in \text{Sub}(\Gamma'')$ holds for some sub game trees $\Gamma', \Gamma''$ in $\Gamma$, then there exists a unique sequence $(\Gamma_1, \cdots, \Gamma_k)$ such that $\Gamma_1 = \Gamma'', \text{Sub}_1(\Gamma_1) \ni \Gamma_2, \cdots, \text{Sub}_1(\Gamma_{k-1}) \ni \Gamma_k = \Gamma'$.

**Proof.** Obvious(induction with respect to the depth of $\Gamma$). $\square$

The common notion of “strategy” in game theory is to be redefined below. Assume $(\Gamma, v, f)$ is given.

**Definition 6 (Actions and Strategies of the players).** For a game $(\Gamma, v, f)$, an action of a player $i \in \text{cod}(f)$ is a finite set $A_i$ adjoined with a surjective mapping (I will call it a result map of the agent $i$).

$$\alpha_i : f^{-1}(i) \times A_i \xrightarrow{\text{(onto)}} \bigcup_{\Gamma' \in f^{-1}(i)} \text{Sub}_1(\Gamma')$$

such that $\alpha_i(\Gamma', a) \in \text{Sub}_1(\Gamma')$ always holds. A strategy of player $i$ is a mapping:

$$\sigma_i : f^{-1}(i) \rightarrow A_i.$$ 

For a player $i$, a strategy $\sigma_i$ determines his/her action on each stage $\Gamma' \in f^{-1}(i)$. I use $\Sigma_i$ to denote the set of all possible strategies of agent $i$.

Now, let us review the formal definition of “information structure($\mathcal{I}, \mathcal{S}$, and so on).

**Definition 7 (Information Structure).** An information structure is a pair $\langle \Omega, P \rangle$, where $\Omega$ is a set and $P : \Omega \rightarrow 2^\Omega$.

Following the usual interpretation, $\Omega$ is thought to be all of the states of the world and $P$ is sometimes called probability correspondence, for it is an abstraction of inability of human being to notice the precise state in the world. The easiest interpretation of it would be, if the state $\omega$ is the case, then $\omega', \omega'' \in P(\omega)$ if and only if the agent who has the probability correspondence $P$ can not distinguish $\omega'$ from $\omega''$.

In the situation of a game, each of the agents(namely “players”) is to be considered to have his/her probability correspondence respectively, but to have the set of the states of the world in common. Let us write $\langle \Omega, P_i \rangle$ to denote the information structure of player $i$.

**Definition 8 (Model).** Let us be given a game $(\Gamma, v, f)$. Also, for each $i$ in $\text{cod}(f)$ (codomain of $f$ in the game, namely, the players in it), let him/her have his/her information structure $\langle \Omega, P_i \rangle$ respectively. A strategy profile of an agent $i \in \text{cod}(f)$ is a function $s_i : \Omega \rightarrow \Sigma_i$. A model of the game $(\Gamma, v, f)$ is a
tuple $(\Omega, \{P_i\}_{i \in \text{cod}(f)}, s)$, where $s$ is said to be strategy profile of the model, which maps on $\Omega$ into $\Pi_{i \in \text{cod}(f)} \Sigma_i$ satisfying $P^i(s(\omega)) = s_i(\omega)$\footnote{Of course, if the set of players is enumerable, $s(\omega)$ is considered to be the sequence of strategies: $(s_1(\omega), \ldots, s_m(\omega), \ldots)$.}, where $P^i$ is a projection.

### 2.2 Structure of Players

Now I try to define a structure of players in order to make it possible to “characterise” them. The “characters” seems to be divided into two kinds, one for the “sensitivity” of “feedback” and another for the basic “strategy” of the player. The characteristic of the former case is defined on the bases of the information they own and some mechanism of “feedback” of result of previous game to the information. In this subsection I will introduce the basic context of former case, that is, formal definitions of “information” owned by the players. After the formalisation, the mechanism of “feedback” will be a conceptual problem of “how people usually learn a case for the next time”.

They may be essentially crucial because they could determine the global behaviour of the simulations which will be mentioned in the next stage of this study.

Given an information structure $\langle \Omega, P \rangle$, $E \subseteq \Omega$ is in general called an event and the following definition is usually used to describe “a player knows $E$”.

**Definition 9.** Given a player $i$ adjoined with his/her information structure $\langle \Omega, P_i \rangle$, define a mapping $K_i : 2^{2^\Omega} \rightarrow 2^{2^\Omega}$ as:

$$K_i : E \mapsto \{ \omega \mid P_i(\omega) \subseteq E \}.$$ 

The player $i$ is said to know the event $E$ in $\omega$ if $\omega \in K_i(E)$. Also, $K_i(E)$ is the event that $i$ knows that $E$ happens.

I regard that all games in this paper are complete information games, meaning for each player, the structure of games are known\footnote{Also, it is common knowledge ($\mathbb{K}, \mathbb{K}$ and so on) that every player know the structure. To be more precise, it also is common knowledge that every players can not break the structure, which means that the players follow rule of the game.}. Based on this assumption, we can make all players be rational, which means a player always take the action which make his/her payoffs best. If the information is perfect, namely $K_i(\{\omega\}) = \{\omega\}$ for each player $i$, then the game trivially has the result called backward induction solution, to which I will mention later.

Given a game $\langle \Gamma, v, f \rangle$ and its model $\langle \Omega, \{P_i\}_{i \in \text{cod}(f)}, s \rangle$, a player $i \in \text{cod}(f)$ is said to be aware of his/her strategy provided $\omega' \in P_i(\omega)$ always implies $s_i(\omega) = s_i(\omega')$. This assumption seems to be essential, because I am going to examine some kind of “behaviour” with will, but not “random walk”. In this study I assume this condition is always satisfied.

**example:** Let the game $\langle \Gamma, v, f \rangle$ be given, where the game tree $\Gamma$ is $<<><><><><><><><>>$, as is shown in figure 2. $\text{cod}(f) = \{A, B\}$, $G(\Gamma)$ =
\{G_1, \cdots, G_4\}, signs of bold characters are indices of \(\text{Sub}(\Gamma)\) and for each \(G \in G(\Gamma)\), \(v(G)\) is as shown in the figure.(Upper values are A’s and downer for B. \(v_A(G_3) = 3\) for example.) For both of A and B, the action set is identical: \(\{1, 2\}\). Also \(\text{Sub}(\Gamma) - G(\Gamma) = \{B_1, A_1, B_2\}\), \(f(B_1) = f(B_2) = B, f(A_1) = A\), \(\Sigma_A = \{(1, 2)\}\) and \(\Sigma_B = \{(1, 1), (1, 2), (2, 1), (2, 2)\}\). Result maps are as shown in the figure. Let \(\Omega = \{\omega_1, \cdots, \omega_8\}\) and suppose, \(s_A(\omega) = \begin{cases} (1) & \text{if } \omega \in \{\omega_1, \cdots, \omega_4\} \\ (2) & \text{o.w.} \end{cases}\) and \(s_B(\omega_1) = s_B(\omega_5) = (1, 1), s_B(\omega_2) = s_B(\omega_6) = (1, 2), s_B(\omega_3) = s_B(\omega_7) = (2, 1), s_B(\omega_4) = s_B(\omega_8) = (2, 2)\).

**Fig. 2.** An example: Game tree and information structure of the players

Now let us consider the information structures of them such as

\[
P_A(\omega) = \begin{cases} \{\omega_1, \cdots, \omega_4\} & \text{if } \omega \in \{\omega_1, \cdots, \omega_4\}, \\ \{\omega_5, \cdots, \omega_8\} & \text{o.w.} \end{cases}, \quad P_B(\omega) = \begin{cases} \{\omega_1, \omega_5\} & \text{if } \omega \in \{\omega_1, \omega_5\}, \\ \{\omega_2, \omega_6\} & \text{if } \omega \in \{\omega_2, \omega_6\}, \\ \{\omega_3, \omega_7\} & \text{if } \omega \in \{\omega_3, \omega_7\}, \\ \{\omega_4, \omega_8\} & \text{if } \omega \in \{\omega_4, \omega_8\}. \end{cases}
\]

Figure 2 describes this condition. These are the “worst” information structures in which they can be aware of their strategies.

In such condition like this, how should we predict the actions of the players? At least in this condition, for example, player A does not know the event that player B takes the action 2 in \(B_2\). Each of them just have to “take a chance”. In the context of this paper, the choice under such the imperfect information must also be reasonable and coherent. In order to let it have “reasonable solution” on imperfect information, the notion of expectation solution will be introduced later.

**Proposition 10.** Given a game \(\langle \Gamma, v, f \rangle\), there is a model for the game in which the set of states of the world \(\Omega\) satisfies:

\[
|\Omega| = \prod_{i \in \text{cod}(f)} \left[ \sum_{\Gamma'' \in f^{-1}(i)} |\text{Sub}_1(\Gamma'')| \right]. \tag{1}
\]
Proof. Obvious.

Let us call this state of the world primitive world. I employ the notation $\Omega^{\Gamma,f}$ to denote it. Also, the probability correspondence for each player can be decided so as for all of them to be aware of their strategy on the.

**Definition 11 (Primitive Model).** Let us be given a game $\langle \Gamma, v, f \rangle$. The primitive model for the game, denoted by $\langle \Omega^{\Gamma,f}, \{\ldots, P_i^{\Gamma,f}, \ldots\}, s \rangle$, is a model such that $\Omega^{\Gamma,f}$ is primitive world and for all $i \in \text{cod}(f)$, $s_i : \Omega^{\Gamma,f} \to \Sigma_i$ is surjective and

$$\omega' \in P_i^{\Gamma,f}(\omega) \iff s_i(\omega) = s_i(\omega').$$

**Proposition 12.** Given a game $\langle \Gamma, v, f \rangle$ and its primitive model $\langle \Omega^{\Gamma,f}, \{\ldots, P_i^{\Gamma,f}, \ldots\}, s \rangle$, for each $i \in \text{cod}(f)$, $P_i^{\Gamma,f}$ constructs a partition on $\Omega^{\Gamma,f}$ with each $\omega \in \Omega^{\Gamma,f}$, $\omega \in P_i^{\Gamma,f}(\omega)$.

Proof. 1) Trivially $\omega \in P_i^{\Gamma,f}(\omega)$, because $s_i(\omega) = s_i(\omega)$. 2) $\omega \in P_i^{\Gamma,f}(\omega') \& \omega \in P_i^{\Gamma,f}(\omega'')$ implies $P_i^{\Gamma,f}(\omega') = P_i^{\Gamma,f}(\omega'')$. In fact, if $\omega'' \in P_i^{\Gamma,f}(\omega')$ then $s_i(\omega'') = s_i(\omega')$. Also, the assumption implies $s_i(\omega') = s_i(\omega'')$, therefore $s_i(\omega') = s_i(\omega'')$ which implies $\omega'' \in P_i^{\Gamma,f}(\omega')$. Similarly $\omega'' \in P_i^{\Gamma,f}(\omega'')$ results $\omega'' \in P_i^{\Gamma,f}(\omega')$. 

**Proposition 13.** For a game $\langle \Gamma, v, f \rangle$, its primitive model is unique up to isomorphism.

Proof. "Isomorphism" between games is undefined in this paper. I do not go into details of the theory of "similarity" of the games. However it is easy to define "morphism" in some class of games. Here I demonstrate as follows: Assume both $\langle \Omega^{\Gamma,f}, \{\ldots, P_i^{\Gamma,f}, \ldots\}, s \rangle$ and $\langle \Omega''^{\Gamma,f}, \{\ldots, Q_i^{\Gamma,f}, \ldots\}, s'' \rangle$ are primitive models. Fix some $\omega \in \Omega$ and let $\omega' \in P_i^{\Gamma,f}(\omega)$. Then $s_i(\omega) = s_i(\omega')$ which yields $s_i'(h(\omega)) = s_i'(h(\omega'))$ for the isomorphism $h : \Omega^{\Gamma,f} \cong \Omega''^{\Gamma,f}$ and hence $h(\omega') \in Q_i(h(\omega))$, and so on.

**Definition 14 (Perfect Information Model).** A model $\langle \Omega, \{P_i\}_{i \in \text{cod}(f)}, s \rangle$ of a game $\langle \Gamma, v, f \rangle$ is said to be perfect information model provided $\Omega = \Omega^{\Gamma,f}$ and for every $i \in i \in \text{cod}(f)$ $P_i(\omega) = \{\omega\}$ for all $\omega \in \Omega$.

This is of course, every player is aware of each state itself, the awareness of other players’ awareness, and so on. This condition is usually mentioned that the rationality of every player is common knowledge([3], [4] and so on). I do not mention to the notion in this study.

In fact, a player is able to evaluate (to attach a value to the result of) his/her actions under imperfect information. Before formalising this process, several prerequisite results/notion are to be introduced.
Definition 15 (Sub Game Occurrence). Given a game \( \langle \Gamma, v, f \rangle \) and its model \( \langle \Omega, \{ P_i \}_{i \in \text{cod}(f)}, s \rangle \), \( E_{\Gamma'} \subseteq \Omega \) for some \( \Gamma' \in \text{Sub}(\Gamma) \) is an event which satisfies following condition;

\[
\omega \in E_{\Gamma'} \iff (\exists (a_{i_1}, \ldots, a_{i_k}) \in A_{i_1} \times \cdots \times A_{i_k})
\]

\[
\begin{bmatrix}
\alpha_{i_1}(\Gamma, a_{i_1}) = \Gamma_1 & \text{and} & a_{i_1} = s_{i_1}(\omega)(\Gamma) \\
\alpha_{i_2}(\Gamma_1, a_{i_2}) = \Gamma_2 & \text{and} & a_{i_2} = s_{i_2}(\omega)(\Gamma_1) \\
\vdots \\
\alpha_{i_k}(\Gamma_{k-1}, a_{i_k}) = \Gamma_k = \Gamma' & \text{and} & a_{i_k} = s_{i_k}(\omega)(\Gamma_{k-1})
\end{bmatrix}
\]

(2)

where \( A_{i_1}, \ldots \) stand for the action sets of \( i_1, \ldots \) respectively, \( \alpha_{i_1}, \ldots \) for result maps of respective agents and \( s_{i_1}, \ldots \) for strategy profiles of them. I call the event \( E_{\Gamma'} \) an Occurrence of sub game \( \Gamma' \).

The meaning of a sub game occurrence \( E_{\Gamma'} \) is the event (subset of states of the world) in which strategies of the players in the state make sub game tree \( \Gamma' \) occur in the process of original game.

Lemma 16. Let a game \( \langle \Gamma, v, f \rangle \) be given. For \( \Gamma', \Gamma'' \in \text{Sub}(\Gamma)(\Gamma' \neq \Gamma'') \),

1) if \( \Gamma' \in \text{Sub}(\Gamma'') \), then \( E_{\Gamma'} \subseteq E_{\Gamma''} \).

2) Otherwise, \( E_{\Gamma'} \cap E_{\Gamma''} = \emptyset \).

Proof. 1) : Pick up \( \omega \in E_{\Gamma'} \) arbitrarily, which means(from \( \Gamma' \)), there is a sequence \( (a_{j_1}, \ldots, a_{j_l}) \) such that

\[
\begin{align*}
\alpha_{i_1}(\Gamma, a_{j_1}) = \Gamma_1' & \quad \text{and} \quad a_{j_1} = s_{j_1}(\omega)(\Gamma) \\
\alpha_{i_2}(\Gamma_1', a_{j_2}) = \Gamma_2' & \quad \text{and} \quad a_{j_2} = s_{j_2}(\omega)(\Gamma_1') \\
& \vdots \\
\alpha_{i_l}(\Gamma_{l-1}', a_{j_l}) = \Gamma_l' = \Gamma' & \quad \text{and} \quad a_{j_l} = s_{j_l}(\omega)(\Gamma_{l-1}').
\end{align*}
\]

(3)

The goal : \( \omega \in E_{\Gamma''} \), so we have to find out some sequence of actions \( (b_{g_1}, \ldots, b_{g_f}) \) which satisfies

\[
\begin{align*}
\alpha_{g_1}(\Gamma, b_{g_1}) = \Gamma_1'' & \quad \text{and} \quad b_{g_1} = s_{g_1}(\omega)(\Gamma) \\
\alpha_{g_2}(\Gamma_1'', b_{g_2}) = \Gamma_2'' & \quad \text{and} \quad b_{g_2} = s_{g_2}(\omega)(\Gamma_1'') \\
& \vdots \\
\alpha_{g_f}(\Gamma_f'', b_{j_f}) = \Gamma_f'' = \Gamma'' & \quad \text{and} \quad b_{j_f} = s_{j_f}(\omega)(\Gamma_f'').
\end{align*}
\]

(4)

Claim : \( (\Gamma, \Gamma_1', \ldots, \Gamma_f' (= \Gamma')) \) is unique. Because \( \Gamma' \in \text{Sub}(\Gamma) \) and from the definition of result maps, \( \Gamma_1' \in \text{Sub}_1(\Gamma) \ldots \) hold. Similarly \( (\Gamma, \Gamma_1', \ldots, \Gamma_f' (= \Gamma'')) \) must be unique. From the assumption : \( \Gamma' \in \text{Sub}(\Gamma'') \), Fact yields some unique sequence \( (\Gamma_1, \ldots, \Gamma_k (= \Gamma')) \). Then,

\[
(\Gamma, \Gamma_1', \ldots, \Gamma_f') = (\Gamma, \Gamma_1'', \ldots, \Gamma_f'', \Gamma_1, \ldots, \Gamma_k),
\]
must hold, because otherwise there are two sequences both of which satisfies 
\( \ldots, \Gamma_3' \in \text{Sub}_1(\Gamma_2'), \ldots, \Gamma_4'' \in \text{Sub}_1(\Gamma_3'') \ldots \) and so on, causing contradiction to Fact 5. Consequently chopping the sequence \((a_j_1, \ldots, a_j_l)\) (or, totally \((a_j)\)) off after \(f+1\)-th, letting \((b_g, \ldots, b_g') = (a_j_1, \ldots, a_j_l)\), and then we get the sequence which satisfies \((4)\).

\[ \square \]

2) Suppose \(\omega \in E_{i''} \cap E_{i'''}\). Definition 15 yields some sequences \((a_1, \ldots, a_n)\) and \((a_1', \ldots, a_m')\) respectively, which satisfy \((2)\). Definition 3 and \((2)\) require \((a_1, \ldots, a_n) \neq (a_1', \ldots, a_m')\). Suppose, \(l < n \& l < m\) and for all \(i \in \{1, ..., l - 1\}\) \(a_i = a_i'\), and in \(l, a_l \neq a_l'\). Then, for \(\Gamma_{l-1}\) such that

\[
(\text{al-1}(\Gamma_{l-2}), \text{al-1}) = \Gamma_{l-1} \text{ and } \text{al-1} = \text{s}_{l-2}(\omega)(\Gamma_{l-2}),
\]

\[ a_l = \text{s}_{l-1}(\omega)(\Gamma_{l-1}) = a_l' \text{ must hold, which is contradiction}. \]

\[ \square \]

**Lemma 17.** Let a player \(i\) be fixed, on given game \(\langle \Gamma, v, f \rangle\) adjoined with a model \(\langle \Omega, \{P_i \}_{i \in \text{cod}(f)}, s \rangle\). Define a function \(\tilde{P_i} : f^{-1}(i) \rightarrow 2f^{-1}(i)\) as follows. Suppose \(\Gamma \in f^{-1}(i)\). Then:

\[
\Gamma'' \in \tilde{P_i}(\Gamma) \Leftrightarrow_{\text{df}} K_i(E_{\Gamma}) \subseteq K_i(E_{\Gamma''}).
\]

Let us call this function possible stage correspondence. For this function, (let \(\Gamma''\) be arbitrary in \(f^{-1}(i)\)):

\[
\Gamma'' \in \tilde{P_i}(\Gamma) \quad \text{and} \quad \Gamma'' \in \tilde{P_i}(\Gamma) \Rightarrow \tilde{P_i}(\Gamma'') \subseteq \tilde{P_i}(\Gamma)
\]

holds.

**Proof.** For \((\ref{def:meta}), K_i(E_{\Gamma''}) \subseteq K_i(E_{\Gamma''})\) trivially holds. As for \((\ref{def:partition}), \Delta \in \tilde{P_i}(\Gamma''')\) yields \(K_i(\Gamma'') \subseteq K_i(E_{\Delta})\) from definition. Also, precondition of \((\ref{def:meta})\) implies \(K_i(E_{\Gamma}) \subseteq K_i(E_{\Gamma'''})\). Consequently \(K_i(E_{\Gamma}) \subseteq K_i(E_{\Delta})\) which is the definition of \(\Delta \in \tilde{P_i}(\Gamma)\).

\[ \square \]

The statements I introduced so far in this section (especially Lemma 17) are, in fact, to be used to justify some reasonable “partition” on \(f^{-1}(i)\) for some player \(i\). Under the given condition (I mean \(P_i\) for a player in some model), there should be some class of stages on each of which the stages are indistinguishable. If some procedural way to construct such partition for each player can successfully be established, the player can attach a calculation result of estimation value for each action on “the stages’. Lemma 17 indicates one way to construct such class on \(f^{-1}(i)\). In this point, \(\{\tilde{P_i}(\Gamma)\}_{\Gamma \in f^{-1}(i)}\) itself is not necessarily a partition.

**Definition 18 (Strategy Evaluation).** Given a game \(\langle \Gamma, v, f \rangle\) adjoined with a model \(\langle \Omega, \{P_i \}_{i \in \text{cod}(f)}, s \rangle\). Assume that the possible stage correspondence \(\tilde{P_i} : f^{-1}(i) \rightarrow 2f^{-1}(i)\) defined in Lemma 17 generates a partition \(\{\tilde{P_i}(\Gamma)\}_{\Gamma \in f^{-1}(i)}\). Let us call a function

\[
e_i : \{\tilde{P_i}(\Gamma)\}_{\Gamma \in f^{-1}(i)} \times A_i \rightarrow \mathbb{R}
\]

a strategy evaluation.
I would emphasise again that every player is rational, hence the action evaluation function should directly affect the choice of action of the owner. It means, for each player \( i \), he/she chooses a desirable states \( P_i(\omega) \) in which \( e_i \) gets maximal estimation payoff, as far as he/she can.

**Proposition 19.** Given a game \( \langle \Gamma, v, f \rangle \) and its primitive model \( \langle \Omega^{\Gamma,f}, \{P_i^{\Gamma,f}\}_{i \in \text{cod}(f)}, \mathcal{S} \rangle \), a strategy evaluation determines one and only one state in \( \Omega^{\Gamma,f} \).

**Proof.** Firstly, using (1) of proposition 10, a bijective map :

\[
\phi : \Omega^{\Gamma,f} \equiv \prod_{i \in \text{cod}(f)} \left[ \bigcup_{\Delta \in f^{-1}(i)} \text{Sub}_1(\Delta) \right]
\]

can be build as follows:

\[
\phi(\omega) = ( (\Delta^1_1, \cdots, \Delta^1_{m_1}), (\Delta^2_1, \cdots, \Delta^2_{m_2}), \cdots, (\Delta^n_1, \cdots, \Delta^n_{m_n}) ) \leftrightarrow \bigwedge_{j \in \{1, \ldots, n\}} \left[ \alpha_j(\Gamma^j_1, s_j(\omega)(\Gamma^j_1)) = \Delta^j_1 \wedge, \cdots, \alpha_j(\Gamma^j_{m_j}, s_j(\omega)(\Gamma^j_{m_j})) = \Delta^j_{m_j} \right] \quad (7)
\]

here \( \Gamma^j_1 \) etc. represent elements of \( f^{-1}(j) \), \( \{1, \ldots, n\} = \text{cod}(f) \), and so on.

For each \( i \), assume his/her decision(based on strategy evaluation) is \( \hat{a}_i \). In a primitive model \( s_i \) is surjective, therefore there exists \( \omega^i \in \Omega^{\Gamma,f} \) such that \( s_i(\omega^i)(\Gamma''') = \hat{a}_i \), for all \( \Gamma''' \in \hat{P}_i(\Gamma') \) \( (= f^{-1}(i) \), for we are thinking about a primitive model.) Let

\[
\left[ \begin{array}{c}
\alpha_i(\Gamma^i_1, s_i(\omega^i)(\Gamma^i_1)) = \Delta^i_1 \\
\vdots \\
\alpha_i(\Gamma^i_{m_i}, s_i(\omega^i)(\Gamma^i_{m_i})) = \Delta^i_{m_i}
\end{array} \right]
\]

where \( \{\Gamma^i_1, \cdots, \Gamma^i_{m_i}\} = f^{-1}(i) \). Doing similarly for each player in \( \text{cod}(f) \), we can build a sequence : \( ((\Delta^1_1, \cdots, \Delta^1_{m_1}), (\Delta^2_1, \cdots, \Delta^2_{m_2}), \cdots, (\Delta^n_1, \cdots, \Delta^n_{m_n})) \), which is a member of \( \text{cod}(\phi) \). Thus, we obtain some

\[
\omega' \in \phi^{-1}(((\Delta^1_1, \cdots, \Delta^1_{m_1}), (\Delta^2_1, \cdots, \Delta^2_{m_2}), \cdots, (\Delta^n_1, \cdots, \Delta^n_{m_n}))),
\]

because of the bijectivity of \( \phi \).

Assume \( \omega', \omega'' \in \Omega^{\Gamma,f} \) such that \( \omega' \neq \omega'' \) results \( s_i(\omega')(\Gamma'') = s_i(\omega'')(\Gamma'') = \hat{a}_i \) for every \( \Gamma''' \in \hat{P}_i(\Gamma') \) \( (= f^{-1}(i) \). Then, (using projection, )

\[
P^i(\phi(\omega)) = \left( \begin{array}{c}
\alpha_i(\Gamma^i_1, s_i(\omega)(\Gamma^i_1)), \\
\vdots \\
\alpha_i(\Gamma^i_{m_i}, s_i(\omega)(\Gamma^i_{m_i}))
\end{array} \right),
\]
yields the following:
\[
P_i^t(\phi(\omega')) = P_i^t(\phi(\omega'')) = \begin{pmatrix}
\alpha_i(\Gamma_i^1, \hat{a}_i) \\
\vdots \\
\alpha_i(\Gamma_i^{m_i}, \hat{a}_i)
\end{pmatrix}, \text{ hence } \phi(\omega') = \phi(\omega''),
\]
which is contradiction. \(\square\)

**Fig. 3.** An example of primitive world

**example:** The right side of figure 3 shows a primitive world of the given game \(\langle \Gamma, v, f \rangle\) which is described in the left of the figure. \(\text{cod}(f) = \{A, B, C\}\), \(f^{-1}(A) = \{A\}\) and so on, actions for each player is common; \(\{1, 2\}\), and

\[
v_A(\alpha_C(C1, s_C(2112)(C1))) = 1
\]

for example. (That is, say \(\omega \in \Omega\) is 22111, then it represents \(\phi(\omega)\) which is introduced in proposition 19, namely \(\{\alpha_A(A, 2), ((\alpha_B(B), 2)), ((\alpha_C(C1), 1), (\alpha_C(C2), 2))\}\).) According to proposition 19 we can identify only one state which decides action for each player. In fact, for A, if he/she employ a simple and common “estimation value” for his/her strategy evaluation,

\[
e_A(\{A\}, x) = \begin{cases}
\frac{1}{2}(5 + 4) = \frac{9}{2} & \text{if } x = 1, \\
\frac{1}{2} \times 8 + \frac{1}{2} \times (1/2 + 3/2) = 5 & \text{if } x = 2.
\end{cases}
\]

Therefore the player A is to choose the strategy 2. Similarly,

\[
e_B(\{B\}, x) = \begin{cases}
2 & \text{if } x = 1, \\
\frac{9}{2} & \text{if } x = 2.
\end{cases}
\]

\[
e_C(\{C1, C2\}, x) = \begin{cases}
1/2 + 1/2(1/2 \times 4 + 1/2 \times 6) = 3 & \text{if } x = 1, \\
1/2 \times 6 + 1/2(1/2 \times 4 + 1/2 \times 1) = \frac{17}{4} & \text{if } x = 2.
\end{cases}
\]

As a result, A – 2, B – 2 and C – 2 are chosen respectively, whence 2222 in \(\Omega\) is the resulting state.
3 Conclusion: Perspective and Direction

For a set of agents, which plays the role of “player in a game” and also, can derive an “optimal” solution as far as they can, Proposition 19 provides the “first stage” to them. To be precise, in the moment in which a particular game is firstly given, the knowledge of each of them about others’ knowledge should be almost empty. The emptiness is to be represented by the primitive model. Even in such kind of uncertain situation, we can define one particular “state” which corresponds to the first decision making for the agents. From the state, I consider the iteration of games should begin. In the process of “iteration”, the information structure of each agent is to become “more precise”. In the following, I will introduce the rough story of that idea.

3.1 Around Backward Induction

A condition of given game and its model in which “backward induction” is possible for each player, should be thought of as “final situation” of the game. Until then, the players must have some process to “learn” the others’ knowledge. It is easy to define the notion of backward induction ([4] and so on) using the formulation I have introduced.

Definition 20 (Backward Induction Solution). Given a game \(\langle \Gamma, v, f \rangle\), the backward induction solution of the game \(\Gamma\), denoted by \(BI(\Gamma)\) is the germ \(g \in G(\Gamma)\) defined as :

1) If \(\Gamma = <>\) (namely, a germ) then \(BI(\Gamma) = g\).
2) Otherwise, \(BI(\Gamma) = BI(\Gamma')\), where

\[
\Gamma' \in \text{Sub}_1(\Gamma) & (\forall \Gamma'' \in \text{Sub}_1(\Gamma))[v_f(\Gamma)(BI(\Gamma')) > v_f(\Gamma)(BI(\Gamma''))].
\]

In a perfect information game, it may be natural to expect that the final result should be that of “backward induction”. Actually the situation can be stated as follows:

Definition 21 (Result Germ). Given a game \(\langle \Gamma, v, f \rangle\) and its model \(\langle \Omega, \{P_i\}_{i \in \text{cod}(f)}, s \rangle\), define a function \(G : G(\Gamma) \rightarrow 2^\Omega\) as follows :

\[
\omega \in G(g) \Leftrightarrow (\exists (\Gamma, \Gamma_1, \cdots \Gamma_{k-1}, g)) \left[ \begin{array}{c}
\alpha_{i_1}(\Gamma, s_i(\omega)(\Gamma)) = \Gamma_1 \\
\vdots \\
\alpha_{i_k}(\Gamma_{k-1}, s_i(\omega)(\Gamma_{k-1})) = g.
\end{array} \right]
\]

Proposition 22. Given a game \(\langle \Gamma, v, f \rangle\) and its model \(\langle \Omega^\Gamma,f, \{P_i\}_{i \in \text{cod}(f)}, s \rangle\) where \(\Omega^\Gamma,f\) is a primitive world, if the model is a perfect information model, then each player has its strategy evaluation function. Furthermore, let \(\hat{\omega}\) be the state which is determined following the manner of Proposition 19. Then, \(\hat{\omega} \in G(BI(\Gamma))\) always holds.

Proof. It is intuitively obvious. I do not mention the formal proof here. \(\square\)
3.2 Process: In a Game and in the Iteration of the Game

Let us go back to the example shown as figure 3. Before players take action, as we have seen, the state was 2222. But once the action begins, namely a process of the game begins, some indicative situation seems to occur. Imagine the stage of B, which happens after player A take the action 2. If the players are aware of “process” itself, then B should be aware of the situation that “A took 2, and now it is my stage”. As for B, his/her only one stage is B and he/she will take 2 following the result of strategy evaluation. So, what about C? He/She also is aware of the process that at first A took 2 and in the next B also took 2. But for him/her just following the result of 2222 causes contradiction, for, despite he/she is aware of the process (at first A took 2 and in the next B also took 2), he/she is going to take 2, which results his/her payoff to be 1. (Of course at that point taking 1 brings him/her the payoff 6.) This seems to be a violation of the assumption that players are “rational”.

The state should be moved. To be concrete, in the stage $C_1$, (or in every stages for the players,) he/she (they) re-calculate the estimated value of the action, based on the fact that “he/she is in that stage”. In the case we are considering, the state $\omega$ is to be determined so as to

$$v_C(\alpha_C(C_1, s_C(\omega)(C_1)))$$

maximises. In this case, 2212 is one alternative. Consequently, during the process of one particular game, it is natural to think that the state may move, if we assume the players to be aware of “process of the game”.

Now, let us imagine the primary game has finished, in which in the final stage, C has chosen the action 1. Before that stage, A and B were not aware of the “rationality of C”. After they got the result of 2212, it is natural to consider that “both of A and B is now aware of C’s rationality, cause they saw that C took 1 in the stage $C_1$.” So, what we should anticipate the situation in the 2nd play of the iteration is, the change of $P_A$ and $P_B$, because they now know that C is rational. In this manner, if the iteration of “same game” as the example goes, the final situation of the model (namely $\{P_A, P_B, P_C\}$) will be brought. Simple calculation shows that it is perfect information.

3.3 Direction

Following points seem to be worth while being investigated.

- It is not clear at all if an iteration of arbitrary game always bring the situation of perfect information or not. I am considering both the theoretical and experimental approach to the issue.
- Generating arbitrary games, giving them to the fixed set of players, observing information structures of them.
- Variation of strategy evaluation: Cooperative, selfish, moralistic in a cooperation, totally moralistic, and so on.
- etc., etc......
In spite of the fact that it is intuitively clear that the agent adjoined with information structure is far more complex than automata-type agent, the essential difference from, for example, traditional iteration of one-shot games, is quite ambiguous. Concerning to the study so far, it seems to be possible to investigate with purely theoretical way. But it is a matter of strategy of study itself.

It is only the beginning stage of the study. I would be appreciated if I could have indicative or critical advises.

References

Towards a Generic Model of Information Processing Systems

Gábor Németh

Budapest University of Technology and Economics
nemeth@hit.bme.hu

Abstract. The convergence of communication and computer systems leads to new architectures, such as the intelligent network. Such a network provides information processing, retrieval and transfer services for the user in a transparent way. Its structure is based on nodes with various information processing models and performances. The structure is dynamically changing because of failures and modifications, with static or dynamic task allocation over inhomogeneous or homogeneous resource types. Such a system cannot be handled by the classical models. A generic model is presented having logical, temporal and topological structures together with operations on these structures. The necessary mathematical concepts are parameterised hierarchical relations, logic functions, hierarchies of variables with their hierarchical control operators, and neighbourhood/similarity structures. Introducing a model time as a partial ordered set of time instants, an evolutionary system can be considered, both its capabilities and its active structure.

1 Introduction

The advancement of computing and telecommunications lead to large distributed systems with nodes based on various information processing models and with various performances. Besides, the topology is changing dynamically because of failures and/or modifications. As a consequence, the classical formal models, limited to well defined specific problem classes must be generalised [1].

The generic element of the formal model proposed is the knowledge base system. A knowledge base system is a database system with logical, temporal and topological structures together with operations on these structures. The necessary mathematical concepts for modelling such a system are parameterised hierarchical relations, logic functions, hierarchies of variables with their hierarchical control operators, and neighbourhood/similarity structures. These concepts are then applied to define a model of a knowledge module. By composition of knowledge modules the knowledge base system (KBS) is obtained.
2 Basic Concepts and Definitions

The generic system model KBS consists of the following components:

- **S** a set of primitive objects,
- **S(S)** a hierarchy of relations over S, all parameterised (referenced) by indices of hierarchically structured index sets,
- **F(S)** an explicitly given part of S(S), the facts,
- **D(S) = S(S) \ F(S)** the implicitly given part of S(S), obtainable by composite applications of functions of R,
- **R** a set of inference, deduction rules, the application of which is in general subject to constraints, conditions, grammatical rules, collected in a set
- **Γ(R)** the grammar of R.

**var x** variables defined on sets of components on all hierarchical levels.

**P** control functions; assignments to variables and reciprocal, reassignments to substitutable components are performed by a hierarchy of control functions and their reciprocals, whereby a control function val: P×{var x} → X is associated with each variable var x, and where X = {x[p] | p ∈ P} is the variability domain (type) of var x and P is a set of control parameters p. (An assignment to var x is then expressed by val(p, var x) = x[p]). Domains of variables can contain variables of lower hierarchical level and variables can be defined on sets of lower level control parameters of variables.

To operate on the components of KBS a set of operations **OP** has to be given (e.g. selectors like subset forming, projections, cuts, selection of substructures by properties, constructors like set forming, set products, set union, set intersection, concatenation of relations, and transformations of objects and indices, counting cardinalities).

A grammar **Γ(OP)** for the application of operations of OP may be given.

To express structural properties of KBS, we need

**PR** a set of predicates, e.g. generalised quantors, is part of property, etc.

Given a partially or linearly ordered logical or physical model time **(T, <)** all components of KBS can be indexed by time points and processes (KBS_t)_{t ∈ U ⊆ T} with varying states KBS_{t[p]} at time points t ∈ U ⊆ T can be considered. Temporal properties can be adjoined to P.

Finally, on each hierarchical level, sets of objects, rules and parameters can be topologised by introducing a topological structure (e.g. general distance or similarity measures). (In engineering topological structures are used under the name fuzzy.)

3 Knowledge Representation

In the human mind knowledge is represented in memorised perceptions, concepts, behavioural and intellectual processes. Physically structured physical objects in space-time dimension, which we are able to interpret, represent it. Mathematically, these objects are abstracted and represented by normed symbols in mathematical space,
subject to mathematical operations [2]. The time dimension is mapped onto ordering in space.

Parameterised sets (families, relations) and operations on these can be used for the mathematical description. One of the key features of the model proposed is the utilisation of variables. A KBS is represented by a hierarchy of variables and their control functions.

Starting on top, we consider:

\[ \text{var KBS} = (\text{var S, var S(var S), var F(var S), var D(var F), var R(var F), var } \Gamma(\text{var R}), \text{var OP, var } \Gamma(\text{var OP}), \text{var P, var } (T, <). \]

All variables var x range on given domains X parameterised by \( P[X] \) and have control functions \( \text{val: } P[X] \rightarrow X \) with control parameters \( p \in P[X] \).

The assignment steps in logical time are:

\[ \text{var } S := S \neq \emptyset, \text{selection of the primitive objects; var } I := I \neq \emptyset, \text{selection of the primitive indices;} \]

\[ \text{var } OP := OP, \text{var } \Gamma(\text{var OP}) := \Gamma(OP), \text{ selection of admitted structors for } S; \]

\[ \text{var } P := P \text{ selection of structural predicates; } \]

For bottom up construction of the hierarchy F:

\[ \text{var } F(0) := \text{pow S}\setminus\emptyset, \text{selection of var } F(0) := F(0) = \emptyset; \]

\[ \text{var } N : N, \text{ var } N := N, \text{ for } n = 0,1,2,...N-1: \]

\[ \text{var } I^{(n+1)} := \text{pow I}\setminus\emptyset, \text{selection of var } I^{(n+1)} := I^{(n+1)}; \]

\[ \text{var } F^{(n+1)} := \bigcup_{J \subseteq \text{var } I^{(n+1)}} (F^{(n)})^J \setminus\emptyset, \text{selection of var } F^{(n+1)} := F^{(n+1)}, \]

\[ \text{var } F(n+1) :=: F(n) \cup F^{(n+1)}; \]

Selection of admitted rules:

\[ \text{var } R(F(N)) := R(F(N)), \text{ var } \Gamma(R(F(N))) := R(F(N)). \text{ Assignments to composite variables can be performed in partial steps. } \]

This results in var KBS := KBS.

The deduction steps in logical time are:

\[ \text{var } f(var D) := R \text{ with } \Gamma(R). \text{ Selection of f : var } f := f, \text{ follows var } D := D, \text{ var } W := f(D). \text{ Selection of an argument: var } d: D, \text{ var } d := d, \text{ evaluation of var } w := w = f(d). \]

Decision on operation on (F, w): var op(F, w) := OP with \( \Gamma(OP) \), var op(F, w) := op(F, w).

As an illustration let us consider knowledge represented by binary relations with valuated elements. If \((y, x)\) is a proposition (object \( y \) has property \( x \)), it can be valuated by \( v \in V = \{"t", "f"\}, \) yielding \((y, x, v)\). This includes of course composite objects (relations) \( y \) and composite properties (relations) \( x \) and arbitrary sets \( V \) with any structures. Given a valuation \( v' \) to \( y \), a valuation \( v'' \) to \( x \), and a function \( \phi: (v', v'') \rightarrow v \), then to \((y, v'), (x, v'')\) can be assigned \((y, x, v)\), see Fig. 1.
Fig. 1. Knowledge represented by binary relations.

The valuations are of course arbitrary, thus extending the traditional data base concepts. If the knowledge module contains variables, e.g. \((y_j, x_i), \text{var } v_{ji}\), they express indeterminacy in the sense that the domain (type) of the variable is known but the value to be assigned is not yet determined. This case has to be distinguished from elements not appearing in the module, e.g. index pairs \((j', i') \in (J \times I) \setminus U\). Queries with variables to a module with variables in general result in answers with variables (Fig. 2).

Fig. 2. Extending the traditional data base concept by using variables.
The type of the items (e.g. properties) may also be variable (Fig. 3).

Fig. 3. Representation of properties of variable types.

For the query operations topological structures are used, but the topological structures may also be variable (Fig. 4).

Fig. 4. Variable topology used in query operation.
Composite matching operations are evaluated according to the relevant variable topologies [3], e.g.:
selecto where Weight > 115 or Eyecolor ~ “dark” (Fig. 5).

Fig. 5. Composite matching operation with variable topology.

Such knowledge modules can be used to compose a knowledge base system by concatenation, i.e. feeding (part of) the answer of one module as (part of a) query to the same or another module (Fig. 6).

Fig. 6. Composition of a knowledge base system.
Towards a Generic Model of Information Processing Systems

Time may be handled the following way [4]. Let us consider a linear or partial ordered set \((T, <)\) as time, and for simplicity binary modules which are time parameterised, i.e. a process \(((y_{tj}, x_{ti}), v_{tji})_{(ji)} \in U(t))\), with \(U(t) \subseteq J(t) \times I(t)\). Let us define

\[ I = \bigcup_{t \in T} I(t) \quad \text{and} \quad J = \bigcup_{t \in T} J(t), \]

then concatenation of the family of families \(((v_{tji})_{(ji)} \in U(t))\) yields \((v_{tji})_{(tji)} \in S\) with a suitable \(S \subset T \times J \times I\), i.e. a ternary module. In other words, history is handled just as another dimension in the hierarchy. Queries are then possible for example in \(T \times J\) and in \(T \times I\) space.

References

Evolutionary Information and Decision Support Systems: An Integration Based on Ontologies

Maria V. Hurtado and Jose Parets

Department of Languages and Information Systems
University of Granada. ETS Ingeniería Informática.
Avda. Andalucía, 38. 18071 - GRANADA (Spain)
{mhurtado, jparets}@ugr.es

Abstract. In order to use information stored in different repositories, it is necessary to improve the automatization process of decision-making. As a result, it would be necessary to share data. One of the main motivations of our emphasis on Semantic Views building is the possibility of sharing and reusing knowledge between the Information System, the Decision Support System and other external repositories. In this paper, we will try to show that Ontologies can be used in practice as a Semantic View. We also discuss some of the limitations of the Ontology described using Description Logics in a complex domain and the limitations which we have discovered when the intrinsic evolutionary aspects of Information Systems and Decision Support Systems must be modeled. Another important problem is how a change in such systems may produce a propagation of changes which affect the levels of the system (data) and the metasystem (data structure). For this purpose, we propose a specialized ontology based on the Object-Oriented Approach. Subsequently, we have obtained some equivalent heterogeneous graphs and these can be used to represent change propagation in the IS, DSS and Semantic Views. Most of the problems mentioned will be exemplified by means of a concrete case: a DSS for risk operations in financial institutions, the class structure of which is outlined.

1 Introduction

At present, it is widely accepted that information constitutes one of the most important resources in an organization, and information is undoubtedly the key element in the decision-making process. Under these premises, it is necessary to work with a large quantity of heterogeneous and distributed data in the automatization process of decision-making. In such a situation, Decision Support Systems (DSSs) are frequently developed in an *ad hoc* way which does not take into account the fact that part of these data are included in the Information System (IS).

In the first section, we will discuss the need to share these data between both ISs and DSSs in order to avoid problems of redundancy and those arising from updating information and change propagation. This obviously implies the classic problem which arises from the sharing of information.

---

1 This research is partially supported by a R+D project of the Spanish CICYT (TIC2000-1673-C06-04).

© Springer-Verlag Berlin Heidelberg 2001
This problem may be partially solved by providing semantic views over the repositories that hide all the technical and organizational details associated with data.

A study of the main possible mechanisms used in the design of semantic views will then be presented.

In the following section, as a result of research carried out in the last decade, Ontologies [12], [13] described through Description Logics (DLs) [9] are suggested as an interesting semantic representation, which can be used here.

However, although the DLs are apparently equipped with complete sound reasoning and terminating procedures, they still suffer from several limitations not only when they are used to represent complex domains but also efficient evolution mechanisms.

Consequently, a new semantic representation can be considered in order to take into account the evolution of both kinds of systems (Information and Decision Support).

Following the lines of previous papers on software evolution [18], [19], [20], [22], section 7 studies the possible evolution of semantic views to be considered and investigated by providing effective graph mechanisms for semantic evolution.

Throughout the paper, we will use an example based on the study of a DSS for risk operations in financial institutions. This example will help to understand many of the outlined problems and the proposed solutions.

2 Information Integration through Semantic Views

When Decision Support Systems are developed in an *ad hoc* way some important problems are produced:

- Redundancy problems: data is in both the IS and the DSS, most of the time in different formats.
- Update problems: changes produced in either the IS or the DSS are not reflected in the other.
- Change propagation problems: when data structures are modified, legacy data can remain inconsistent.

In order to avoid these problems nowadays, organizations need to have solved the problem of sharing information with the IS and with other repositories (available through the communication networks) when the DSS is developed and used. In this situation we do not overlook the fact that data included in different repositories is always interpreted and has a meaning which depends on the system. Obviously this entails the classic problem of information sharing. But the problem of sharing may be partially solved by providing semantic views over the repositories that hide all the technical and organizational details associated with the data. Furthermore, semantic views on specific domains reduce the problem of semantics associated with stored data [3]. Figure 1 shows a general scheme in which semantic views (ontologies) are used to integrate heterogeneous information content in the IS, DSS and other repositories.
3 Formalisms Used in Obtaining Semantic Views

Semantic views are usually expressed by using declarative knowledge representation languages. These languages provide syntax, a set of inference rules, a vocabulary of non-logical symbols, and sometimes a graphical representation that restricts the acceptable interpretations of the symbols in the vocabulary.

Different formalisms have been proposed in order to obtain these views:

- Systems based on DL, [9]: this kind of system takes an object-centered view, where the world is modeled as individuals connected by binary relationships and grouped into classes (called concepts). In every DL_System, the concepts of the application domain are described by means of concept descriptions. These are built from atomic concepts and roles using the constructors provided by the DL Language. DL-Systems support a variety of inference mechanisms, such as subsumption, inconsistency detection, memberships and others.

- Knowledge Based Systems descendants of the KL-ONE [6], [7], [23] called Terminological Systems, (i.e. BACK [21], CLASSIC [4], [5], Kris [2], LOOM [14]). These systems are used to make the terminology of an application domain explicit in a similar way to DL_Systems. In addition, Terminological Systems automatically classify these definitions and queries into a taxonomy according to semantic relations such as subsumption and equivalence. The queries also help to discover what the relevant repositories are.

- Ontologies [12],[13] are suggested as an interesting semantic representation, which can be used here, for several reasons:
  - They provide declarative and concise specifications of semantic information.
  - We can obtain a logical schema of the shared information, which reduces the semantic loss in classical data models. For instance, the relational approach does not
Evolutionary Information and Decision Support Systems 149

explicitly convey what the entities and the relationships are in the real model, since both are represented as tables. This situation implies different difficulties when common information belonging to more abstract artifacts is needed.

-They allow the information found in data repositories to be described independently of the underlying syntactic representation of data. Each data repository is viewed at the level of the relevant semantic concepts, and maintains a hierarchical organization of concepts which is very useful when dealing with large collections of definitions (a very simple case would be a type hierarchy which specified classes and their subsumption relationships). In this context, a mapping process of the ontology concepts and the repository terms can be used. The syntax and the semantics of the mapping can found in [15]. DSS and IS can express their information needs using these semantic views (ontologies) and the query processor must obtain the corresponding answer by accessing the underlying data repositories.

For all of these reasons and because ontologies have nowadays been made as unifying formalisms used in different domains, we advocate the use of ontologies as the Semantic View.

4 Understanding Ontologies

In the context of knowledge sharing, the term ontology is used as an explicit specification of a conceptualization [12]. That is to say, the ontology is a description of the concepts and relationships that can exist for an agent or a community of agents.

Ontologies are designed to enable knowledge to be shared and reused. Moreover, ontologies are agreements about shared conceptualizations. Shared conceptualizations include conceptual frameworks for modeling domain Knowledge. Ontological commitments are agreements to use the shared vocabulary coherently and consistently. Consequently, different kinds of systems sharing a vocabulary need not necessarily share a knowledge base.

In our case, a common ontology defines the vocabulary which allows queries and assertions to be exchanged between the IS, DSS and other repositories. Figure 2, which is explained in greater detail in the next section, shows the terms included in the ontology proposed, terms such as *Client, Transaction, Asset, Loan*, etc., which can be used for the IS and DSS in a Financial System.

Each system knows facts the other does not. For example the IS has information about *Financial_I or Branch*, which is not required by the DSS. On other hand the DSS has information about *Expedient* or data about *Non-client*, which is not necessary for the IS.

Furthermore, a system that commits to an ontology is not required to answer all the queries that can be formulated in the shared vocabulary.

In short, a commitment to a common ontology is a guarantee of consistency but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology. Finally, it should be noted that ontologies are specified in the form of definitions. Traditionally Description Logics [9] are used to describing the terms definitions of the ontology.
5 Using Application Ontologies to the DSS Domain

In order to understand many of the outlined problems and the proposed solutions, we will use an example based on the study of a DSS. In this section, we show an example of how these mechanisms can be used to make a semantic view.

The proposed example tries to model a DSS to automate the operations and decisions accomplished during a risk operation in a financial institution. Because we are interested in some aspects of the problem, which is considered later, we will simplify the specification.

We need to take into account the fact that in all financial institutions, there are two groups of operations: those involving liabilities (investment funds, fixed term deposits, current accounts, thrift accounts, values, etc.) which constitute a deposit for the institution, and those involving assets (loans, credits, warranties, discounts, etc.) which imply a risk for the institution. The DSS will focus on the last group.

An operation of this type can be requested by individuals as well as by legal entities, whether they are clients of such an institution or not. The financial institution calculates the Leverage Coefficient by using information given by the user and information contained in some external repositories such as ASNEF databases (banking databases of possible non-fulfillment and their current situation) and the RAI database (a database which refers to unpaid bank bills, bills of exchange, etc.) and data included in the IS when the customer is a Client.

As a result of this coefficient and information provided by the customer, the institution approves, refuses or interrupts the operation and communicates the result to the person who requested it.

Figure 2 shows a partial view of the ontology. The knowledge domain is more complex (for example not all the terms used by the Debit operation are considered)
but the ontology which we have made only considers some of the important, relevant concepts and roles used in the DSS domain.

In this figure, you can see the concepts which are only used by the IS (Financial-I, Branch, Current_account, Debit-Transaction, etc.), other group of concepts and roles such as Expedient or No_client which are only used by the DSS, and another group which is shared between the DSS and IS. There are others concepts (Anything and Person) which have only been included for the semantic view design.

In this proposed example, IS is formalised through the Relational Approach. Figure 3 shows the mapping between the concept ‘Legal-entity’ and its equivalent table. The interface programs are responsible of connecting concepts in the ontology with the relational tables (external model).

<table>
<thead>
<tr>
<th>Table Model</th>
<th>Attribute-name</th>
<th>Null?</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIF</td>
<td></td>
<td>N</td>
<td>Id</td>
</tr>
<tr>
<td>Sector</td>
<td></td>
<td>S</td>
<td>Sector</td>
</tr>
<tr>
<td>Incorporate-date</td>
<td></td>
<td>S</td>
<td>Date</td>
</tr>
<tr>
<td>CNAE</td>
<td></td>
<td>N</td>
<td>CNAE</td>
</tr>
</tbody>
</table>

SQL Code

```
CREATE TABLE Legal-entity
    ( CIF char(10) not null,
      sector char(20)
      incorporate-date date
      CNAE char(2)
    PRIMARY KEY (CIF) ;
      Sector
```

Fig. 3. Mapping between the concept ‘Legal-Entity’ and the equivalent table

6 Limitation of the Description Logic in the Complex Domain

Description Logics [9] are Artificial Intelligent formalisms which allow domain knowledge to be represented by focusing on classes of objects and their relationships and by offering inferences on the class structure.

However, previous works [8] have shown that the Description Logics that are equipped with sound, complete, and terminating procedures still suffer from several limitations that are not acceptable when representing complex domains similar to those we have proposed above. Here is a list of the most important limitations:

- The interpretation domain is flat, in the sense that logics consider the world to be made up of elementary objects (grouped in concepts) and the binary relation between them.
- One consequence of the previous property is that n-ary relationships are not supported.
In our example there are certain kinds of properties which cannot be represented by simply modeling the n-ary relation in terms of n-binary relations (for example it might be desirable to assert that Account linked to Expedient by the respective role is Entail-Account.)

- General inclusion axioms are not usually supported. Although inclusion axioms are essential when we want to assert properties of classes and relations as required in the complex domain or when we want to study the possible evolution of semantic views (for example you do not include an axiom of a-cyclic relationships or incompatible relationships).

Although these limitations are important, they are partially solved by special DLs as proposed by Nebel [16], [17], Baader [1] and by De Giacomo and Lenzerini [8]. But these DLs do not include evolution mechanisms. This is a very important problem because these mechanisms are needed when we want to study the possible evolution of semantic views. We should not overlook the fact that both the Information System and the Decision Support System are dynamic systems (i.e. they change through time), active systems (they carry out processes of change) and that they are open systems (changes in the environment produce changes in the system), and for all these reasons change mechanisms are necessary.

7 The Evolution Problem in the Integration Framework

The logical schema of information obtained in the previous semantic view must be enriched in order to allow us to manage those intrinsic evolutionary aspects, and this characterizes the modeling process of both kinds of systems.

In order to provide concrete change mechanisms, we will study the evolution of the IS and the DSS when the IS is formalized using the Relational Approach. This model is very easy to use because of the mathematical rigor in the definition of data representations, operators and the simplicity of data structures. In addition, the technological advance in relational databases facilitates its use in information systems.

The DSS is formalized using the Object-Oriented approach because notions of identity, classification, polymorphism and inheritance promote an interesting way of organizing the objects and their activity.

The O-O approach and some advanced O-O mechanisms (multiple inherence, multiple membership and dynamic classification) introduce changes in the semantic views which improve the intrinsic evolutionary nature of the IS and DSS.

Figure 4 shows the class model of the previously proposed DSS using the Object-Oriented Methodology UML. Examples such as multiple inherence (terms Open-credit) or dynamic classification (being a Client or Non-client depends on the Trade-relation) have an implicit semantic which can be considered as terms and roles in a specialized ontology based on the object model.
7.1 A Specialized Ontology Based on the Object Model

Although Description Logics allow changes to be made in Semantic Views, these changes have to be made by hand by changing the models and re-structuring the system. This is one reason for introducing implicit dynamic mechanisms (such as classification, reusability and maintainability) that may be considered as an extension of the O-O approach.

Classification

The ontologies, described using DL, also allow implicit classification and sub-classification by means of the concepts and the roles. The instances are facts of type 'is a' and the classification relationships can be expressed by means of definition.

In the O-O approach, classification is a core concept. The objects which share data structure and behavior are grouped into classes, and the classes can be sub-classified by means of inheritance mechanisms. This implies a high degree of abstraction, which describes important properties and ignores irrelevant ones, and this is a conceptual process which is independent of the programming languages. This
process is usual in human knowledge acquisition and allows abstract concepts to be expressed.

The classification schema in object-orientation is always explicit and implies a description of the static structure of the classes of the system and their relationships (see figure 4).

**Reusability**

Two important features of object-orientation entail a better reusability than in DLs: inheritance and information hiding.

- Inheritance has important benefits when developing Ontologies:
  - Code reusability: the code of the behavior of a class is reused by its subclasses thereby increasing maintainability and reliability.
  - Code sharing: different users and systems can share the same classes.
  - Interface consistency: inheritance guarantees that the inherited behavior is the same for the subclasses and that the objects of the subclasses interact in a very similar way.
  - Rapid prototyping: the classes developed by previous systems can be reused and refined.

- Information hiding means that when concepts and roles are reused, it is only necessary to understand their nature and interface.

**Maintainability**

Some of the previous properties increase maintainability in Ontologies based on the object model.

- Explicit classification facilitates the introduction of new classes and the re-structuring of previous ones.
- Inheritance allows the reuse of previous concepts.
  
  Information hiding allows the code of the behavior of a class to be changed without changes being made in the uses of the class.

### 7.2 Facing the Evolution

The IS, DSS and Semantic Views are transformed over time because their structures are transformed, and as a result these systems evolve.

In this context of evolution, some of the following changes will be considered:

- Changes in the data structure of the IS: structural changes in the IS such as creation, elimination or modification of tables can entail changes in mapping rules. However, other changes such as creation, elimination or modification class instances or tuples in the tables do not imply changes in the structure of the ontology.

- Changes in the data structure of the DSS: in the same way, operations such as creation, elimination or modification of classes or links can change concepts or roles in the ontology and the generated data which might need to be included in the IS. Sometimes, this fact may produce a change in the structure of the IS, when there are no tables where the information can be included.

- Changes in the terms or in the relationships between terms in the semantic views: in this case it is necessary to take into account that the integrity of the Semantic View is
guaranteed, and to consider what changes must be propagated to the Decision Support System or to the Information System.

Consequently, the widest and most integrated vision of the evolution in these systems is needed as a result of the propagation of changes in the IS, DSS or Semantic Views to the others.

7.3 Graphical Approach

As we mentioned above, certain advantages in the specification of the semantic views evolution can be obtained when an O-O Approach enriched with advanced object-oriented techniques, such as dynamic classification, multiple inheritance, etc., is used. These techniques improve the evolutionary nature of semantics views.

Moreover, the previous example of the UML Class Model (figure 4) can be equivalent to a heterogeneous graph (see figure 5) where the nodes are different types of classes (ellipses in the graph) and arcs are different types of relationships between them (associations, specialization, generalization, etc.) or relationships defined by users (have, link, associate-to-...).

Fig. 5. Heterogeneous graph equivalent to the UML class model example for the DSS
We are now researching practical ways of specifying these graphs. In previous papers [10], [11] we presented a set of basic operations on the graph (such as Create_node, Del_node, Create-arc, Del_arc, Nodes-connected-to, Connection-by, etc.) and a set of basic restrictions on the graph (tree, acyclic_graph, weak_connected, etc.), restrictions on the nodes (unify_name, etc.) and restrictions on the arcs (have a label, acyclical, reflexive, anti-symmetrical, transitive, incompatible_with and so on).

These operations allow the structure of the graph to be changed, and the set of restrictions on the graph, nodes and arcs help to propagate the changes.

The relation defined by the user restrictions and semantics must be explicitly defined. In the graph, not only is a special semantics adopted to O-O relationships (generalization, specialization, etc.) but there are also some implicit restrictions for these relationships. For instance, Figure 6 shows that the relation ‘Kind of’ always verifies some restrictions such as acyclical restriction, anti-symmetrical property, and it is incompatible with Part_of or with is-a.

![Fig. 6. Restrictions about kind-of relation](image)

Changes in the structure of the IS, DSS and Semantic View would result in change operations in the structures and restrictions of the graph. This fact helps the propagation of changes to be represented and automated. For instance, Figure 7 shows how the IS structure evolves as a result of introducing a new dynamic classification on the Account. This causes a change propagation in the mapping rules and also a change in the Link relation used by the DSS. All of these changes result in change operations in the graph. Before allowing the change, the preconditions of each operation must be checked.

We think that a graphical approach has certain advantages. From the point of view of formalization, it is easier for propagation changes to check the preconditions of operations and restrictions on the graph than the propagation change in the concepts and roles through the DL. It is also possible for abstract data types or classes to be used for implementation (for example a C++ data structure). Therefore, from a user’s point of view, this approach is easier to understand because it encodes the shared information by means of a simple graph with the labeled concepts and relations.

8 Conclusions

Ontologies are an explicit partial specification of a conceptualization for the purpose of modular design, redesign and the reuse of knowledge.
The use of semantic views based on Ontologies allows, on one hand, the flexible encapsulation of data in repositories and, on the other hand, the sharing of information between the IS and the DSS.

A translation of the schemas used by ISs and DSSs to a common semantic schema is needed in order to improve information sharing.

Evolution mechanisms of both types of systems and semantic views are needed in order to be able to take advantage of the dynamic, active and open characteristics of these systems.

As a result of several limitations found in Description Logics when they are used to describe ontologies for complex domains, we have proposed that ontologies be described using an object-oriented approach and therefore using graphical representation and graph restriction.

Once again it is important to note that our approach provides mechanisms to propagate changes, and that it focuses on the evolutionary nature of the Information System, the Decision Support System and Semantic Views.

References


A Proposal for Project Management Using CommonKADS

Isabel María del Aguila¹, Samuel Túnez¹, Joaquín Cañadas¹, Alfonso Bosch¹, and Roque Marín²

¹Dept. of Languages and Computation. University of Almería. Campus Universitario. La Cañada (Almería) 04120-E, SPAIN {imaguila, stunez, jjcanada, abosch}@ual.es
²Dept. of Information and Communications Engineering. University of Murcia. Campus de Espinardo. Espinardo (Murcia) 30100-E, SPAIN roque@dif.um.es

Abstract. This work describes a proposal for software lifecycle and project management when developing knowledge based systems using CommonKADS methodology. To build this proposal, an analysis of the similarities between software engineering and knowledge engineering was made. As a result, the anchor point milestones and the invariant definitions from the spiral model lifecycle for software development have been translated to CommonKADS methodology. This proposal is being applied to a project in order to develop a phytosanitary advisor system used by farmers in greenhouse crops.

1 Introduction

Among the different categories of information systems, knowledge-based systems (KBS) are the most complex software systems [1]. KBS are characterized by their high risk, loose definition and the need of applying specific techniques of knowledge engineering. Usually, KBS development projects are hard to control, because they solve poorly structured problems, with subjective requirements, and often their input is incomplete, uncertain or fuzzy. However, this kind of system is still a software system. Then, it must be developed using a software engineering method and to obtain results at a predetermined cost and time and with the desired quality level.

Today the idea that the modeling approach is a constant in current software development methodologies is widely accepted; and also that the knowledge level is most appropriate for modeling KBS, following the computation level taxonomy proposed by Newell [7], not the symbol level where classical software systems are modeled.

There are many modeling frameworks; the most representative are CommonKADS, MIKE y PROTÉGÉ-II. Each one of them has its own modeling approach. CommonKADS is prominent for having defined the structure of the Knowledge Model (or Expertise Model), MIKE puts emphasis on formal and executable specification of the expertise model, as the result of the knowledge acquisition phase, and PROTÉGÉ exploits the notion of ontology.
The different knowledge engineering approaches have their own lifecycle and project management methods, for KBS development. In this last method, it exists a closer matching between software and knowledge engineering. The first attempts were based on the waterfall model, but early on, due to the experimental nature of these systems, many of the development methodologies evolved into a kind of incremental model. Knowledge modeling is a cyclic process. New observations lead us to refine or complete the model. This model is only an approximation to the reality and is typically faulty. That is, a continuous feedback is needed in the modeling process. The result of each stage is enhanced in following stages; evolutionary prototyping is a good example [6]. More functional executable versions of the system are delivered, and the software delivered is in turn used as a prototype to be evaluated in order to develop the next version.

Incremental models make the project management process difficult, because resource planning and intermediate control of results cannot be done in advance. In software engineering context, the Boehm spiral model [3] was introduced to reduce this type of management problem in generic incremental models. Certainly, many of the KBS development methodologies most applied, such as CommonKADS and MIKE [11], use a variant of this model, but the application of the lifecycle for management tasks is not clearly specified.

This work presents a proposal for management tasks in a KBS development project using the CommonKADS methodology, which represents an alternative to the original lifecycle approach of this methodology. This paper will be structured as follows. Section 2 will summarize the knowledge engineering evolution, from techniques applied to lifecycle and management, defining the existing parallelism between knowledge engineering and software engineering. Section 3 describes CommonKADS methodology and its lifecycle. Once the methodology is described, Section 4 presents the proposal for project management, showing how to implement the concepts of spiral Boehm’s model. Finally, Section 5 describes an example of application in a project for the development of an advisor system for farmers in intensive cultivations in southeast Spain.

2 Survey of Knowledge Engineering

Knowledge engineering is a working field that appears along the evolution of artificial intelligence. In a similar process, with a certain time advantage, and without interactions, software engineering has been developed. The last can be defined as: the application of tools, methods and techniques to the production and maintenance of automated real-world problems, which are economically reliable and work efficiently on real machines.

From the perspective of developing of automated systems that solve problems (software systems), and depending on the structure, kind of decision, requirements, context and certainty of the target problem, software systems can be classified as Transaction processing systems, Support systems in decision making and Knowledge based systems [1]. The later are the scope of knowledge engineering. Then, knowledge engineering can be defined as the set of principles, methods and tools to be applied in order to build an maintain automated KBS, which solve poorly
structured problems, with subjective requirements, and often their input is incomplete, uncertain or fuzzy. In addition, there are not classic algorithmic solutions for them.

However, in the upcoming knowledge society, this is a very limited perspective. If software is the target of software engineering, the target of knowledge engineering must be knowledge, and not only the development of software that uses knowledge. From this point of view, knowledge engineering must be considered as a key technology in Knowledge Management.

In order to know the actual state of knowledge engineering, it is interesting to review its evolution from its origin to the present day.

2.1 Historical Review

In its origin, research in Artificial Intelligence was focused on the development of general methods to solve problems, as GPS or STRIPS. During seventies, the efforts were concentrated on the development of small KBS. These systems used general inference engines and the specific knowledge of an application domain. Then it was necessary to acquire this specific knowledge and transferring it to the KBS.

In early eighties, the development of KBS was seen as a transfer process of human knowledge into an implemented knowledge base. This kind of software system was applied to many different domains.

The next evolution step was exporting this technology to industrial use, but this transfer process was failed in many cases. This is the same situation appeared for traditional software systems in the so called software crisis in seventies. This crisis was solved by means of software engineering and software developing methodologies. In a similar way, in KBS development, the crisis originated knowledge engineering and KBS development methodologies.

In nineties, these knowledge engineering methodologies have been established, appearing new methodologies known as second generation methodologies. Like in the evolution of traditional software system development methodologies, these methodologies focus on the use of models as essential pieces. In addition, they define a lifecycle and apply model reusing. Among the multiple existing methodologies, CommonKADS, MIKE and PROTÉGÉ II can be highlighted as the most interesting ones.

The future of knowledge engineering in this decade is directed towards the application of this engineering in Knowledge Management and the integration and sharing of knowledge and information via WWW [12].

2.2 Relationships between Software Engineering and Knowledge Engineering

Both engineering approaches have experienced a similar evolution, with a difference of a decade, but along parallel paths. Despite its similitude, they have also great differences. Then, some authors consider them as two completely different branches of engineering, while others point to similarities in their development [8].

Any engineering development (both in knowledge engineering and software engineering) requires the definition and standardization of a lifecycle, which goes from requirement definition to maintenance of finished product. Both engineering
approaches have evolved through different lifecycle models, which indicate the order of completion of the different development phases. In knowledge engineering methodologies, this lifecycle evolution encompassed Code and Fix, (used to develop first KBS) Waterfall, (KADS, KLIC) Prototyping (Waterman) and Incremental models (CommonKADS, MIKE).

There are many overlapping areas in knowledge engineering and software engineering. Knowledge engineering has provided methods that have been taken up by software engineering, making the system built thus more powerful and robust. On the other hand, software engineering has supplied knowledge engineering with techniques to make its products more useful by increasing KBS robustness and reliability. For instance, knowledge engineering should learn about system production, reuse and maintenance and project management from software engineering, and knowledge engineering should teach software engineering to replace passive tools by interactive aids and build flexible behavior into system and interfaces.

However, there is a bottom difference between knowledge engineering and software engineering. For software engineering, the highest level is the symbol level, while knowledge engineering needs to model and start from the knowledge level, as stated by Newell [7], separating the knowledge from its representation.

3 CommonKADS Methodology

CommonKADS methodology is a standard in Europe and covers all KBS development processes, from the specific aspects of management to the design, based on the direct modeling of expertise provided by the expert.

In CommonKADS, we can see three ideas emerged not only from knowledge engineering experience, but also from general software engineering: modeling, reusing and risk management.

3.1 Modeling Approach

Nowadays there exists a consensus about that the process of building a KBS may be seen as a modeling activity. To build a KBS means building a computer model with the aim of achieving problem-solving capabilities comparable with those of a domain expert. This modeling approach is present on both knowledge engineering and software engineering, and it is reflected in the methodologies developed for both engineering.

The main product obtained from CommonKADS is a model suite, which is organized on three levels. The first level, called the context level, describes the context where the system will work. The second one, called the concept level, represents the nature of the system and models the knowledge and communication methods needed by it. The last level, the system level, is a description of the software solution and its implementation features [10].

In the case of traditional software development methodologies, models are used firstly to describe the problem proposed by the user and to be solved by the software system. Secondly, models are used to describe what the software system that solves
these problems will be like. These two kinds of models can be called conceptual model and computational model. They are reflected in the concept of level from CommonKADS, more specifically in knowledge model and system level.

In software engineering, the most commonly used conceptual models are relatively simple. In knowledge engineering this models are fairly rich and complex. The methodologies usually divide conceptual model into three different representation levels: strategy models, reasoning models and domain models. The CommonKADS context level represents these strategic models, and the others are included into the knowledge model. This separation between domain knowledge and reasoning knowledge enhances reusing at two levels: generic task libraries and ontologies.

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>MODEL</th>
<th>PROJECT DELIVERABLE</th>
<th>DEVELOPMENT STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>Organization model</td>
<td>Scope and feasibility study</td>
<td>Identify problem/opportunity areas and potential solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Decide about economics, technical and project feasibility</td>
</tr>
<tr>
<td></td>
<td>Task model</td>
<td>Impacts and improvement study for</td>
<td>Gathering interrelationships between the tasks, agents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the selected target solution</td>
<td>involved, and uses of knowledge.</td>
</tr>
<tr>
<td></td>
<td>Agent model</td>
<td></td>
<td>Deciding about organizational measures and task changes.</td>
</tr>
<tr>
<td>Knowledge</td>
<td>Knowledge model</td>
<td>Knowledge model report</td>
<td>Knowledge identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Knowledge specification</td>
</tr>
<tr>
<td></td>
<td>Communication model</td>
<td>Design report</td>
<td>Knowledge refinement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Identify the core information objects and list of transactions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Build the dialogue diagram</td>
</tr>
<tr>
<td></td>
<td>Design model</td>
<td>Software deliverables</td>
<td>Design System Architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System documentation</td>
<td>Identify target implementation platform</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test report (V &amp; V)</td>
<td>Specify architectural components</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
<td>System documentation</td>
<td>Specify application within architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>System and user evaluation</td>
</tr>
</tbody>
</table>

**Fig. 1.** Relationship among CommonKADS level, project deliverables and development steps

CommonKADS methodology explains the software development process output in depth, that is, the model suite and steps followed to obtain it. Figure 1 shows the relationship among models, development steps and project deliverables. However, the mapping of the steps from model building to management and control actions is not specifically described, keeping such actions in the background. That is, the lifecycle must indicate the order for completing the different development phases, but it is not clearly specified. Furthermore, knowledge engineering in general has never taken an interest in fully defining all the activities to be performed when building a KBS.
3.2 Lifecycle and Project Management

Planning, monitoring and control of a development project is required to commercialize a KBS, that is, a technically impeccable development is not enough to guarantee the success.

The CommonKADS lifecycle proposes to follow two parallel processes: Development and Management. The first one builds and validates the models, while the second introduces the spiral approach, defining four activity areas; a) review of the status of the project; b) risk assessment, where potential risks are identified and valued, and control measures to solve them are outlined; c) planning the activities during cycle, making resource allocation; and, d) monitoring, where tasks are controlled and results are evaluated. These activities recur in every cycle of the project.

Project management concentrates first on products and outputs to be delivered, rather than on activities or phases. Therefore, the CommonKADS models are the key artifacts to be completed. CommonKADS introduces the concept of state of a model to define its completion degree. There are five states: empty (none), identified (information is there), described (draft available), validated (draft has gone through evaluation cycle) and completed (signed-off).

The main problem of this lifecycle approach is the separation between management and development tasks. It is uncertain when the two processes meet, that is, when risk assessment or planning are necessary during development or validation of the models. By contrast, the original Boehm model defines from four to six activity areas or quadrants, where at least one of these represents the development of the key artifacts or products, and the others are management and control activities.

4 Spiral Development in CommonKADS

This section contains a lifecycle proposal that attempts to clarify the steps to be carried out in CommonKADS methodology, applying the flexibility of the spiral incremental model.

4.1 Brief Description of Spiral Model

We start from the classical spiral model, which has four quadrants: communication with users, planning, risk assessment and development. The spiral development model is a risk-driven process model generator. It is used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system’s degree of definition and implementation. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

Boehm proposes three anchor point milestones for the typical software development: lifecycle objectives (LCO), review to ensure that at least one architecture choice is viable from organizational perspective, lifecycle architecture
(LCA), review of the artifacts and initial operational capability (IOC) is the stakeholder’s commitment to support operations. These milestones produce a spiral model with at least four cycles: new concept development and objectives, new product development, product enhancement and product maintenance, as described in [9]. This cycles invariably display a set of characteristics that must be followed to succeed [4]: cyclic and concurrent engineering, risk driven determination of process and product, growing a system via risk-driven experimentation and elaboration, and, use anchor point milestones for ensuring stakeholder commitment.

4.2 Proposal of Lifecycle

Firstly, the invariant characteristics of Boehm model are described, and how to adapt them to the KBS development.

Concurrent determination of key artifacts. Artifacts are CommonKADS models; the models are developed incrementally. The versions of the models are enhanced in following cycles. This concurrent development avoids predetermining too early certain model elements that are not clear or are subject to modifications. Generally, when developing a KBS it is difficult to define the requirements to be built in the models at the first development steps [2]. Then, the obtaining of the models has to be carried out in a concurrent way, because the decisions made when elaborating a model can affect to other models.

Each cycle does objectives, constraints, alternative risks, review and commitment to proceed. In each spiral cycle, we must to considerate all activity areas in order to avoid commitment to stakeholder-unacceptable and wasted effort in elaborating unsatisfactory alternatives. This invariant reflects the need of defining objectives in each cycle, that is, to determine which will be the CommonKADS model or models to be developed, to search alternatives depending on the risk areas, revising the stakeholder needs, and, at last, to reach a commitment with him for each proposed model.

Level of effort and degree of detail driven by risk considerations. Risk analysis determines “how much is enough” to each activity and to develop each CommonKADS model. KBS are the kind of software that involves more risks. To reduce the risks, it is necessary that these risks drive the development process, making the effort level and the development detail of each model being determined by risk. If one of the severe risks detected is that the domain knowledge level of system users can be very different, the effort needed to obtain the communication model will be stronger than the expected one. It may need risk control measures, like the elaboration and valuation of the user interfaces prototypes with the users.

Use anchor point milestones, to avoid development paralysis. A major difficulty in original spiral development was its lack of intermediate milestones to serve as commitment points and progress checkpoints. Boehm [4] describes three anchor point milestones for the typical software development to manage stakeholder life-cycle commitments as LCO, LCA, and IOC. We are redefining these anchor point
milestones for CommonKADS, one for each level of models, CD (context definition), OD (ontology development), and SD (software development), obtaining differences in the starting point axis and the number of basic cycles defined for the spiral (Figure 2).

![Figure 2. Lifecycle for project management using CommonKADS](image)

Secondly, it is described how to use the concepts of activity areas, starting points and milestones. The proposed lifecycle is based on the use of a spiral model guided by risk and some milestones related with KBS development process, that present a closer matching with a level of the CommonKADS model suite.

The defined activity areas are: risks assessment, planning, development and stakeholder interaction (communication and evaluation). The first and the second are the typical management tasks. Planning includes building a detailed plan and scheduling tasks and resources. Risk assessment is the basic task, and it guides the overall process. CommonKADS authors offer a set of potential risks that can be used as a checklist [5]. The risks that can affect to the development of the target model of the cycle are solved by applying control measures, that usually are associated with setting alternatives and using prototypes to achieve the stakeholder commitment.

The classical spiral model defines four cycles: new concept development, new product development, product enhancement and maintenance. With CommonKADS, a development with at least five cycles has been proposed, with its corresponding five starting points. However, the number of cycles can change in different cases. Depending on the risk associated to building the models, it is possible that the cycle for the concept level (Knowledge Model, Communication Model) splits on two or more cycles. For instance, when the knowledge to be modeled is complex, or if the knowledge to be used for solving the problem is unstable. It is not recommended to merge cycles in order to execute less than five, because of the complexity and instability of the requirements associated with a KBS.

The starting point axis is used when modeling any software engineering project. The clearest case is maintenance. When the project goal is to carry out a maintenance
adjustment of an existing application, the first stages will be contacting with costumer, risk analysis, planning, development and evaluation of the modifications that have been done. This need also appears in knowledge engineering. It is usual that when having a running software application based on the same CommonKADS knowledge model (defined, validated and completed), it appears the need of building new applications, starting the project on the new product development starting point.

We have proposed a mapping between starting points of Boehm’s spiral model and the model suite of CommonKADS. The new concept development starting point, branches out into two points, which map the context and concept levels in the CommonKADS model hierarchy. The next two starting points are the same as the original ones: new product development and product enhancement. In CommonKADS, the artifact level is a single level, but its authors distinguish between design and implementation. Therefore, two starting points are proposed. Finally, there is a last cycle of the spiral for maintenance.

![Mapping between Boehm’s lifecycle for software development and CommonKADS lifecycle](image)

Boehm’s anchor point milestones are used as commitment points and progress checkpoints. CommonKADS has three important milestones: when describing the context where the application will work (CD), when modeling the interactions and the knowledge needed to solve the problem (OD), and when the software system is ready to be delivered (SD). These milestones are closely related with model suite and milestones LCO, LCA and IOC.

By unrolling the spiral, as shown in Figure 3, the mapping between the original Boehm's model and its adaptation to CommonKADS methodology becomes clearer.

5 Practical Application

This lifecycle proposal is being put into practice in the development of the FEDER project “Development of an intelligent system for decision support in agriculture in southeast Spain”. The main objective of this project is building an advisor program for farmers, to help them when making decisions about the phytosanitary state of
greenhouse crops. Next, there is a description indicating the cycles that have been developed and what the forecasts for project termination are. The evolution of the state of CommonKADS models is also described, and it is shown in Figure 4.

Cycle 0 corresponds to the first modeling level, which is aimed at analyzing the main organization characteristics, in order to discover where and why the KBS may play an important role. The first project milestone (CD) has produced as a result the discovery of the existence of two major working lines. The first one corresponds to the phytosanitary information management system, and the second to the system for assisting in phytosanitary control decisions. Furthermore, the overall structure of organization tasks and the input, output, and performance of each of them are obtained. In Cycle 0 of project management, the first CommonKADS context level models where the system is located have been handled. The organization model (OM) has been completed, and the task and agents models (TM and AM) are also described.

Phytosanitary information management may be considered as a problem of traditional data management, and therefore, it has been addressed using the waterfall software development lifecycle, without applying CommonKADS.

The most important activity in Cycle 1 has been the construction of the knowledge model where domain knowledge, inferences and tasks are specified. In the risk assessment of this cycle (analysis of the first versions of organization and task models), one of the decisive results for the rest of development is the confirmation that experts clearly differentiate two major tasks in the process of advising the farmer. In the first place, a decision is made in respect to whether it is necessary or not to use chemical control on a crop, and if so, a later decision is made about the product to be used. This has caused the setup of two subsystems within the phytosanitary control model, which are approached separately: a) The analysis of the need to act on the crop, and b) the decision on the type of action.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>OM</th>
<th>TM</th>
<th>AM</th>
<th>KM – a</th>
<th>CM</th>
<th>KM – b</th>
<th>DM – a</th>
<th>DM – b</th>
<th>Implem-a</th>
<th>Implem-b</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed</td>
<td>Defined</td>
<td>Defined</td>
<td>Identified</td>
<td>Identified</td>
<td>Identified</td>
<td>Identified</td>
<td>Identified</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Completed</td>
</tr>
<tr>
<td>1</td>
<td>Completed</td>
<td>Completed</td>
<td>Defined</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
</tr>
<tr>
<td>2</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
<td>Milestone</td>
</tr>
<tr>
<td>4</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
</tr>
<tr>
<td>5</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
</tr>
<tr>
<td>6</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
<td>Completed</td>
</tr>
</tbody>
</table>

Fig. 4. Cycles and States of CommonKADS models

At present, a large part of subsystem a has been developed, and subsystem b has been started. During Cycle 2, we have developed finished versions of the knowledge and communication models, for target problems: tomato crop affected by greenhouse whitefly and grapes affected by thrips. Specifically, the construction and development of subsystem a has been completed during two cycles, 3 and 4. Along Cycle 3 KM-a has been finished, and DM-a has been built and validated. For risk assessment, it has been necessary to design operational prototypes for a target problem, which have enabled to start the implementation process. Now the prototype is operational and
validated, and the modules corresponding to other detrimental agents in tomatoes and grapes to be considered are being developed.

The actual cycle is Cycle 4. Requirement elicitation of subsystem b, that is, the corresponding knowledge model, is being completed. The estimates are to require at least one cycle for developing this subsystem and a last cycle for integration. Nevertheless, the final iteration number for completing the project will depend mainly on the risk analysis for each case. Figure 4 shows the model evolution in each cycle.

6 Conclusions

Knowledge engineering helps to build better knowledge based systems that are easier to use, have a well-structured architecture and are simpler to maintain. This is achieved by means of methodologies. These methodologies have evolved from code and fix to second generation methodologies that cover all software development process. However, management methods are less developed than in other engineering approaches, like the methodologies characteristic of software engineering.

We propose a lifecycle model for KBS development that clarifies the link between knowledge engineering and software engineering, enhancing the definition of the CommonKADS methodology. It makes easier the use of this KBS methodology, defining a minimum number of cycles based on its model suite. The stakeholder lifecycle commitments are managed with at least three anchor point milestones: CD, OD and SD.

This proposal applies the existing parallelism between knowledge engineering and software engineering, translating the most recent developments from spiral model to software development with CommonKADS methodology. This is a general trend. For instance, the modeling notation uses UML for describing the knowledge model.

This proposal has been applied in the development of a KBS building project for plague and disease control in agriculture, funded by EC (1FD97-0255-C03-03).

References


A General Planning Method for Allocation of Human Resource Groups

Manfred Mauerkirchner

Polytechnic University of Upper Austria, A-4272 Hagenberg, Austria
manfred.mauerkirchner@fhs-hagenberg.ac.at
http://www.fhs-hagenberg.ac.at

Abstract. Planning (creating optimized plans) is the key feature of every management system. The behavior of human resources, especially the domain of description of cooperation in groups is not easy to model. Generally, this kind of planning is a classic NP problem and, together with the need of variable project structures and project flows, it leads to a very complex computational task.

To perform allocations of qualified human resources within a dynamic environment we propose a specific non analytical multiobjective function. Our first target is to get optimal resource groups at any time and in such a way that global project time is minimal and that constraints of costs are satisfied. To meet these requirements we use a specific three(3) tier optimizer architecture with different methodologies: fuzzy sets, numeric non gradient methods and evolutionary computation methods.

1 Introduction

Generally, all planning methods are based on formalisms like abstract networks, which have to model both structure and flow of activities at least. The way in which such a network is configured significantly affects the work required and the accuracy of results that are obtained when the network is analyzed. Three(3) well known characteristics in this specific field are worth mentioning in particular: size, precedence and parallelism. However, since their inception two(2) main categories of networks can be distinguished:

1. non probabilistic, deterministic networks like CPM, MPM
2. probabilistic activity networks like PERT, GAN

ad 1. CPM (Critical Path Method) is the oldest representative of a so-called AOE-network, where activities are defined by edges. Contrary to it, in MPM (Metra Potential Method), a so-called AOV-network, all activities are situated at vertices while edges describe predecessor-successor relationships between activities.

ad 2. PERT (Program Evaluation Review Technique), developed in the late 1950’s, is a so-called EOV-network with events at vertices and additional probabilistic values to integrate the possibility of risk management. GANs (Generalized Activity Networks) extend this mechanism by introducing probabilistic
branching between activities, an aspect where PERT networks are still deterministic.

Current thinking within the field of general project planning not only identifies the need to manage the objectives of time and cost, but also the aspects of organization and quality, too. To achieve realistic results in this comprehensive area a current allocation of qualified and efficient resources (planning of resources simultaneously) is necessary. Over the years various techniques have been developed that attempt to solve, more and more efficiently, this classic NP problem, known as Resource Constrained Project Scheduling Problem (RCPSP).

Because of the very high degree of complexity, exact solution methods (based on the Branch-and-Bound principle) are of minor practical relevance. What is preferred is the implementation of heuristic methods that generate solutions iteratively by checking so-called priority rules (serial priority rules versus parallel priority rules). To improve these results, more than one priority rule can be taken into consideration simultaneously and, in addition, be weighted by adaptive random sampling methods.

Another approach to solve the RCPSP is the accomplishment of searching strategies based on methodologies of evolutionary computation. We propose a problem specific adapted genetic algorithm as base mechanism (extended by the means of fuzzy logic and numeric non gradient methods) to solve the RCPSP completed by the possibility to plan human resource groups automatically.

2 Basic Model

To be successful in implementing a general project planning methodology first the appropriate modelling of logical project flow, project structure and, in addition, of skills and behavior of human resources are necessary.

To describe the logical dependencies of such a flexible system all project structures are represented by directed non circular graphs with so-called project subtasks \((i, j, k)\) in fig.1) as nodes, where each subtask is a comprehensive container of an arbitrary sequence of so-called project activities \((i_1, i_2, j_1, k_1, k_2, k_3)\) in fig.1), and where, in addition, each project activity is mapped into an unambiguous project activity type. Activities are the elementary labor units of our project model and characterized by the assignability of an unique type of human skill and by an externally estimated and generated time \((t^m_p)\), where \(m\) is a project subtask and \(p\) is a project activity). That time defines how long an average skilled person would need to perform the task of executing a specific project activity. Mapping the logical project flow into a work breakdown structure, one has to observe that at one extreme a project activity can be started only in the case of successful termination of its predecessor activity (concerning the same subtask), and at the other that the first activity of a new subtask will only be able to start, if all predecessor subtasks of the graph have finished successfully. The described situation is depicted in fig.1.

Human resources play a very important role in modern development projects nowadays and the effective modelling of their behavior is one of the most chal-
lenging tasks. Each resource of an existing resource pool \((r, s \text{ in fig.1})\) is characterized by a discrete set of resource steps (Step \(r_1\), Step \(r_2\) and Step \(s_1\) in fig.1) to enable the specification of variable costs and skills of each person. A unique amount belongs to each step of a person specifying individual costs per labor hour (Cost \(r_1\), Cost \(r_2\) and Cost \(s_1\) in fig.1). Because experience, routines and knowledge of single persons are parts of one’s qualification profile we introduce some specific additional values (so-called resource capacities, \(CapT_{i_1 - j_1}\) and \(CapT_{i_2 - k_3}\) in fig.1) to define the personnel skills for performing the existing project specific activities (to play project type specific roles)\(^7\). Once more the situation is illustrated in fig.1.

There is no doubt that the constellation of teams play a crucial role in the economic field and that groups have to be composed of people with according abilities to fulfil the group’s task (project activity)\(^5\). We propose a specialized database for storage of relationships concerning cooperation between all human resources to achieve calculable results in creating optimal efficient resource groups for each single project activity. We use a so-called groupworking matrix \(A = [a_{i,j} \mid i, j \in \mathbb{N}, 1 \leq i, j \leq |R|, 0 \leq a_{i,j} \leq 1]\), where \(R\) is the set of human resources and each element \(a_{i,j}\) a cumulated assessment of person \(j\) by person \(i\). Assuming \((a_{i,i} = 0 \mid 1 \leq i \leq |R|)\) implies that values \(a_{i,j} \approx 0\) describe perfect harmony while greater values express growing refusal (see next matrix as example with three(3) resources).

\[
\begin{array}{ccc}
R_1 & R_2 & R_3 \\
R_1 & 0 & a_{12} & a_{13} \\
R_2 & a_{21} & 0 & a_{23} \\
R_3 & a_{31} & a_{32} & 0 \\
\end{array}
\]

Fig. 1. Project and Resource Structure
In addition, to obtain a measurement concerning suitability for cooperation between persons a function named $\text{group}(R^*)$, where $R^* \subseteq R$ has to be defined which uses the elements of database $A$ in a specific manner\cite{7}.

3 Planning Methodology

Based on the previous model a planning process can be represented by a three-tier architecture using three different methodologies:

The so-called \textit{strategic optimizer} (StratOpt) using fuzzy sets, the so-called \textit{global optimizer} (GlobOpt) using numeric non gradient methods and the so-called \textit{local optimizer} (LocOpt) using evolutionary computation methods, where the StratOpt is the top level tier and LocOpt is situated at the lowest level.

3.1 The Local Optimizer (LocOpt)

\textbf{Definition of Multiobjective-Function} We have to regard three criteria simultaneously:

1. \textbf{Criterion(time):} $time^m_p(R^*) = T^m_p(R^*) + \Delta T^m_p(R^*)$
   \hspace{1cm} with $T^m_p(R^*) = \sum_{r \in R^*} \kappa^r_p(gr) \cdot \text{comm}(|R^*|)$, where the function $\text{comm}(n) = a^{\frac{n(n-1)}{2}}, \ a > 1$ calculates the communication overhead (which depends upon the number of group members $n$), $t^m_p$ are the externally estimated project activity execution times, $\kappa^r_p$ are resource capacities (dependent directly on resource costs $g_r$ and for that reason also dependent on resource steps),
   \hspace{1cm} with $\Delta T^m_p(R^*) = \text{Max}\{\Delta T_r \mid r \in R_{\text{busy}} \cap R^*\}$, where $\Delta T_r$ are remaining times of such resources, which are still involved in executing other project activities (a feature which is used by the GlobOpt).
   \hspace{1cm} Remark: $T^m_p(R^*)$ is the calculated time, which is necessary to perform the project activity $p$ of project subtask $m$ through resource group $R^*$. $\Delta T^m_p(R^*)$ is the calculated \textit{waiting time}, which appears only in the case of allocation of so-called busy resources. In addition, both times get influenced by the selected resource steps.

2. \textbf{Criterion(cost):} $cost^m_p(R^*) = T^m_p(R^*) \sum_{r \in R^*} g_r$
   \hspace{1cm} with the following constraints: $cost^m_p(R^*) = G^m_p \leq (G^m_p)_{\text{max}}$, where $(G^m_p)_{\text{max}}$ is a project activity specific cost limit, which will be used by the numeric non gradient method in the GlobOpt.
   \hspace{1cm} Remark: $cost^m_p(R^*)$ are the calculated total costs of each project activity. Optional fixed salaries of resources (costs during times of inactivity) are not taken into account. Once more, costs per man hour $g_r$ depend on selected resource steps.

3. \textbf{Criterion(groupworking):}

\begin{equation*}
\text{group}^m_p(R^*) = \sum_{r \in R^*} \sum_{r' \in R^* \land (r' > r)} \text{Max}(a_{r,p'}, a_{r',r})
\end{equation*}
Remark: $\text{group}_p^m(R^*)$ defines a possible measurement to assess the suitability of resource group $R^*$.

**Standardization of Multiobjective-Function $Q_p^m$** It is the main task of the LocOpt to find resource sets $R^*$ which minimize the previous defined functions $\text{time}_p^m(R^*), \text{cost}_p^m(R^*)$ and $\text{group}_p^m(R^*)$ simultaneously (all of them delivering values $\geq 0$). Using the principle of dualism a task of searching minimum of a function $f(x) \in R^+$ can be transformed into another task of searching maximum of function $g(x) \in [0, 1]$, where $g(x) = \frac{f_{\text{max}} - f(x)}{f_{\text{max}} - f_{\text{min}}}$. Using this mechanism in our case brings us to standardized functions $\text{time}_p^m(R^*), \text{cost}_p^m(R^*)$ and $\text{group}_p^m(R^*)$.

Defining additionally a set $\{\lambda_i \in [0, 3], i \in \{1, ..., 3\} \mid \sum_{i=1}^{3} \lambda_i = 3\}$, where $\lambda_i$ are so-called strategic factors (initialized and modified by the StratOpt) lead to a prefinal version of our multobjective function

$$Q_p^m(R^*) = \lambda_1 \text{time}_p^m(R^*) + \lambda_2 \text{cost}_p^m(R^*) + \lambda_3 \text{group}_p^m(R^*)$$

Till now we defined our multobjective function with no regard of cost limitation $\text{cost}_p^m(R^*) = G_p^m \leq (G_p^m)^{\text{max}}$. One practicable solution would be the introduction of a specific penalty function

$$\text{pen}(R^*) = \log_2(1 + \frac{(\text{Max}\{\text{cost}_p^m(R^*) - (G_p^m)^{\text{max}}, 0\}) \times 5}{(G_p^m)^{\text{max}}})$$

Now we have reached the final version of our multiobjective function

$$Q_p^m(R^*) = Q_p^m(R^*) - \text{pen}(R^*)$$

**Optimization Process** We propose a problem specific evolutionary computation method to be successful in optimizing our multiobjective function $Q_p^m(R^*)$, a problem adapted classic genetic algorithm [4]. Without restriction of generality the following assumptions are made:

- each resource is characterized by exactly four(4) resource steps, named 0, 1, 2, 3
- each project subtask consists of a constant sequence of project activities $(p_1, p_2, p_3, p_4)$
- $\forall r \in R \land \forall p \in \{p_1, p_2, p_3, p_4\}$ both resource status and resource step can be coded binary as followed:

<table>
<thead>
<tr>
<th>ResStatus</th>
<th>ResStep</th>
<th>BinCode</th>
<th>Cost $g_r^x$</th>
<th>Capacity $(\kappa_p^r)^i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>inactive</td>
<td>0</td>
<td>0 0 0 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>inactive</td>
<td>1</td>
<td>0 0 1 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>inactive</td>
<td>2</td>
<td>0 1 0 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>inactive</td>
<td>3</td>
<td>0 1 1 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>active</td>
<td>0</td>
<td>100</td>
<td>$g_r(0)$</td>
<td>$\kappa_p^r(g_r(0))$</td>
</tr>
<tr>
<td>active</td>
<td>1</td>
<td>101</td>
<td>$g_r(1)$</td>
<td>$\kappa_p^r(g_r(1))$</td>
</tr>
<tr>
<td>active</td>
<td>2</td>
<td>110</td>
<td>$g_r(2)$</td>
<td>$\kappa_p^r(g_r(2))$</td>
</tr>
<tr>
<td>active</td>
<td>3</td>
<td>111</td>
<td>$g_r(3)$</td>
<td>$\kappa_p^r(g_r(3))$</td>
</tr>
</tbody>
</table>

A so-called chromosome (binary string) could now be constructed randomly like shown in the next scheme (an example with four(4) resources)
A General Planning Method for Allocation of Human Resource Groups

<table>
<thead>
<tr>
<th>Resource</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinCode</td>
<td>101</td>
<td>011</td>
<td>111</td>
<td>110</td>
</tr>
<tr>
<td>Status</td>
<td>active</td>
<td>inactive</td>
<td>active</td>
<td>active</td>
</tr>
</tbody>
</table>

Step 1-3

Remark: Only such resources are authorized, where specific skills are available ($R^I = R \setminus \{ r \mid \kappa^r_p \equiv 0 \}$) and resource status is suitable ($R^{II} = R^I \setminus \{ r \mid (\text{Status} = \text{zero}) \lor (\text{Status} = \text{res}) \}$). Therefore the generation of $R^{II}$ is both project activity type specific and full dynamic (dependent on resource states). Each randomly created binary string represents a specific resource set with arbitrary resource steps (resources with inactivity bit are omitted).

Our proposed framework (classic genetic algorithm based optimization process) consists of the next few steps:

- definition of a set of chromosomes, length $l = 3|R^{II}|$, size of population (number of rows) =3$l$
- random initialization of population matrix $S^0 = [s^0_{i,j}]$ with binary values ($l$ columns, 3$l$ rows)
- calculation of function value $Q^m_p$ for every chromosome
- so-called reproduction of $S^0$ (new initialization because of relative frequency)
- random determination of pairs of chromosomes
- random determination of so-called crossover position within chromosome pairs
- execution of crossover operation [4] (changing of substrings)
- mutation of single bits in $[s^0_{i,j}]$ with constant probability
- new calculation of function value $Q^m_p$ for every chromosome
- maximum of all 3$l$ function values is the so-called fitness of the resource group
- check of specific break condition

Remark: The latter steps should be repeated as often until a break condition is reached (problem specific). To improve the performance of the optimization process the so-called simplex operator [1] (instead of crossover operation), other penalty functions, another size of population, a variation of mutation probability, etc. could be taken into consideration.

Every solution delivers a group of resources and furthermore

- the specific resource step of each group member
- the execution time for the requested project activity
- the waiting time for the requested project activity (optionally)
- the costs for the requested activity
- a prereservation of resources (optionally)

### 3.2 The Global Optimizer (GlobOpt)

At any time of the project realization the planning process should perform allocations of qualified human resources such that
1. the global time $T$ of the project realization is minimal, i.e.

$$T_0 = \min T = \varphi(time^m_p(R^*) | m \in M, p \in \rho(m)), \forall R^* \subseteq R$$

2. or the global costs $G$ of the project realization are minimal, i.e.

$$G_0 = \min G = \psi(cost^m_p(R^*) | m \in M, p \in \rho(m)) = \sum_{m \in M, p \in \rho(m)} G^m_p, \forall R^* \subseteq R$$

In both cases the constraints of costs of each project activity satisfy

$$cost^m_p(R^*) = G^m_p = T^m_p(R^*) \sum_{r \in R^*} g_r \leq (G^m_p)^{max}$$

Because it is probably impossible to define the functions $\varphi$ and $\psi$ in the analytical way we use the simulation to obtain its values. To build the simulation model we use our previously explained basic model and the so-called DEVS (discrete event system specification) formalism, which is a system theory concept to model event oriented systems \[11\]. To be able to approach to optimal values of our global objective functions $\varphi$ and $\psi$ we propose the application of the following iterative procedure called Rosenbrock’s method\[10\]. Permanent step-by-step changing of the $n$ elements of a vector in a specific way lead to a global minimum of an arbitrary function.

The execution time of each project activity is substantially influenced through the limit of costs $(G^m_p)^{max}$. What we need first are initial values of the vector $G^{max} = ((G^1_p)^{max}, (G^1_p)^{max}, ..., (G^m_p)^{max}, ...)$. The definition $(G^m_p)^{max} = f.t^m_p$ where $f$ is an average activity type specific salary per man hour would bring us efficient values. With

$$\sum_{m \in M} \sum_{p \in \rho(m)} (G^m_p)^{max} = G_0$$

and, in addition

$$\frac{(G^m_p)^{max}}{G_0}, i \in \{1, ..., \sum_{m \in M} |\rho(m)|\}$$

we have all requirements to start our global optimization process. Each elementary variation of $G^{max}$ implies a calculation of $\varphi$ (or $\psi$) and therefore a simulation of the whole project graph (with an explicit call for the LocOpt by each individual project activity). The very flexible construction of our base model enables us to regard all varying circumstances efficiently\[5\].

### 3.3 The Strategic Optimizer (StratOpt)

Till now we are able to control the situation of varying the former defined ”limit of cost” vector $G^{max}$ in such a manner that either the global project duration $T_0$ or the total project costs $G_0$ approximate to their minimum in an iterative way. What remains to be mastered is to integrate the influence of our three(3) (hitherto) constant factors $\lambda_i$ to realize strategic decisions.

Remember that \{ $\lambda_i \in [0, 3] , i \in \{1, ..., 3\} | \sum_{i=1}^3 \lambda_i = 3$\}, where

- $\lambda_1$ is the factor to weight the influence of time,
- $\lambda_2$ is the factor to weight the influence of costs,
- $\lambda_3$ is the factor to weight the influence of groupworking.
It should be clear that specific assumptions like $\lambda_1 = 3$, $\lambda_2 = \lambda_3 = 0$, $\varphi$ is the selected objective function, would optimize the overall project time with no regard of costs and cooperation of persons (a situation which seems to be the most usual nowadays and known as ”time to market” strategy). Our major interest concentrates on the transition of verbal strategic decisions like ”time and costs are less important than groupworking” and their integration into the optimizing process. Any of such requirements lead straightforward to a conceptualizing with means like fuzzy logic\textsuperscript{3}.

Next we consider the following vague elementary strategic statements:

1. time ($\lambda_1$) is most important
2. costs ($\lambda_2$) are most important
3. groupworking ($\lambda_3$) is most important
4. time ($\lambda_1$) is unimportant
5. costs ($\lambda_2$) are unimportant
6. groupworking ($\lambda_3$) is unimportant
7. all parameters are important simultaneously

All parameters have a common domain, the interval $[0, 3]$. The individual fuzzy sets are characterized through the next definitions\textsuperscript{3}.

ad 1. Membership function is a $\Gamma$-function with

$$\Gamma(\lambda_1) = \begin{cases} 0, & \text{if } \lambda_1 < 2 \\ (\lambda_1 - 2), & \text{if } 2 \leq \lambda_1 \leq 3 \end{cases}, \lambda_2 = \lambda_3 = (3 - \lambda_1)/2, \varphi \text{ function}$$

ad 2. Membership function is a $\Gamma$-function with

$$\Gamma(\lambda_2) = \begin{cases} 0, & \text{if } \lambda_2 < 2 \\ (\lambda_2 - 2), & \text{if } 2 \leq \lambda_2 \leq 3 \end{cases}, \lambda_1 = \lambda_3 = (3 - \lambda_2)/2, \psi \text{ function}$$

ad 3. Membership function is a $\Gamma$-function with

$$\Gamma(\lambda_3) = \begin{cases} 0, & \text{if } \lambda_3 < 2 \\ (\lambda_3 - 2), & \text{if } 2 \leq \lambda_3 \leq 3 \end{cases}, \lambda_1 = \lambda_2 = (3 - \lambda_3)/2, \varphi \text{ or } \psi \text{ function}$$

ad 4. Membership function is a $\Lambda$-function with

$$\Lambda(\lambda_2) = \begin{cases} (\lambda_2 - 1)/0.5 \text{ if } 1 \leq \lambda_2 \leq 1.5 \\ (2 - \lambda_2)/0.5 \text{ if } 1.5 \leq \lambda_2 \leq 2 \\ 0, & \text{if } \lambda_2 > 2 \end{cases}, \lambda_1 = 0, \lambda_3 = 3 - \lambda_2, \psi \text{ function}$$

ad 5. Membership function is a $\Lambda$-function with

$$\Lambda(\lambda_3) = \begin{cases} (\lambda_3 - 1)/0.5 \text{ if } 1 \leq \lambda_3 \leq 1.5 \\ (2 - \lambda_3)/0.5 \text{ if } 1.5 \leq \lambda_3 \leq 2 \\ 0, & \text{if } \lambda_3 > 2 \end{cases}, \lambda_2 = 0, \lambda_1 = 3 - \lambda_3, \varphi \text{ function}$$

ad 6. Membership function is a $\Lambda$-function with

$$\Lambda(\lambda_1) = \begin{cases} (\lambda_1 - 1)/0.5 \text{ if } 1 \leq \lambda_1 \leq 1.5 \\ (2 - \lambda_1)/0.5 \text{ if } 1.5 \leq \lambda_1 \leq 2 \\ 0, & \text{if } \lambda_1 > 2 \end{cases}, \lambda_3 = 0, \lambda_2 = 3 - \lambda_1, \varphi \text{ or } \psi$$
ad 7. Membership function is a \( \Lambda \)-function with

\[
\Lambda(\lambda_1) = \begin{cases} 
0, & \text{if } \lambda_1 < 0.5 \\
(\lambda_1 - 0.5)/0.5 & \text{if } 0.5 \leq \lambda_1 \leq 1 \\
(1.5 - \lambda_1)/0.5 & \text{if } 1 \leq \lambda_2 \leq 1.5 \\
0, & \text{if } \lambda_1 > 1.5 
\end{cases}, \quad \lambda_2 = \lambda_3 = (3 - \lambda_1)/2, \ \varphi \ or \ \psi
\]

4 Conclusion

Our proposed methodology is flexible, effective and efficient enough to meet the following requirements[9]:

- calculation of project flow and allocation of resource groups
- integration of plans that have been executed partly
- change of personal qualification profile and groupworking characteristics of resource pool
- change of logical project flow and project structure

Most of these features are implemented as prototype in a multitasking environment, with C++ as programming language and MFC (Microsoft Foundation Classes) as object class library. Our general planning method is used by an on-line and off-line runable planning subsystem, which has to create optimized project plans whenever needed and, in addition, by a project control subsystem that has to integrate real-time decisions.

References


Optimizing Organizational Structure: A Hausdorff Benchmark for Complex Social Systems

Markus Schwaninger

Institute of Management, University of St. Gallen,
Dufourstrasse 48, CH-9000 St. Gallen, Switzerland
Markus.Schwaninger@unisg.ch

Abstract. The search of optimal structures for social organizations has been an ongoing concern of management science, but reliable answers have not been given. Optimization theory has provided solutions, but these have always addressed very specific organizational problems, not the issue of overall optimality in terms of the fitness of the organization as a whole. In this paper, a new approach, grounded in system theory is suggested for assessing and ultimately also for pursuing such a general optimality of organizational designs. It is proposed that the optimal fractal dimensionality of organizations is equivalent to that of living organisms. This hypothesis is submitted to and corroborated by a first test. Herewith, a new path to a theory-based design of optimal organizations has been opened up, the implications of which for the methodology of organization design may be substantial.

1 The Question and Traditional Ways of Dealing with It

The question “Are there optimal structures for organizations?” has recurrently been posed in this general form. The answers have varied, but to date the tenor has been: “No, there is no generally optimal structure. Probably there is an optimal structure for each organization. But we are still looking for a solid theory to ascertain that.”

In the past, the question of organizational optimality has been considered discussible only in very specific and therefore limited contexts. The more famous examples are linked with the optimization of organizational processes, e.g. routing problems, resource allocation problems, and control problems. Applications of this type have resulted in substantial improvements as far as the economic use of scarce resources is concerned. The types of problems of enduring structures, which have been studied in terms of optimization, are different:

- Single-objective optimization: Under this title the classical types of optimization can be subsumed. A typical case is the question of the optimal span of control in a hierarchy of agents with largely uniform tasks (e.g., the optimal number of salespersons in service centers for a market).
- Multi-objective, multi-parameter and multi-level optimization: Large and complex logistic systems may call for “multiple-issue” solutions which allow for an overall optimum, taking into account different objectives at the same time (e.g., a
postal system where cost, time and ecological performance may be the pre-
eminent criteria, and where a structure of distributive centers with several levels
may be considered). Solutions for this type of problem may involve multi-
objective, multi-parameter and multi-level optimization in a combined mode.

It must be noted that only a subset of the optimization techniques can provide “opt-
timal solutions” in the strict sense of the word. Given the “curse of dimensionality”
most of the more complex solutions mentioned are numerical and only approximate
optimality. In other words, it is known that these solutions are close, often very close
to the theoretical optimum, although that theoretical optimum remains unknown. The
exact boundary between the cases with “optimum” solutions and those with “close to
optimum” solutions is given by the theoretical bound of feasible computation, which
establishes the physical limits of analysis. Essentially, the use of heuristic methods is
motivated by the question as to whether the decision problems which are part of these
optimization problems are “NP-complete”. This means, to put it in a nutshell, that an
exact solution of a given optimization problem is only possible if one can construct a
polynomial algorithm which can solve all the decision problems of the complexity
class in focus [cf. 10, 28].

Another approach to the search for optimality has been multidimensional structur-
ing. In principle, optimization can be applied to multi-dimensional problems, as has
been said: Multi-objective optimization is a case in point. When we talk about “multi-
dimensional organization” we mean something different: We indicate the simultane-
ous structuring of an organization according to different structuring criteria. Even
though multidimensionality has contributed substantially to absorbing complexity, the
question of an optimal degree of dimensionality remains open. In fact, it has barely
been addressed.

To deal with this very issue, I have taken an approach, which is very much in the
tradition of system theory and cybernetics: The search for an isomorphism, in other
words, an invariant structural feature, which is relevant in different contexts. Such
isomorphisms allow mapping systems from different domains of reality onto one and
the same model. For this purpose I have reverted to the natural sciences.

Systems theorists and cyberneticians have traditionally leveraged knowledge origi-
nating from biology for the domain of social systems (e.g., Miller, Pask and Beer).
Also, several of the founding fathers of Systems Theory were trained biologists: The
Society for General Systems Research was founded in 1954 by Ludwig von Bertalanffy, Ralph Gerard (both biologists), Anatol Rapoport (a mathematician, who had
applied mathematics to biology) and Kenneth Boulding (an economist). Rashevsky
and Rapoport spearheaded the application of mathematical modeling to both biologi-
cal, and social relationships. Finally, General Systems Theory has also identified
structural similarities which extend beyond mere comparisons of the social and bio-
logical domains. These also include analogous situations, e.g. in chemistry and phys-
ics, where such analogies have also been formalized mathematically, in the sense that
practically the same mathematical model may be employed to express theories widely
disparate in content [cf. 18, 19].

Many of the loans from biology taken by social scientists have been at the level of
analogy, - stimulating thought, but not leading all the way to rigorously formulated
theory. There have at least been two notorious exceptions:
1.) *James Grier Miller’s Living Systems Theory (LST)*: Miller identified 19 subsystems which make up a living system, and which are invariant across a wide spectrum of organized wholes, from the living cell to a society [17]. LST originated in the biological studies of Miller, who by training is a medical doctor.

2.) *Stafford Beer’s Viable Systems Model (VSM)*: Beer discerned a set of control functions, and their interrelationships, which are the necessary and sufficient preconditions for the viability of any human or social system. This model is based on an isomorphic mapping originating from studies of the structure of the human central nervous system. A crucial feature of the VSM is that it introduces the notion of recursiveness, in that the viability of social systems is grounded in the recursive existence and functioning of the system of (self) control it specifies [4, 5, 6].

Both of these models are outstanding in their originality—They have triggered numerous studies and applications in the realms of organization, society, and engineering. They both address the preconditions for viability, but not the issue of the optimality of structures. 

1 Idea for an Innovative Approach and Hypothesis

If we ask “What dimensionality is optimal?” current organization theory itself cannot provide us with satisfactory answers. Again, we have to take recourse to the natural sciences, and once more biology appears to be a reliable source of knowledge.

From biology we know that living organisms (plants, animals, humans) are structured in a fractal mode: Their metabolism, breathing, blood circulation and other vital functions are optimized by these fractal structures. Several research teams, in Germany, one of them at the University of Giessen, headed by Manfred Sernetz, have studied these structures. They have shown why and how this “strategy of fractal organization” entails the living organisms’ functional superiority as compared with manmade bio-reactors.

These bio-reactors require continuous stirring to bring about the turbulence of the liquid necessary for higher efficiency of the chemical reaction, which can only be induced by high-energy input. The organism of an animal or human produces catalytic processes similar to those of a bio-reactor. However, there is no need for stirring; what the pump-function of the heart induces is just a laminar blood flow, enabled by a much lower energy level. Even so, the mixture of the liquid phase (blood) and the stationary phase (tissue) are achieved in an optimal way, with a result that a continuously stirred tank reactor (CSTR) could only mimic at enormously high cost. This stupendous superiority of the natural “reactor” is largely due to the fractal structure of the living organism.

A fractal structure is one where the principle of organization is applied in an iterative mode. Consequently the parts show the same structure as the whole; the organization is self-similar.

Fractal structure is a special case of recursive structure. However, the term “fractal” adverts to a specific notion: It derives from the Latin participle “fractus” - broken,

---

1 A comparative evaluation of these two models is yet to be accomplished.
which in this case denotes that the dimensionality of the object under study is usually
to be expressed by a broken number, not by 1, 2 or 3. A dimensionality of one would
be applicable to a line; two corresponds to an area and three to a sphere. Certain as-
pects of self-similarity cannot be described adequately by means of the classical
measures, length, surface and volume.

For example, the curve of Koch’s snow flake, a classical fractal, has a finite sur-
face but an infinite length. Also, the mathematical idealization of a tree of blood ves-
sels with infinitely thin branches has a volume of zero although it reaches every single
point inside the organ it feeds [25].

My idea was the following: If the fractal dimensionality of living organisms is op-
timal (as proposed by biology; [cf. 25]), and if optimal structures are invariant across
different domains of reality (as claimed by systems theory; [cf. 17, 4, 5, 6]), then the
benchmark for optimal structures should be the size of that dimensionality.

Friedrich Hausdorff, in his landmark article on dimension and external measure
[11] defined a measure for the size of objects independent of the resolution of the
measurement, ‘e’. The Hausdorff dimension is a very general measure for the dimen-
sionality of objects, applicable to any shape.

The dimension of a fractal can be ascertained by determining its conventional
measures with the help of increasingly fine yardsticks, and by ascertaining how much
the measures grow as a function of this refinement. Real biological objects can be
measured in this way, whereby their fractal dimension can be determined, with the
help of different methods (e.g. Mass-radius-analysis). Biometric studies have ascer-
tained that the metabolic activity of living organisms follows a law of power, ex-
pressed by the following formula, which has been derived from empirical evidence
[after 24, 26, 14]:

\[ M = L^D, \text{ where } 2.2 > D > 2.3. \] (1)

M is the metabolic activity, measured as the organism’s throughput in Joule per
second, and L is the length of the organism. D is the fractal dimensionality of the
organism.

On the basis of measures of multiple species, Sernetz and his team ascertained an
allometric exponent of b=0.74, measuring the progression of throughput in Joule per
second, as a function of body volume in litres. Expressed by the length instead of the
volume, the applicable exponent is D=3b=2.22, which denotes the growth of the
metabolic rate as a function of the length of an organism. On the assumption that the
extant living organisms are optimally structured, Sernetz concludes that an optimally
built organism must be 2.22-dimensional [25]. In other words, according to this the-
ory, for optimally built organisms the law of power must be:

\[ M = L^{2.22}. \] (2)

If we continue taking the functioning of living organisms as a role-model for social
organizations, we may be on the threshold to a further development of organizational
theory.

We know that social organizations, in terms of communication and information,
show properties, which are in a certain sense identical to those of living organisms.

---

2 The dimensionality is usually, albeit not necessarily fractal.
Even though there are fundamental differences - social organizations are constituted by autonomous agents with their own goals, preferences and values [1] - they resemble living organisms to a high degree. If we take the internal transfer of knowledge as a case in point, we can suppose that this is, in principle, the most important mechanism of adaptation. This can at least be derived from studies made during the last decade [cf. 12 and literature quoted therein].

If the structural laws governing the behavior of social organizations and living organisms are the same, and there is a large body of evidence indicating that they are, then we can make use of a powerful isomorphy (i.e., structural invariance):

Similarly to an optimally built organism, an optimally conceived organization should exhibit a dimensionality of about 2.2 to 2.3.

Then, the hypothesis to be tested is:

*An optimally conceived organization shows a dimensionality between 2.2 and 2.3.*

This law should hold independent of size, sector or activity, or other situational factors (context). Small exceptions are conceivable: for example, in the case of an enduringly constant environment, a lower dimensionality would be thinkable as optimal, but probably only in economic terms. Note, however, that optimization in one dimension only, is in principle problematic in complex systems: Whenever one variable in a complex system is maximized, the likelihood increases that bottlenecks will spread and that consequently non-stable or chaotic behavior will ensue [2].

### 3 Test of Hypothesis

The hypothesis to be tested is that:

*An optimally conceived organization shows a dimensionality between 2.2 and 2.3.*

There are several ways of testing this hypothesis. The usual one would be to proceed with a comparison, scrutinizing a number of organizations in similar situations, measuring a) their performance and b) their organizational dimensionality, and subsequently examining whether the structure-performance link implied by this hypothesis is corroborated by the data.

A second approach would be to test whether a structural arrangement already proven or at least considered optimal is in accordance with the hypothesis.

Even though the first test would be the stronger option, initially the second one is to be carried out, as it involves a lower cost.

### 3.1 Setup

The setup will provide for examining a theory for an optimal structuring of interactions and communications in large groups, - the *Team Syntegrity* model (*TSM*). This is a structural framework to foster cohesion and synergy in larger groups of individuals, or to encourage the transformation of mere aggregates of individuals with similar interests into organizations with their own identities. Invented by Stafford Beer [7], *TSM* is a progressive design for democratic management in the sense of the heterar-
chical-participative type of organization [cf. 20, 21]. The TSM is a holographic model for organizing processes of communication in a non-hierarchical fashion that can be shown to be mathematically optimal for the (self-) management of social systems. Based on the structure of polyhedra, it is especially suitable for realizing team-oriented structures, and for supporting processes of planning, knowledge generation and innovation in turbulent environments. In the following, I shall illustrate the architecture of the model by using the geometry of an icosahedron, which is one of the structures commonly used to organize syntegration events, - in this case with 30 participants.

The formation of networks by persons who are connected by mutual interests is a manifestation of the information/knowledge society and a structural answer to challenges of our time. An infoset is a set of individuals who share a common concern and who are in possession of pertinent information or knowledge connected with the issues of concern, and who have the will (and most likely also the enthusiasm) to tackle these. The Team Syntegrity model supplies the structural framework for the synergetic interaction of an infoset which is intended to lead to an integration of multiple topics and perspectives towards a shared body of knowledge. The term Syntegrity results from a combination of synergy and tensile integrity. We speak of synergy when the interaction or co-operation of two or more agents produces a combined effect greater than the sum of their individual efforts. Tensile integrity is the structural strength provided by tension, as opposed to compression [9].

### 3.2 Structural Architecture of Team Syntegrity

An infoset of 30 persons, for example, can organize itself according to the structure of an icosahedron, the most complex of the regular, convex polyhedra (Fig. 3), whereas for smaller gatherings, structures based on other polyhedra are possible. Each member of the a 30 member infoset is represented by one edge on the icosahedron. Each vertex stands for a team of five players (→ five edges) working on one topic; in an icosahedron there are 12 vertices that would be marked by different colors in a Syntegration. Therefore, given the geometry, each participant as a player/actor is connected by his/her edge to two different teams. Ms. Red-Yellow, for instance, belongs to the teams (vertices) Red and Yellow. At the same time, she acts as a critic to two other teams (for example, Black and Silver, which are her immediate neighbors). This means that each team consists of 5 players and 5 critics. Altogether, the thirty agents perform a total of 120 roles (30 times 2 roles as a player and 2 as a critic). In addition, there is the observer role, which will be explained in a moment.

This structure resolves the paradox of peripherality versus centrality of actors in an organization (as formalized by Bavelas [3]): While peripherality leads to communication pathologies, alienation and low morale, centrality is needed for effective action. However, as a group grows, centrality can only be „purchased“ at the cost of increasing peripherality [16]. Team Syntegrity enables an Infoset to acquire „centrality“ via a reverberative process (each team will meet more than once), although the peripheral-ity of each one of its members equals zero, i.e., there is no peripherality at all.

---

3 Kindly made available by TSI – Team Syntegrity Inc., Toronto, Canada.
Fig. 1. Icosahedral structure of the Team Syntegrity model

Typically, the structure of Team Syntegrity is applied in the context of processes by which ill-defined issues have to be tackled and for which contributions of multiple agents with different backgrounds are required. This is the case e.g. in strategy making, knowledge generation and organizational development. A Syntegration process has the following phases: After the phases of initialization and joint design of the agenda around the common subject of interest, the 12 individual teams (consisting of 5 players and 5 critics each) explore their respective topic. Each team meets several (usually three) times and writes up a summary of its results to share with the whole infoset. Discussions evolve as follows: The sessions are developed in a parallel mode with two teams working at a time. This means that 20 of the 30 members of the infoset are involved in these discussions. The remaining 10 can attend any one of the meetings as observers, in order to complement the views derived from their activities as players and critics in their respective, individual set of 4 teams. They may also use some of that time for lateral conversations with other “idle” members or simply relax.
The fact that the same issue with its different but interconnected aspects is continually processed by the same set of people, who gather in alternating compositions (topic-oriented teams) implies strong reverberation and leads to a self-organizing process with high levels of knowledge integration. There is no center required to integrate the individual efforts; integration just occurs of its own accord. It can be shown mathematically that this is a geometrically ergodic process, in which the eigenvalue of the process converges to a minimum: Ninety percent of the information in the system will be shared after three iterations, and ninety-six after four iterations [13].

3.3 Calculus of the Dimensionality of the Team Syntegrity Architecture

In the following, a rough calculus of the dimensionality of an infoset as structured by the icosahedral architecture of Team Syntegrity ensues:

\[ R_I = R_T + R_C, \]  \hspace{1cm} (3)

where \( R_I \) denotes the total set of actual relationships of the members of the infoset. Its components are \( R_T \) - the relationships at the team level, and \( R_C \) - the complementary relationships of the observers.

\( R_T \) is the composite of the relationships within the teams \( (i) \), 12 in the ideal case. \( n_i \) expresses the number of team members of team \( i \), the ideal number of members being five players \( (p) \) plus five critics \( (c) \) for all teams. The number of relationships between a pair of members is denoted by \( (m) \).

\[ R_T = \sum_{i=1}^{t} \frac{m \cdot n_i(n_i - 1)}{2}, \quad n_i = p_i + c_i. \]  \hspace{1cm} (4)

For this ideal case, with reciprocal relationships between each pair of members, i.e. \( (m=2) \), \( R_T \) amounts to:

\[ R_T = 12 \times 2 \times 10 \times 9 \times 1/2 = 1'080 \]

\( R_C \) is the composite of the relationships between the members of the discussing teams \( (b) \) and the members of the team they observe. The arrangement is that, while team discussions are going on, some of the observers relax whereas others switch from team to team, visiting both sessions going on at the time. Switches can also be made between the iterations of the team discussions, i.e., an observer could distribute his activities between the three iterations: For example, in the three iterations of the parallel discussions of teams A and B he or she could observe team A in the first iteration, relax in the second, and observe team B in the third iteration. To take account of these aspects, some assumptions must be made explicit to arrive at a first, rough calculation. We establish a parameter \( f \) denoting the average percentage of the

---

4 The formula to calculate the eigenvalue is: \( y = \left( \frac{1}{\sqrt{5}} \right)^n \), with \( n \) denoting the number of iterations.

5 Such relaxation is essential to keep the vigor, concentration and involvement of participants high.
total number of observers $b$ (ideally 10) which are actively observing teams during a given pair of sessions. Furthermore, we introduce a parameter $s$ which expresses the average fraction of those active observers who switch between teams. Based on the many Syntegration events realized to date, including those accompanied by the author, the assumption of $f=1/2$ and $s=1/3$ appears to be realistic for a rough approximation of an idealized Syntegration.

Consequently, the following formula can be applied:

$$R_C = \sum_{i=1}^{t} b \cdot n \cdot f \cdot (1 + s).$$

(5)

For this ideal-type we get:

$$R_C = 12 \times 10 \times 10 \times 1/2(1+1/3) = 800$$

Adding up $R_T$ and $R_C$ leads to:

$$R_I = 1'080 + 800 = 1'880$$

In other words, the set of relationships of the icosahedral infoset, as specified above, totals 1’880. At this point, the dimensionality of the structure ($x$) can be calculated as a function of $R_I$ and the total number of the members of the infoset ($N$):

$$R_I = N^x.$$ 

(6)

To solve this equation the following transformations are necessary:

$$\log R_I = x \log N,$$

$$x = \frac{\log R_I}{\log N}.$$ 

(7)

With a total set of relationship ($R_I$) of 1’880 and a total number of infoset members ($N$) of 30, the result is:

---

6 To date approximately 150 Syntegrations have been realized, despite the relative recency of the model. The author has directed or co-directed several, among them the first worldwide electronic Syntegration [cf. 8] and accompanied many more, within the framework of a research association with Stafford Beer, the creator of the model, and TSI-Team Syntegrity Inc., Toronto, the organization which makes Team Syntegrity available to organizations.

7 The assumptions made explicit here try to capture a structure which enables an „optimal“ flow of information, taking into account the psycho-physically limited resilience of participants. Variations of the parameters $f$ and $s$ as a function of the situation at hand should also be considered (see below).

8 Equation (6) is isomorphic with equation (3) in section 2. Therefore, $N$ is formally identical with the $L$, and $R_I$ with the $M$, in the latter. In other words, an isomorphic correspondence between Length and Number of members of an Infoset, as well as between Metabolic Rate and Number of Actual Relation- ships between Infoset Members is assumed.
In other words, the dimensionality of the icosahedral architecture of Team Syntegrity is $2,21658$. In sum, the working hypothesis formulated above is strongly corroborated.

The surprising fact is that this size of $x$ is very close to the optimal dimensionality observed in biological organisms. It is actually closer to 2,22 than originally expected (cf. hypothesis above).

3.4 The Revised Theorem

In the light of these results, the hypothesis formulated above can be slightly revised, in the sense of proposing the following Theorem for an Optimal Structure of Organizations:

An optimal organization structure shows a dimensionality of approximately 2,22.

3.5 Discussion

I am aware that this proposition is bold, but it conforms to Popper’s principle of falsifiability. In principle, this Theorem for an Optimal Structure of Organizations provides a powerful conceptual instrument to establish whether the dimensionality of a structure is too high or too low. The benefit lies in avoiding potentially huge costs and a host of disfunctionalities, - not only economic but also social and ecological ones.

However, the theorem also prompts questions. One major question that emerges is, how general this theorem is. Does not contingency theory postulate that organizational structures are and should be a function of the contexts they face? According to contingency theory, placid environments require and induce less complex structures than turbulent ones [cf. 15, 27]. The answer is straightforward: The theorem proposed here defines optimality in terms of contexts similar to those faced by living biological organisms. These are always confronted with complex, turbulent environments, at least potentially. Also, in the social domain potential high-level complexity and turbulence are ubiquitous.

Team Syntegrity, the reference model used for the test above, is definitely a model to be recommended for dealing with complex issues, but it would not be advisable for the structuring of a mere routine task. In addition, coping with that kind of task would most probably not require an organization of a dimensionality of 2,22. However, routine tasks are usually part of more encompassing organizations, which in the end strive for viability and development [22, 23]. As a whole, these organizations are in principle exposed to high complexity, at least potentially.

Further research should explore the possible limits of this theorem. Admittedly, a limitation of this paper is one of extension: Therefore, not all the practical implications, which are already discernible at this point, can be treated in detail.

For example, this first test has been confined to one organizational model, albeit under consideration of multiple modalities of its use. Other models for organizational
structuring, which cannot be examined here, should be studied in the light of this
Theorem for an Optimal Structure of Organizations. Also, the test applied here, has
essentially been realized in a deductive mode. In addition, empirical tests of the type
mentioned at the beginning of Section 3 should be carried out in the future.

Furthermore, variants of the assumptions underlying formula (5) of the calculus
should be considered. For example, possible trade-offs between parameters \( f \) and \( s \)
should be studied; see also the sensitivity analysis in the following section. Finally, a
great deal could be gained by improving and fine-tuning organizational models and
methodologies, - Team Syntegrity being one of them - , in the light of this theorem.

4 Sensitivity Analysis

Following up on the discussions of the calibration of parameters, a number of sce-
narios were calculated in order to test the sensitivity of the TSM structure’s dimen-
sonality to changes in these parameters. A summary is presented in Table 1.

|--------------------|-------------------------------|-------------|-------------|------------------|-----------|-----------------------------|---------------|-----------------|----------------|-------------------------------|----------------------------|ality | Differen-
|                    |                               | m           | p           | c                | n         | i                           | b             | f               | s             | RC                             |                           |ce from ideal | Deviation % |
| Base               | 2                             | 5           | 5           | 10               | 12        | 1080                        | 10            | 0.500           | 0.333         | 800              | 1880                         | -0.00342 | -0.15 |
| Incomplete Teams   | 2                             | 3           | 3           | 6                | 12        | 360                         | 6             | 0.500           | 0.333         | 288              | 648                          | 0.01981 | 0.89 |
| Less Teams         | 2                             | 5           | 5           | 10               | 9         | 810                         | 7.5           | 0.500           | 0.333         | 450              | 1260                         | 0.07286 | 3.28 |
| "Non"-communicators | 1.5                          | 5           | 5           | 10               | 12        | 810                         | 7.5           | 0.500           | 0.333         | 450              | 1260                         | -0.04900 | -2.21 |
| Workaholics        | 2                             | 5           | 5           | 10               | 12        | 1080                        | 10            | 0.800           | 0.800         | 1728             | 2808                         | 0.11454 | 5.16 |
| Lazybones          | 2                             | 5           | 5           | 10               | 12        | 1080                        | 10            | 0.200           | 0.200         | 288              | 1368                         | -0.09689 | -4.36 |

The scenarios are:

1. **Base**: This scenario corresponds to the “ideal case” as in the calculations above.
2. **Incomplete Teams**: Parameters \( p \) and \( c \), which denote the numbers of players and critics per team are set to 3 respectively, instead of 5, as in the base scenario.
3. **Less Teams**: Parameter \( i \) which represents the number of teams is reduced from 12 to 9.

\( ^9 \) Empirical studies will – ceteris paribus (all other factors being equal) - show different values for \( f \) and \( s \) depending on the circumstances of the respective Syntegration event: For example, a Syntegration with obligatory participation and limited commitment of participants tends to exhibit lower values for \( f \) and for \( s \), than the ones chosen in the calculus above. The opposite - higher values for \( f \) and for \( s \) - will tendentially be the case in a Syntegration of a group of people tackling a difficult issue all of them are highly committed to.
4. **“Non”-communicators**: The average number of relationships between each pair of members of the infoset – captured by parameter $m$ - is reduced from 2 to 1.5.

5. **Workaholics**: The share of active observers – denoted by parameter $f$ - and the share of those who switch teams in a given session – $s$ – are drastically increased.

6. **Lazybones**: The share of active observers - $f$ - and the share of those who switch teams in a given session – $s$ – are drastically reduced.

The results show deviations between 0.15% and 5.16% from the ideal of $D=2.22$. The Team Syntegrity structure appears to be very robust against incompleteness of teams, weak communicators and even reduced team numbers. While the deviation of plus 5.16% in the case of the “Workaholics” scenario probably does not imply more than some unproductive work due to excessive activism, the deviation of minus 4.36% indicates that a low level of commitment may lead to some, albeit not even very strong decrease in the shared information. Altogether, it appears that it is difficult to be unproductive in the context of the Team Syntegrity structure.

5 **Synopsis and Outlook**

This paper has addressed the question of the optimality of organizational structures. First, the quest for optimality has been traced as observable throughout the endeavors of management science and organization theory. It has been shown that the pertinent research has come up with methods to identify optimal, or mostly close to optimal, solutions for specific problems. Theory has also provided models which define necessary prerequisites for viability (in the case of Living Systems Theory) and even sufficient structural preconditions for viability (in the case of the Viable Systems Model). However, the established body of knowledge has not furnished a general theorem, which would establish a norm for the dimensionality of optimally designed organizations.

Biological research into organic structures has empirically ascertained the fractal dimensionality of living organisms, which can be assumed to be optimal. Building on this body of knowledge, a new theorem for the design of optimal organizations has been proposed here. Also, a first test of the main proposition has been undertaken. The results suggest that the theorem is surprisingly accurate.

In addition to further testing of the proposition, follow-up research should address several important questions, two of which shall be pointed out here. The first question is: “What are operational measures of fractal dimensionalities, and how can they be achieved?” The second question is: “To what degree can the optimal dimensionality vary as a function of the properties of an organization, such as the cohesiveness or diversity of goals, values and preferences of its members?”

In sum, this Theorem for an Optimal Structure of Organizations is applicable to all kinds of social organisms, be they private firms, public organizations or social initiatives, etc. It opens up new prospects of a more rigorous assessment of models of structure proposed by theories of organization. But it also enables a better-founded evaluation of concrete structuring options, as well as a theory-based design and implementation of structural models in practice. In both contexts, this theorem offers a
benchmark by means of which obsolete fads and fashions can be exposed and dis-
functional propositions refuted. Finally, it must be emphasized that this theorem sheds
new light on structural issues of the design of the structures by which a society gov-
erns itself: The political system, the “state”, i.e., government and the public sector in
general can benefit from it.

Acknowledgements

The author extends his thanks to Professor Anatol Rapoport and to Mr. Peter Vrhovec
for reading drafts of this paper as well as for their valuable and encouraging com-
ments. He is also very grateful to Mr. Heiko Eckert for making him aware of Sernetz’
work and to Dr. Andreas Klose for reading parts of this paper and giving helpful
technical advice. Finally, he thanks Professor Roberto Moreno-Díaz for providing an
ideal working environment at the Institute of Cybernetics, University of Las Palmas
de Gran Canaria, where most of this paper was written during a visiting professorship.

References

ganzheitlich-verstehende Betrachtung der sozialen Leistungsordnung, Festschrift Josef
Kolbinger, Wien/New York: Springer (1985)
the Eighth Conference, New York: Josiah Macy Jr. Foundation (1952)
7. Beer, S.: Beyond Dispute. The Invention of Team Syntegrity, Chichester etc.: Wiley
(1994)
8. Espejo, R./ Schwaninger, M., eds.: To Be and Not to Be that is the System, Festschrift für
9. Fuller, R.B./ Applewhite, E.J.: Synergetics: Explorations in the Geometry of Thinking,
Completeness. Freeman, New York (1979)
12. Henschel, A.: Communities of Practice. Plattform für individuelles und kollektives Lernen
sowie den Wissenstransfer, Doctoral Dissertation, Universität St. Gallen (2001; also to be
published by Gabler Verlag, Wiesbaden)
Spectral Analysis. In: Beer, S., Beyond Dispute. The Invention of Team Syntegrity,
Chichester etc.: Wiley (1994) 263-281
14. Jestczemski, F./ Bolteraer, H. / Sernetz, M.: Comparison of the Surface Dimension and
the Mass Dimension of Blood Vessel Systems. In: Miyazima, S., ed.: The Future of Frac-
tals, Singapore etc.: World Scientific (1997) 11-20


A Planning under Uncertainty Model

Enrique Paniagua-Aris, Jose T. Palma-Mendez, Fernando Martin-Rubio
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia
{paniagua, jpalma, fmartin}@dif.um.es

Abstract. In classical planning, actions are assumed to be deterministic, the initial state is known and the goal is defined by a set of state facts, and the solution consists of a sequence of actions that leads the system from the initial state to the goal state. However, most practical problems, especially in non observable and uncertain contexts, do not satisfy these requirements of complete and deterministic information. The main goal of this work is to develop a generic planning under uncertainty model at the knowledge level enabling plan viability evaluation so that the most possible, effective, and complete plan can be determined. The proposed model in this work is presented at different levels of analysis: meta ontological, ontological, epistemological and logical levels, and applied to the post and ex ante approaches. The planning task is composed of a set of planning subtasks: plan generation, plan prevention, plan support, plan correction, and plan replacement.

1 Introduction

In classical planning, it is assumed that actions are deterministic, the initial state is known and the goal is defined by a set of state facts, and the solution consists of a sequence of actions that leads from the initial state to the goal state. However, most practical problems, specifically, non observable and uncertainty contexts do not satisfy these requirements for complete and deterministic information [1].

The main aim of this work is to develop a generic planning under uncertainty model at the knowledge level for planning in non observable an uncertain contexts. This kind of contexts make plans viability evaluation necessary since the unique solution which may be produced, has to be stated in terms of the most possible and effective plan. Of course, since the beginnings of Artificial Intelligence, a lot of effort has been made by researchers in developing intelligent planning systems. Therefore, even today, there are some problems that are not completely solved, referred to as, Qualification Problem [2], [3], Frame Problem [4], Ramification Problem, and Effects Determination Problem [5], [6].

If all these problems are taken into account, a process for the estimation of plan viability is necessary. Plan viability can be defined in terms of possibility measure or some certainty degree. The use of viability measures for the estimation of plan viability conforms a specific kind of planning problem, known as planning under uncertainty.
Several papers describe planners which generate conditional and probabilistic/possibilistic plans for solving the planning under uncertainty problem. In this paper, we synthesize this work, making an analysis of the post and ex ante approaches at the different commitment levels of the knowledge level [7]: meta ontology, ontology, epistemology and logic. Finally, we present a unifying model of the planning under uncertainty task, decoupling the planning process with the planning model [8] into planning generation, planning prevention, plan support, plan correction and plan replacement.

2 The Ex Post and Ex Ante Approaches

There are two ways to approach the development of an intelligent planner. The ex post way: the planner comes first, and the attempt to formalize it later (STrips [9], SNLP [10], CNLP [11], Buridan [5], C-Buridan, Cassandra [12], PosPLan [1], or Mahinur [13]), and the ex ante way: the formalization comes first, and the planner is implemented later (Situation Calculus, Possible Worlds, PASCALE [6], or UCPP [14]). Figure 1 shows the chronological evolution of the ex post and ex ante approaches, and Table 1 shows the different contributions of ex post and ex ante approaches.

![Fig. 1. Chronological evolution of ex post and ex ante planners.](image-url)
<table>
<thead>
<tr>
<th>Ex post</th>
<th>Contribution</th>
<th>Ex ante</th>
<th>Contribution</th>
</tr>
</thead>
</table>
| STRIPS | State Description: Conjunction of Propositions  
State Changes: STRIPS Operator (STRIPS assumption)  
Assessment: All sub-goals obtained | Situation Calculus | State Description: Situations  
State Changes: result predicate, frame axioms (automatic theorem demonstration) |
| SNLP | Composition: Complete partial plans (least commitment strategy)  
Assessment: Complete plans | Possible Worlds | State Description: Possible worlds consistent with Domain Constrains  
State Changes: the conjunction of the effect possible worlds (conservative model) |
| CNLP | State Changes: STRIPS operators transformed to Context dependent actions  
Composition: Partial plans with reasons, contexts and conditioning links  
Assessment: Complete and Solid plans | PASCALE | State Description: Possible Models consistent with Background Knowledge, Incomplete descriptions  
State Changes: Context dependent actions with uncertain effects  
Composition: Similar to SNLP  
Assessment: probabilistic model |
| Cassandra | State Changes: STRIPS operators transformed to Context dependent actions and uncertain effects  
Composition: Similar to SNLP with information gathering steps | CommonKADS Library | Definition of the Plan Model |
| Buridan | State Description: Probabilistic distribution of situations  
State Changes: STRIPS operators are transformed to context dependent actions with uncertain effects  
Composition: Similar to SNLP (plan refinement)  
Assessment: Achieving the sub-goals with a threshold | UCPP | Definition of the preventive, corrective and replacement strategies for conditional and probabilistic planning |
| PosPlan | State Description: Possibilistic distribution of situations  
State Changes: STRIPS operators are transformed to context dependent actions with uncertain effects  
Composition: Similar to SNLP  
Assessment: Achieving the sub-goals with a necessity (plan acceptability) | MPUU | Definition of a generic planning under uncertainty model at the knowledge level |

**Table 1.** Contributions of *ex post* and *ex ante* approaches.
The approach followed here consists of applying *inverse engineering* process to the *ex post* planners, and a modeling approach at knowledge level to the *ex ante* planning theories in order to define a generic model for a planning task in uncertain environments. In this sense, we propose to make an analysis of the different components of the classical planning model [8] at four levels, i.e. meta ontological, ontological, epistemological and logical [15], in order to accommodate it to the planning under uncertainty model. In Figure 2 we show the different components of the planning model.

![Components of the planning model](image)

**Fig. 2.** The components of the planning model.

### 3 The Meta Ontological Commitments

Classical planning systems are based on the following commitments: 1) the initial state is assumed to be known, 2) the goal is defined by a set of final facts, and 3) the actions are deterministic (i.e. each effect state of the application of an action is true or false, and there is only one effect state that is true). However approaching a planning system under uncertain context imposes new commitments at meta ontological level:

1. The planning task is an off-line process. The whole system is supposed to be a closed energy system. Therefore, all the knowledge that may be generated can be obtained from the set of possible initial states as a combination of the initial knowledge. Here, the certainty degree associated to the new possible states is always less than or equal precise as the certainty of the initial knowledge (in a multi-valued logic model).
2. State descriptions may be incomplete, but in this work we assume that they are complete.
3. The environment is assumed to be static, that is, changes in the environment are produced as a consequence of the plan execution.
4. Actions are context-dependent with non deterministic effects, qualifications and ramifications, that is, each possible effect is a consequence of a specific context; the effects of a context can generate multiple possible states; the set of contexts is exhaustive and exclusive; each action is applicable to a situation if the preconditions of the actions is a subset of the state facts of the situation; and each effect can trigger a causal law of the scenario domain.
5. The sets of possible states (i.e. the set of possible initial states and the set of possible temporal states) are uncertain.
6. The goal is assumed to be flexible, that is, we want to obtain the set of goal facts with a certainty degree.
7. The decision process is non observable. In an off-line system, we can only simulate the information gathering process by informational actions, which generate exhaustive alternative branches on the plan. It is a direct consequence of the first commitment.

Taking into account all these commitments, a plan generated following this model, must be a) complete (i.e. a causal link must exist between each sub goal and a plan step that resolves it), b) solid (i.e. if we apply all the steps of the plan to the set of initial possible states a final possible state with the subset of goal facts is obtained), c) possible (i.e. the degree of feasibility of all the steps of the plan is greater than a success threshold), and d) effective (we obtain the set of goal facts with a minimum distance). This restrictions impose some ontological level requirements on the planning task, then the following subtasks must be taken into account:

Plan completeness. This subtask obtains a set of quasi complete plans that solve all the goal facts. These plans may contain threats (i.e. state facts that may contradict the preconditions of a step in a time interval).
Plan prevention. This subtask prevents the plan, assuming that we go from a set of possible initial states to a set of possible states that confirms the goal facts.
Plan support. In case of threats, a refinement of the initial plan in order to increase the possibility of the goal facts must be applied.
Plan correction. This subtask is applied to the initial plan when there are non desirable effects (i.e. when a plan step generates negative branches (contexts) we must add a contingent plan to it).
Plan replacement. Applied to the planning process when we must obtain a more successful plan.

4 The Ontological Level

At ontological level, all the concepts related to the planning model, as well as the system dynamic processes, are presented. Among these concepts we can find (using the terminology used in CommonKADS Library [8]):

4.1 The Planning Problem

A planning problem, PP, can be defined as:

\[ PP = \langle \text{Initial-States, Goal-Facts, Actions, Threshold-Success} \rangle \]
where Initial-States is the set of possible initial states, Actions is the set of scenario operators that can be used to change the environment, and Threshold-Success is the minimum necessity degree to be obtained for each state fact from the set Goal-Facts.

4.2 World Description

The World Description is composed of the State Description and the State Changes:

State Description. Each possible state is a model of the scenario domain (i.e. the scenario domain is composed of a set of scenario domain constraints that are tautologies and a set of causal laws. The former defines what is always true in the scenario, and the latter defines the possible ramification effects. Each possible state must be consistent with the scenario domain constraints), with a set of state facts, and each possible temporal situation is a set of possible states. This State Description is based on the Possible Worlds [3] and Possible Models [6] theories. The representations of the different elements of the State Description are:

- Scenario Domain Knowledge = \langle \text{Scenario-Domain-Constrains, Scenario-Causal-Laws} \rangle
- State Fact = A scenario atomic proposition.
- Possible State = Set of state facts.
- Possible Situation = Set of possible states.

State Changes. The way the state changes is defined by a set of actions with context-dependent uncertain effects and qualifications. A set of informational actions permits definition of the exhaustive contexts to which the actions are applied. The representation of actions is based on an extended model of the STRIPS [9], Buridan [5], PASCALE [6] and Cassandra [12] systems, in which we have included some modification in order to include information about the different types of threats.

- Action = \langle \text{Action-Intention, Action-Execution} \rangle
  - Action-Intention = \langle \text{Preconditions, Inevitable-Threats, Possible-Threats} \rangle
  - Action-Execution = \langle \text{(Execution-Conditions, Execution-Deterministic-Effects, Execution-Disjunctive-Effects, Execution-Threats)} \rangle
- Informational-Action = \langle (\text{Preconditions, Execution-Disjunctive-Effects}) \rangle

Action-Intention describes the subset of state facts that must pertain to the actual situation, the subset of state facts that must be false in the actual situation, and the subset of state facts that possibly can negate the applicability of the actions.

Action-Execution describes the set of contexts, which are composed by the description of the context, the deterministic effects, the disjunctive effects and the set of possible state facts that can generate lethal effects.

Informational-Action is an action that generates no changes in the environment, it creates the set of exhaustive possible results that are obtained by the gathering information of a state fact.
4.3 Plan Description

The Plan Description component is divided into Plan Assessment and Plan Structure.

Plan Assessment. The Plan Assessment subtask uses the following evaluations: Plan Feasibility Assessment, Plan Effectiveness Assessment, and Plan Completeness Assessment. The first is a necessity degree that describes the Intention and Execution Assessment of each step of the plan; the second is the distance between the goal facts and the state facts of the final situation of the last step of the plan; and the third is a binary logical value that indicates if all the causal links of the plan realise all the goal and temporal sub-goal facts of the plan.

Plan Structure. The plan representation is based on quasi complete partial plans. It is a partial plan with a number of steps, a set of temporal ordering, and a set of causal links that achieve every goal fact. The representation of a quasi complete partial plan is:

\[
\text{Quasi Complete Partial Plan} = \langle \text{Initial-States}, \text{Goal-Facts}, \text{Steps}, \text{Temporal-Ordering}, \text{Causal-Links}, \text{Threshold-Success} \rangle
\]

Where Initial-States is the possibilistic distribution of the set of possible initial states; the Goal-facts is the conjunction of state facts that we want to achieve; Steps is the set of \( \langle \text{Initial Situation, Action, Final Situation} \rangle \) of the plan, temporal-Orderings is the set of temporal constraints between the steps of the plan; Causal-Links is the set of \( \langle \text{Producer Step, State Fact, Consumer Step} \rangle \), and finally; threshold-Success is the minimum necessity degree the plan must achieve.

4.4 The Planning Task

The Planning Task is divided into Plan Generation, Plan Prevention, Plan Correction, Plan Support, and Plan Replacement Sub-Tasks.

Plan Generation Subtask. In the Plan generation Subtask we obtain a quasi complete partial plans that solves the set of goal facts [10]. We use the Complete Plan process, with the non deterministically add-step and reuse-step actions to achieve all the sub-goals of the plan, and the non deterministically demote, promote and confront actions to solve the temporal threats in the plan. Figure 3 shows an Inference Scheme (CommonKADS formalism) of Plan Generation Sub-tasks.

Plan Prevention Subtask. The first process in the prevention of a plan is performed by Plan Projection sub-task which consists of decoupling the assessment process into Action Intent Assessment and Action Execution Assessment, for each step of the partial quasi complete plan. The first is related to the evaluation of the applicability of an action in a given situation with certain action qualifications, it consists of reaching out the preconditions of the action, negating the inevitable threats and evaluating the possible threats; the second is related to the expansion of the actual situation to all the
possible situations produced by the action execution, it consists of the generation of all possible states, combining the deterministic effects of each applicable context with all the disjunctive effects, and the lateral effects triggered by the scenario causal laws of the scenario domain knowledge and the possible execution threats.

![Plan Generation Subtask Diagram]

**Fig. 3. Plan Generation Subtask.**

**Plan Correction Subtask.** With the Correct Plan process the possible execution threats that can be the cause of possible lethal effects are reduced by the addition of contingency actions if the most possible situation is the situation that contains the lethal effects. In Figure 4 we show the Plan Projection Subtask, which includes the Plan Correction Subtask.

**Plan Support Subtask.** This subtask is activated after the evaluation of the feasibility of each goal fact, that is, all the sub-goals are obtained with a necessity degree greater than the success threshold. If the plan is not feasible, then we activate the Support Plan process. It consists of reusing plan steps or adding new step which increases the necessity degree of the sub-goals of the plan.

In Figure 5 we show the Plan Prevention Subtask, which includes the Plan Support Subtask and Plan Assessment Subtask.

**Plan Replacement Subtask.** We apply the plan replacement subtask when we can obtain a more feasible and effective plan than the current quasi complete partial plan. This consists of generating a new quasi complete plan, applying the prevention, correction and support tasks to it, and evaluating the feasibility.
Finally, in Figure 6 we show the Planning Task, including the Plan Replacement Subtask. Figure 7 shows the planning task control.
5 The Epistemological Level

At epistemological level, the main commitments are based on the fact that probabilistic information about the state facts is not present, the distribution of the possible states is not available, and the context-dependent effects of an action applied in a context are undefined. These restrictions force us to use a possibilistic model approach, in which:

```
PLAN (init, goals, actions, success)
plans ← make-init-plan (init,goal)
complete-plans ← ∅; success-plans ← ∅
while plans is not empty do
  CHOOSE (and remove) a plan P from plans
  if P is complete then
    complete-plans ← complete-plans ∪ P
  while complete-plans is not empty do
    CHOOSE (and remove) a plan P’ from complete-plans
    P’ ← prevent(P’)
    if feasibility(P’) ≥ success then
      success-plans ← success-plans ∪ P’
    else
      P’ ← support(P’)
      (if can be applied)
  else
    plans ← complete(P, actions)
  return more-feasible(success-plans)
```
Each state fact has a necessity degree, \( N(\text{state-fact}) \).

Each possible state has a necessity degree measured by Shannon’s entropy over all state facts, \( N(\text{state}) = Sh(\langle N(\text{state-fact}) \rangle) \) [17].

Each set of possible states has a necessity distribution, \( \pi(N(\text{state})) \) [18]. It describes the uncertainty related to a set of possible states.

The necessity degree of the most possible situation of a set of possible states is the most Shannon-entropic state, \( \text{Max } Sh(\text{state}) \) [17].

The necessity degree of a context-dependent effect is the possibilistic modus ponens extension, \( \text{Min}(N(\text{preconditions } \rightarrow \text{ effects}), N(\text{preconditions})) \) [18].

Each partial complete plan has associated a necessity measure (N), and a feasibility measure (F), which can be obtained by the expressions:

\[
N(\text{plan}) = \text{Average}(\langle N(\text{step}) \rangle),
\]
\[
N(\text{step}) = N(\text{most-possible-situation(action(step)))},
\]
\[
F(\text{plan}) = \text{Average}(\langle \text{Min}(\text{Intention(step), Execution(step)}) \rangle),
\]
\[
\text{Intention(step)} = \text{Min}(\text{preconditions(action(step)))},
\]
\[
\text{Execution(step)} = N(\text{most-possible-situation(possible-states(Final-States(step)))}).
\]

Effectiveness of a plan is based on the distances from the state facts necessity degrees of the most possible situation of the last plan step to the sub-goals necessity degrees.

\[
\text{Average}(\langle N-\text{Distance}(\text{MPS(Final-Step(plan)}) \cap \text{sub-goal}) \rangle)
\]

### 6 The Logical Level

For the purpose of this work, the main commitments at the logical level are related to the necessity degree and effectiveness of a plan. All the state facts can be considered as propositions, and the necessity of a state can be defined as the entropy level of the state facts. With this commitment the use of a possibilistic propositional logic is necessary. The Logical System is defined as:

\[
\text{LS} = \langle A, S, X, \text{IR}, I \rangle
\]

where A is the alphabet of the logical system, i.e. the state-facts; S is the syntax of the logical system, i.e. state-facts, \( \wedge \) of state-facts (possible-state), and \( \wedge \) of state-facts \( \rightarrow \wedge \) of state-facts (context-dependent effects of an action and causal laws of the scenario domain knowledge); X is the set of fuzzy logic axioms; IR is the possibilistic modus ponens extension; and finally, I is the semantics of fuzzy logic.
7 Conclusions

In this work, a generic planning under uncertainty task is specified at the knowledge level. This work is based on an analysis of the *ex post* and *ex ante* approaches to the planning problem (included the CommonKADS library [8]) through different levels of description: meta ontological, ontological, epistemological and logical. This analysis allows us to define the commitment on the planning task at these four levels.

A planning task under uncertain and non-observable contexts makes the evaluation of plan viability and effectiveness necessary, since the solution (the plan) must be stated in terms of the most possible plan. This requirement implies the definition of some tasks at Knowledge Level to include a plan evaluation process. It consists of applying the feasibility and effectiveness measures of a complete partial plan, in the Prevention Task.

The main contribution of this work is the definition of the planning task at the knowledge level in uncertain contexts. In [8], the planning task is defined in certain contexts, and oriented to the plan description, while our work is mainly oriented to the world description to solve the uncertainty problem. In [14], a set of strategies of planning under uncertainty is defined, but it has the lack of the description at the knowledge level. Finally we claim that the planning task must be modeled making an analysis of the four commitment levels of the knowledge level.

Our future work will be aimed at the following purposes:

The definition of a language for specifying the planning problem.
Obtain a library of methods to perform the proposed subtasks, and a formalism, based on the context requirement, that makes the automatic configuration of the planning task possible.

References


Jose F. Bienvenido¹; Isabel M. Flores-Parra¹; Rafael Guirado¹; Roque L. Marín²

¹ Dept. Lenguajes y Computación. University of Almería
La Cañada – E-04120 – Almería - Spain
{fbienven, iflores, rguirado}@ual.es
² Dept. Informática, Inteligencia Artificial y Electrónica. University of Murcia
Espinardo – E-30100 – Murcia – Spain
roque@dif.um.es

Abstract. Actual advancements of the design tools for industrial elements include the addition of knowledge based elements to the classic design tools. In this sense, we propose a general design model, which is represented using an extension of CommonKADS, as a framework for the development of design tools for different industrial objects. This model was used developing the latest version of DAMOCIA-Design, a knowledge based design tool of greenhouse structures. One key point of our design framework is using alternative knowledge blocks, which are selected dynamically using an ad-hoc mechanism, assembled using sets of selection criteria associated to the methods. Keywords: Knowledge-based CAD, CommonKADS, dynamic selection of methods, design modeling.

1 Introduction

CAD systems have supported the design of new industrial elements during a long time. Starting as simple representation tools, they have evolved to more advanced tools that simulate the diverse systems and artifacts (from simple chairs to complex airplanes) and simplify their manufacturing processes (CAM). However, it has been during the last decade that it has been proposed, with some success, the automation of the kernel step of every design process, the proposal of new solutions to a given problem. In this sense, the first step was the development of general models of the design process, as the General Design Theory of Yoshikawa [1] [2] or the Knowledge Level Design Theory of Smithers [3] [4]. Actually, this evolution, shown briefly in figure 1, has been possible because of the great advances into the acquisition, representation and management of the knowledge, as we can see in [5]. Knowledge Engineering is one of the keys of the actual and future improvements into the automation of the design process and the development of more powerful design tools.
Automation of the design definition process possibilities the transition from the mass production schema (great series of object from a short catalogue) to the mass customization model (short series of objects from a wide catalogue) [6] [7], as shown in figure 2. Actual saturation of the production possibilities for multiple industrial elements forces the evolution of producers to this new paradigm in order to survive the global market, giving most practical importance to the improvement of the actual design frameworks.
Developing a specific design tool which objective was to guide the design of agricultural structures, we realize the convenience of including knowledge about the design process into it, in order to simplify a great number of cases. Modeling the design process, we found that if was possible to generalize the design schema, covering a great variety of industrial elements. This way, we developed a general design model, called STE (from Specification→Translation→Evaluation) [8], which was modeled using the CommonKADS methodology, as described in [9]. During the modeling process, it was required to extend the CommonKADS notation schemas (CML and task-method diagrams) in order to include new mechanisms of dynamic selection of the methods [10]. In this work, we present the STE design model developed, including a brief description of the mechanism of dynamic selection of methods developed.

2 STE Design Model

We propose a general design model, STE, based on these principles:

- The designed object will be designed in an incremental way. Each step supposes a refinement of its description.
- It is possible to give a formal description of the object at each design step.
- The design process supposes the refinement of an Initial Description (problem domain) until a Final Description (solution domain), via a set of transformations that generate intermediate descriptions (that are included into the work domain).
- Each transformation step, which obtains a new formal description of the object, includes mostly three different types of tasks: specification (S), translation (T) and evaluation (E).
- Specification tasks add some information to the object description. This information can be given by the user (the designer) or extracted from a design repository. As different human designers can specify unequal sets of additional data, it would be possible to include alternative automatic specification blocks from the design repository (corresponding to alternative human proposals).
- Translation tasks integrate previous descriptions of the designed object with new blocks of information (obtained by the specification tasks), generating new and more complete descriptions of the object. With our proposal, each step of the design process supposes the use of a different formal language to describe the object (al least the new step language is an extension of the former one). A translation task transforms a description in the formal language of a step to that of the next one.
- Each design step can include a set of evaluation tasks, which objective is to test the correction and suitability of the actual solution. They can demand the modification of the complementary data blocks (generated by the specification tasks) and the repetition of the translation tasks. If the possible backtracking process involves only the actual step tasks, the evaluation task is referred as a “local evaluation task”. When it affects previous steps, it is referenced as a “global evaluation task”. At a specific design step it is possible to find multiple
evaluation tasks (specialized in different analysis and/or models) or even no evaluation task.

- Specific design problems represent specific methods assignments to the STE tasks. The global architecture of the STE tasks remains the same.

In order to formalize the proposed STE design model, it was necessary to extend the Task Model Diagrams used in the CommonKADS methodology [9], combining it with the Jackson System Development notation [11][12]. Figures 3 and 4 resume this notation, showing their basic elements and relations. This is a simplified version of our Extended Task Method Diagrams (ETMD), where there is no explicit representation of the TASK-METHOD BRIDGES [8].

![Fig. 3. Extended Task Method Diagrams basic elements](image)

Using this graphical notation and an extended version of the CML, we developed a general version of the STE model, as shown in figure 5, where after an initial description of the object (usually a basic description of the required performance), it is activated a set of design STE steps. These steps include specification, translation and evaluation tasks specific of the design problem (greenhouses, plane parts, software architecture, etc).

This model is extremely general, but, when implementing the specific knowledge model, it requires taking account of the step (managing explicitly the level we are working with) inside the task-method bridges in order to assign correct methods to the specification, translation and evaluation tasks.

We choose, then, to develop a new version of the design model where the STE tasks of each design step would be represented differently. This way, we added a set of more specific STE models to our design ontology, classified by the number of design steps and the type of evaluation tasks included (global or local), simplifying the construction of specific design models and reusing the highest parts of the CML description. Table 1 presents the classification of the new general design methods of the extended ontology.

Figures 6a and 6b show the extended TMDs of the design tasks with three steps with both local and global evaluations. Figure 6c presents the structure of the design task for one level (step) of any of these multilevel models. Each specific one-level design task will be refereed with a different name (level number).
Assigning a method to a task

**Task**

Method

Direct resolution of a method via an inference

Method

Inference

Decomposition of a method as an iterative task

Method

Task

Alternative solution methods assigned to a task

**Task**

Method-1 . . . Method-n

Sequential decomposition of a method in different task and/or transfer functions

Method

Subtask/transfer function . . . Subtask/transfer function

Decomposition of a method as a set of alternative tasks

Method

Task-1 . . . Task-n

Decomposition of a method as a set of parallel tasks

Method

Task-1 . . . Task-n

Fig. 4. Relations between simple knowledge elements
Table 1. General design methods with predefined number of steps.

<table>
<thead>
<tr>
<th></th>
<th>1 step</th>
<th>2 steps</th>
<th>3 steps</th>
<th>...</th>
<th>n steps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local evaluation</strong></td>
<td>STE-1l</td>
<td>STE-2l</td>
<td>STE-3l</td>
<td>...</td>
<td>STE-nl</td>
</tr>
<tr>
<td><strong>Global evaluation</strong></td>
<td>STE-1g</td>
<td>STE-2g</td>
<td>STE-3g</td>
<td>...</td>
<td>STE-ng</td>
</tr>
</tbody>
</table>

Fig. 5. Simplified TMD of the most general design model

The model of design, represented with these knowledge models, can be alternatively described using a classic block diagram. Figure 7 shows the classic block diagram of the three levels STE design model, including, in this diagram, main data and knowledge blocks. In this diagram these control and data flow tracks:

(1) Main line of execution of tasks. Represents the 3 steps transformation of the object description.
(2) Line of incorporation of new data and knowledge, linking the specification tasks with the translation tasks.
(3) Line of local evaluation. Control flow to and from the local evaluation tasks.
(4) Line of global evaluation. Control flow to and from the global evaluation tasks.
(5) Actualization of previous data and knowledge as a consequence of the evaluation tasks (higher number for local tasks).
a) Root of the STE-3l design model

b) Root of the STE-3g design model

c) Extended TMD of a general design level

Fig. 6. Design tasks using methods for a fixed number of design steps
Fig. 7. Example of STE implemented architecture for a three step design process
This block diagram suggests implementing the design systems using a distributed architecture. In our case, we have used a general distributed architecture named DACAS [13].

3 Dynamic Selection of Methods

Different specification, translation and evaluation tasks (and their subtasks) can be completed by the execution of alternative methods, which implement different principles, precedence and heuristics. These methods can be added at anytime of the product life cycle. In order to activate the different methods, we propose a framework for the dynamic selection of methods [8], with these characteristics:

- Main elements of the selection process are:
  - Suitability criteria, in the sense of those defined by Benjamins [10].
  - Criterion weights. These modulate the importance of the different criteria for a given task.
  - Data and knowledge of selection. Domain knowledge elements related with the selection criteria and weights.
  - Aggregation functions, that integrate the different selection criteria into single precedence values.
  - A selector. A general mechanism that implements the selection process and manages the method and task specific elements.
- All the general elements, that are independent of the application, can be assembled as a general selection framework, which is included into what we have named meta-application.
- It is possible to implement different models of selection. At this moment, we have developed these:
  - Simple selection.
  - Parallel selection.
  - Waterfall selection.

All of them applicable to a given aggregation function, as the multicriteria function we have used into our case application.

This way, using the mechanism of dynamic selection of methods proposed, it is possible to include into the design tool, not only different blocks of declarative knowledge (basically data), but also different work strategies. These represent different approaches to the design task (usually extracted from different specialists). We can include alternative methods of consecution of the goals for specification, translation and evaluation tasks.

Using an adequate set of selection elements (mainly selection criteria and weights), it is possible to add new design strategies to the tool including into the ontology new methods and establishing correct task-method bridges. Each alternative method includes inside their design potentials via their selection criteria values.

Each different design strategy is included, basically, into the system by an alternative set of specification, translation and evaluation methods, as shown in figure
8. Different design strategies can share some specific methods (in the example, the strategies 1 and 2 of the example use method $S-n$) or even part of the methods (sharing methods located deeper into the knowledge model).

![Flowchart of Design Strategies]

**Fig. 8.** Inclusion of different design strategies

4 Case Application

Using the general design model and the schemes of dynamic method selection proposed, it has been developed a new greenhouse design tool, DAMOCIA-Design. Figure 9 shows its initial screen and figure 10 some results.
Fig. 9. DAMOCIA-Design main and structural definition screens
Fig. 10. Some results obtained automatically, DXF plans and detailed budget
Technical information about this product can be found in http://www.prosoma.lu/(Damocia) and http://www.acad.bg/esprit/src/results/res_area/iim/iim15.htm. This development was done with the financial aid of the EU into the framework of the ESPRIT (P7510 PACE) and the Ministry of Industry of Spain (PATI PC-191).

5 Conclusions and Further Works

Main conclusions are:

− Modern knowledge acquisition and modeling techniques are suitable to build powerful models of the design process. It is possible, using a methodology as CommonKADS, to develop models of the kernel process of the design activity, the new solutions searching.

− The STE design model, composed by several steps where formal descriptions of the designed object are extended (with specification tasks), translated to more suitable descriptions and evaluated, has been formalized using an extension of the CommonKADS representation tools. This model is applicable to the design of multiple industrial elements.

− The existence of diverse design strategies is possible into the STE model using alternative design methods (at diverse levels of the design knowledge model). We propose a general framework for the dynamic selection of methods, which permits the implementation of these different strategies.

− The STE model and the mechanism of dynamic selection of methods have been used, without problems, to develop a commercial design tool, DAMOCIA-Design.

Further works include:

− Extending the application of the STE model to other areas, as the developing of aeronautical parts (actually in progress) or software.

− Assembling a most detailed ontology of the design methods, generalizing the methods we have developed (for the design of the agricultural structures and aeronautical parts). It is interesting to have a library of methods of specification, translation and evaluation, adequately abstracted.

− Moving the actual ontology of design knowledge elements to a distributed web based library. This way, it would be easy to share all these elements between diverse designers.

References

5. Multiple authors, Configuration-Design Problem Solving, special number of the IEEE Intelligent Systems & their Applications, vol. 12 n. 2, 1998
Complexity Analysis of Systems from a Functional and Technical Viewpoint

Stefan Dierneder and Rudolf Scheidl

Department for Foundations of Machine Design
Institute for Mechanics and Machine Design
Johannes Kepler University of Linz
Altenbergerstrasse 69, A-4040 Linz, Austria
{dierneder, scheidl}@mechatronik.uni-linz.ac.at

Abstract. Nowadays, complexity analysis of functional and technical mechatronic system becomes more and more important. This is because of complexity influences almost all phases of product design and system engineering. Therefore, there is the demand for a good assessment of complexity. For this purpose, several questions have to be discussed.

→ What is complexity and what are its characteristics?
→ What are the criteria for a good method to evaluate complexity?
→ How could complexity be evaluated and which method fits the requirements best?

1 What Is Complexity and What Are Its Characteristics?

Many researchers like D. Adam, U. Johannwille, W. Eversheim, F. B. Schenke, L. Warnke, and, R. Beensen have considered the nature of complexity.

D. Adam and U. Johannwille defined complexity in [1] as follows: “Complexity is the totality of all characteristics of a state or an object by the meaning of many-sided!” Furthermore, they pointed out that complexity appears in all phases of product-life-time, such as product development, product planning, product management, cost calculation, manufacturing, logistics, and so on.

W. Eversheim, F. B. Schenke, and, L. Warnke defined complexity in [2] from a totally different point of view. They say: “In general complexity is the huge number of elements with a high degree of netting!” Furthermore, they mention that complexity appears in all areas of product development and leads to loss of transparency and high expenditure of co-ordination. Therefore, it is important to determine and reduce complexity in all phases of system-engineering. Following the authors of [2], they point out that excessive complexity is a fact of the high variety of the functionality of nowadays products.

Such definition of complexity by the degree of netting seems to be all right. But, taking only the variety of the functionality as cause of excessive complexity is too restrictive. The influence of the functional principle, which forces complexity much more than the functional variety should be considered too. To say it in explanation: It is much easier to understand a system with many functions, but structured in an obvious way, than a system with only a few functions realised very lavish.
Taking this fact into account, Nam P. Suhs definition of his so called information-content, see [3], seems to be a good approach for measuring complexity.

The above given definitions of complexity only consider the functional facts, but the different points of view are not taken into account. The importance of the different points of view, which influence the human sensitivity on complexity, is shown by the following examples.

Complexity sensation of a car-driver is quite different from that of a car-mechanic. The driver judges the motor-car mainly by its simplicity of operation, the car-mechanic from his professional point of view. Even a simple cube of steel offers lot of complexity for a metallurgist but only little complexity for an “ordinary human being”. It is a fact, that there is some difference between these given considerations, if there is no additional information, which kind of complexity is of interest.

Therefore, the term “considered” complexity is introduced and it should be used at any time to achieve comparable complexity statements. The second fact, which is of interest, is the evaluation of complexity.

2 What Are the Criteria for a Good Method to Evaluate Complexity?

A literature study [1-5] combined with own proposals results in five main-criteria for a generally valid complexity evaluation.

1st criterion “intuition” ⇒ use the same degree of decomposition for comparison purposes

2nd criterion “sensitivity” ⇒ use a sensible depth of decomposition

3rd criterion “consistency” ⇒ the complexity value of a considered system must be higher then the complexity value of its components

4th criterion “general validity” ⇒ the evaluation method must be applicable to all other systems

5th criterion “simplicity” ⇒ the complexity value must be easily to understood and perfectly clear

3 How Could Complexity Be Evaluated?

3.1 Existing Complexity Values – Results of Literature-Study

Two methods for the evaluation and the judgement of complexity are the methods from Griffin [4] and Kannapan [5]. Both are based on the number of included functions $F_i$.

$$PC = \sum_i F_i \quad \text{PC ... product complexity} \quad (1)$$
This definition does not meet the above mentioned criteria:

The example motor-car shows a hurt of the “sensitivity”-criterion. This is, because of many cars have the same number of functionality, but reflect a difference in complexity.

A further inconsistency of equation (1) is shown by the example flashlight. Following Griffin, the flashlight has the only functionality to produce light. Hence the PC-value is one. But the battery, which has to provide energy has as well as the flashlight the complexity value one. This is the inconsistency on the third complexity criterion, because of the battery is part of the flashlight.

Taking these inconsistencies into account, Bashir and Thomson propose the extension of Griffins and Kannapans method by multiplying the functionality with its level number of the functional tree.

\[ PC = \sum_{j=1}^{l} F_j \cdot j \]

where \( l \) is the number of layers of the functional tree, \( j \) is the layer of the actual counted functions and, \( F_j \) the number of functions at layer \( j \).

Equation (2) is strongly oriented on the functional tree. For further explanations see [6]. But (2) contains an inconsistency, too.

Figure 1 makes clear, that the same function, namely “x”, contributes two different amounts to the product complexity value, and this is inadequate.
3.2 Own Complexity Values

Own considerations are also based on the functional tree, but give strong emphases to the product characteristics such as couplings, tolerances, and so on.

To meet all requirements on complexity evaluation, three complexity values are proposed.

- the Functional Product Complexity $FPCI$

$$FPCI_j = \sum_{i=1}^{N_{Fj}} I_{\kappa_{Fj}} + \sum_{i=1}^{N_{TRj}} I_{\kappa_{FTRj}}$$  \hspace{1cm} (3)

- the Technical Product Complexity $TPCI$

$$TPCI = \sum_{i=1}^{N_{TR}} I_{\kappa_{TR_i}}$$  \hspace{1cm} (4)

- the Reliability Product Complexity $RPCI$

$$RPCI = \sum_{i=1}^{N_{TR}} I_{pTR_i}$$  \hspace{1cm} (5)

with

- $j$ ... level of the functional tree
- $N_{Fj}$ ... number of functions at level $j$
- $N_{TRj}$ ... number of all technical realisations up to level $j$ (level $j$ included),
- $I_{\kappa_{Fj}}$ ... functional coupling information content = information content based on the existing functional couplings
- $I_{\kappa_{TR_i}}$ ... technical coupling information content based on the existing technical couplings of the design parameters and functional requirements
- $N_{TR}$ ... whole number of technical realisations
- $I_{pTR_i}$ ... reliability information content based on the reliability of manufacturing, assembling, design, and so on

These evaluation methods for complexity judgement are strongly influenced by Suhs information content, but there is some difference between Suhs [6] focuses only on the reliability of manufacturing and assembly. Thus, in his work there is no relationship to his independence axiom, which reflects the coupling situation of the given functionality and the selected design parameters only
in a systematic way. But in case of complexity analysis it is important to evaluate the coupling situation. This will be done by the functional and the technical information content, where a low number of couplings also results a low information content and vice versa.

**Functional Product Complexity (FPCI).**

$$FPCI_j = \sum_{i=1}^{N_{F_i}} I_{\kappa_{F_i}} + \sum_{i=1}^{N_{TR_j}} I_{\kappa_{F_i}}$$

This complexity value is the sum of all the so-called functional coupling information contents up to a certain level \( j \). This information content essentially reflects the coupling characteristics of the considered function and reads as term (7).

$$I_{\kappa_{F_i}} = \log_2 \left( \frac{1}{1 - \kappa_{F_i} \cdot \frac{N_{SF_{F_i}}}{N_{SF_{TR_j}}} \cdot \left(1 - \epsilon_{I_{100\%}}\right)} \right)$$

with \( 0 \leq I_{\kappa_{F_i}} \leq I_{100\%} \) and \( 0 \leq \kappa_{F_i} \leq 100\% \), where the limitation factor \( \epsilon_{I_{100\%}} \) represents the information content of a totally (100%) coupled system, where every function of the functional tree is in relation to each other.

\( N_{SF_{F_i}} \) in equation (7) represents the actual highest known number of sub-functions of the functional element \( F_i \) and \( N_{SF_{TR_j}} \) is the according overall number of functional elements \( N_{F_i} \) and technical realisations \( N_{TR_j} \), where \( k \) is the number of the level, the just mentioned sub-functions are part of.

The maximum-value of \( I_{\kappa_{F_i}} \) reads as

$$I_{\kappa_{F_{max}}} = I_{100\%} = \log_2 \left( \frac{1}{1 - 100\% \cdot \left(1 - \epsilon_{I_{100\%}}\right)} \right) = \log_2 \left( \frac{1}{\epsilon_{I_{100\%}}} \right)$$

Alternatively the limitation factor could be calculated.

$$\epsilon_{I_{100\%}} = \frac{1}{2^{I_{100\%}}}$$

e. g. defining \( I_{100\%} = 1 \) results in \( \epsilon_{I_{100\%}} = \frac{1}{2} \).
If top-down analysis (from the overall to detail) is in use, the calculation of the degrees of the existing functional couplings, part of term (7), requires the separation into two cases:

- The considered functional element actually is no elementary function: Thus, internal couplings can exist, and, the considered function is able to couple too.

  The first approximation of $\kappa_{F_i}$ is:

  \[
  \kappa_{F_i} = \frac{N_{x_{F_i}}}{N_{F_j} + N_{TR_j}}. \tag{9}
  \]

- The considered functional element describes an elementary function: This means, that there is no lower functional element and $\kappa_{F_i}$ calculates as:

  \[
  \kappa_{F_i} = \frac{N_{x_{F_i}}}{(N_{F_j} + N_{TR_j} - 1)}. \tag{10}
  \]

For both cases: $N_{x_{F_i}}$ represents the number of functions, that couple with the function $F_i$.

The evaluation of this required coupling-number $N_{x_{F_i}}$ needs concrete ideas of the technical realisation of the elementary functions, because several couplings only may appear at very deep functional levels or even at the technical realisation level. If there is no information and experience about possible technical realisations at abstract levels, worst case must be taken, namely 100 % couplings.

Otherwise, if the elementary functions are determined in detail, the degrees of coupling can be calculate by (11).

\[
\kappa_{F_i} = \frac{N_{x_{EF_i}}}{N_{EF_{ges}} - 1} \tag{11}
\]

$N_{x_{EF_i}}$ ... number of elementary functions that couple with the considered elementary function $EF_i$

$N_{EF_{ges}}$ ... whole number of elementary functions which are part of the considered overall system

The calculation term for the functional degrees of coupling of the upper levels reads

\[
\kappa_{F_i,j} = \frac{1}{N_{EF_{F_i,j}}} \cdot \sum_{k=1}^{N_{F_{k,(j+1)}}} \left( \kappa_{F_{k,(j+1)}} \cdot N_{EF_{k,(j+1)}} \right) \tag{12}
\]
where

\[ j \] \quad \text{considered level of the functional tree}
\[ F_{i,j} \] \quad \text{considered functionality} \ F_i \ \text{at level} \ j
\[ \kappa_{F_{i,j}} \] \quad \text{degree of coupling of the considered function} \ F_i
\[ N_{EF_{i,j}} \] \quad \text{number of elementary functions, which are part of the function} \ F_i
\[ N_{F_{i,(j+1)}} \] \quad \text{number of the son elements of function} \ F_i
\[ \kappa_{F_{k,(j+1)}} \] \quad \text{functional degree of coupling of the direct following functionality}
\[ N_{EF_{k,(j+1)}} \] \quad \text{number of elementary functions, which are part of the direct following function}

Going bottom-up and using (12) the exact degree of coupling can be computed as well as the information content and the product complexity of the overall system even on the top level.

Finally, if the system is well defined and the technical realisations are well known, the technical product complexity can be evaluated to gain further insight.

**Technical Product Complexity (TPCI).** This complexity value considers - in addition to the functional product complexity - the design and product characteristics and reflects the coupling-situation in more detail.

\[
TPCI = \sum_{i=1}^{N_{TR}} I_{\kappa_{TR_i}}
\]  

(13)

At this deep technical level, the dependency between the functional requirements (FR) and the design parameters (DP), respectively the so called special design parameters (SDP) already can be expressed, see (14) and (15). Meaning and characteristics of these special design parameters (SDP) are discussed in [7].

### Functional Dependence

\[
\begin{align*}
FR_1 & \leftarrow \begin{bmatrix}
c^{*1} & c^{*12} & c^{*13} & \cdots & \cdots & c^{*1N_{DP}} \\
c^{*21} & c^{*22} & c^{*23} & \cdots & \cdots & c^{*2N_{DP}} \\
c^{*31} & c^{*32} & \ddots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \ddots & \cdots & \cdots & \cdots \\
c^{*N_{FR}1} & c^{*N_{FR}2} & \cdots & \cdots & c^{*N_{FR}N_{DP}}
\end{bmatrix}
\end{align*}
\]

At this deep technical level, the dependency between the functional requirements (FR) and the design parameters (DP), respectively the so called special design parameters (SDP) already can be expressed, see (14) and (15). Meaning and characteristics of these special design parameters (SDP) are discussed in [7].

### Functional Dependence

\[
\begin{align*}
FR_1 & \leftarrow \begin{bmatrix}
c^{*1} & c^{*12} & c^{*13} & \cdots & \cdots & c^{*1N_{DP}} \\
c^{*21} & c^{*22} & c^{*23} & \cdots & \cdots & c^{*2N_{DP}} \\
c^{*31} & c^{*32} & \ddots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \ddots & \cdots & \cdots & \cdots \\
c^{*N_{FR}1} & c^{*N_{FR}2} & \cdots & \cdots & c^{*N_{FR}N_{DP}}
\end{bmatrix}
\end{align*}
\]

with \( N_{FR} < N_{DP} \), respectively
where \( N_{FR} \geq N_{SDP} \land \{ SDP \} \subseteq \{ DP \} \land \{ C \} \subseteq \{ C^* \} \).

The so called “degree of technical couplings” of the elementary functions calculates to

\[
\kappa_{TR_i} = \frac{1}{N_{FR_{TR_i}}} \sum_{i=1}^{N_{FR_{TR_i}}} K_i
\]

(16)

with

\[
K_i = \frac{S_i}{N_{1_i}}
\]

(17)

\[
S_i = \sum_{j=1}^{N_{SDP}} s_j \text{ with } \begin{cases} s_j = 1 & \text{in case of } c_{ij} \neq 0 \land \exists c_{kj} \neq 0 \text{ and } k \neq i \\ s_j = 0 & \text{else} \end{cases}
\]

(18)

\[
N_{1_i} = \sum_{j=1}^{N_{SDP}} n_{1_j} \text{ with } \begin{cases} n_{1_j} = 1 & \text{in case of } c_{ij} \neq 0 \\ n_{1_j} = 0 & \text{else} \end{cases}
\]

(19)

where

- \( FR_{TR_i} \) ... set of the functional requirements that are part of the technical realisation \( TR_i \)
- \( N_{FR_{TR_i}} \) ... number of functional requirements in \( TR_i \)
- \( N_{SDP} \) ... number of the special design parameters SDP
- \( N_{1_i} \) ... number of entries with the value one in line number \( i \)
- \( S_i \) ... number of lines, which couple with line number \( i \)
- \( K_i \) ... degree of coupling based on the functional requirement number \( i \)

Taking these facts into account, the degree of coupling of the technical realisation reads as:
\[
I_{\kappa_{TRj}} = \log_2 \left( \frac{1}{1 - \kappa_{TRj} \cdot \frac{N_{SDP_i}}{N_{SDP_{ges}}} \cdot (1 - \varepsilon_{I_{100\%}})} \right)
\]  
(20)

where

\(N_{SDP_i}\) is the number of SDP, that are part of the technical realisation and \(N_{SDP_{ges}}\) represents the overall number of SDP.

**Reliability Product Complexity (RPCI).**

\[
RPCI = \sum_{i=1}^{N_{TR}} I_{p_{TRi}}
\]  
(21)

\(I_{p_{TRi}}\) represents the probability \(p_{TRi}\) of the reliability of the functionality of the technical realisation \(TR_i\) and reads as

\[
I_{p_{TRi}} = \log_2 \left( \frac{1}{p_{TRi} \cdot (1 - \varepsilon_{I_{100\%}}) + \varepsilon_{I_{100\%}}} \right)
\]  
(22)

This definition follows directly Suh's proposal for the information content [6].

### 4 Summary

Complexity is not a unique property of a system per se but a matter of human comprehension of it. For the assessment of a design in its various stages of putting it into concrete terms different aspects of complexity are relevant.

Thus, different complexity values for these different stages are meaningful. We propose three of such values: the functional product complexity FPCI, the technical product complexity TPCI, and the reliability product complexity RPCI. All of them are based on a rigorous functional decomposition [7] of systems under consideration.

1. The Functional Product Complexity \(FPCI_j = \sum_{i=1}^{N_{p_{fj}}} I_{\kappa_{fj}} + \sum_{i=1}^{N_{p_{rj}}} I_{\kappa_{rj}}\)

judges on complexity of functional couplings, disregarding the realised technical details.
2. The Technical Product Complexity $TPCI = \sum_{i=1}^{N_{TR}} I_{\kappa_{TRi}}$ reflects the relationship between the functional requirements and the technical design parameters of the considered function or system.

3. The Reliability Product Complexity $RPCI = \sum_{i=1}^{N_{TR}} I_{p_{TRi}}$ represents the probability of designing and producing a faultless product.

Comparing these complexity evaluation method with them from Griffin, Kannapan or Bashir and Thomson, the proposed complexity values are more complicated and require a rigorous functional decomposition, as well as a detailed specification, but yield comprehensible and clear results.

References

A Complexity-Theoretic Approach to the Design of Good Measures of Cryptographic Strength

Juan David González Cobas and José Antonio López Brugos
Departamento de Informática
Universidad de Oviedo
{cobas,brugos}@etsiig.uniovi.es

Abstract. Quality assessment for cryptographic algorithms is usually devised by a combination of statistical or information-theoretic techniques for testing a pseudo-random sequence generator (PRSG for short). Many tests among these share a common framework. The difference among tests usually relies on the choice of the computational device and its suitable definition of “size” that determines the workload needed to match the allegedly random output of a PRSG.

The most outstanding examples among these are compared, showing that there appears to be a hierarchy of tests whose complexity increases with their discriminating power.

The possibility of a tradeoff between computational complexity of the measure and its value in cryptographic strength assessment is explored, and the criteria towards an optimal measure of strength are proposed.

1 Cryptographic Measures of Complexity

The security of an important family of cryptographic algorithms, known as synchronous stream ciphers [10], depends on the pseudorandomness of a (usually binary) sequence generated by a finite automaton with a huge number of states (their key sequence generator or KSG). The output of this KSG is combined with the plaintext stream, and a ciphertext stream is obtained, as shown in Fig. 1.

A usual approach to the analysis of these sequence generators is system-theoretic: the output stream is supposed to have been generated by some automaton chosen from a class $\mathcal{M}$. Automata in this class are usually characterized by a family of parameters (the “program” or “structural properties” thereof), one of which is a dimensional parameter (automaton size).

In the most general case, the KSG should not belong to $\mathcal{M}$. A synthesis algorithm is devised to construct a smallest automaton in $\mathcal{M}$ which generates the output of the KSG, perhaps incrementally. The size of the resulting minimal automaton is labelled the $\mathcal{M}$-complexity of the output of the KSG.

The realm from which the synthesized automaton is chosen gives rise to different measures of complexity, and many are in use in cryptography. Among others, we may cite the following.

© Springer-Verlag Berlin Heidelberg 2001
1.1 Linear Complexity

A linear feedback shift register (LFSR for short) is depicted in Fig. 2. It contains $L$ (usually binary) memory cells, shifted to the right every time a new bit of sequence is required. The feedback is obtained as a linear function of the contents in the previous step. $L$ is the length of the register. The output sequence satisfies a linear recurrence

$$x_L = \sum_{k=0}^{L-1} c_i x_i$$

where $c_i$ is one or zero according to which taps of the register are fed back.

Definition 1. The linear complexity (LC) of a finite sequence $(s_k)_{k=0}^{n-1}$ is the length $L$ of the shortest LFSR generating $s$.

Definition 2. The linear complexity profile of $s$ is the LC of $(s)_{k=0}^{l-1}$ as a function of $l$. 

Fig. 1. Stream cipher operation

Fig. 2. Linear feedback shift register
The algorithm for computing the linear complexity profile of a given finite sequence is the well-known Berlekamp-Massey algorithm \cite{7}. The widespread use of linear complexity profiles for analysis of stream ciphers and sequence generators is due to its many analytical properties. A good account of them can be found in \cite{10}.

Nevertheless, a good LC profile is a necessary condition, but by no means sufficient. It is easy to construct trivial examples of sequences with profiles sticking to all the standard requirements that violate other elementary cryptographic criteria.

1.2 Quadratic Span

If the feedback of a shift register (as shown in Fig. 2) is allowed to be a boolean quadratic polynomial of register taps, we obtain a broader definition of sequence complexity.

**Definition 3.** The quadratic span (QS) of \((s_k)_{k=0}^{n-1}\) is the length of the shortest FSR generating \(s\) if a quadratic feedback is allowed, so the generated sequence satisfies

\[
x_L = c + \sum_{k=0}^{L-1} c_i x_i + \sum_{k=0}^{L-1} \sum_{j=k+1}^{L-1} c_{ij} x_i x_j
\]  

(2)

Quadratic span can be computed by an algorithm of Chan and Games \cite{1}. The computation is much harder and involved than in the Berlekamp-Massey algorithm, because it turns out that allowing quadratic terms makes the generating automaton much more powerful.

Obviously \(\text{QS} \leq \text{LC}\); typically, \(\text{LC}\) is about the square of \(\text{QS}\).

1.3 Maximum Order Complexity

Generalizing previous definitions, we arrive at the idea of maximum-order complexity \cite{3}:

**Definition 4.** The maximum order complexity (MOC) of \((s_k)_{k=0}^{n-1}\) is the length of the shortest FSR generating \(s\) if any boolean feedback function is allowed, so the generated sequence satisfies:

\[
x_L = F(x_0, x_1, \ldots, x_{L-1})
\]

(3)

The properties and cryptographic applications of the MOC of a sequence are thoroughly studied in \cite{4}.

Blumer’s algorithms allow computing the MOC and the associated FSR in time \(O(L^2 \log L)\) and \(O(L)\), respectively. The typical complexity profile grows as twice the logarithm of sequence length.
1.4 Entropy

Ciphertext output of a good cryptographic algorithm should not be compressible. The connection between compression and cryptography dates back to the seminal papers of Shannon [11, 12]. Entropy fits very well the same scheme we observed in complexity measures introduced so far:

Definition 5. The entropy of a sequence \( (s_k)_{k=0}^{n-1} \) over an alphabet \( \Sigma \) can be defined as the length of the compressed sequence obtained from \( s \) by application of Huffman encoding [2].

If a sequence \( s \) is compressed using Huffman encoding, it can be reconstructed from a (constant space) Huffman tree and the compressed sequence. The size of the latter measures the entropy of the sequence. Ideally, a good KSG should generate an incompressible stream, i.e., sequence entropy should equal sequence length.

1.5 Lempel-Ziv Complexity

Definition 6. The Lempel-Ziv complexity of a sequence \( (s_k)_{k=0}^{n-1} \) is the length of the compressed stream that results from applying the Lempel-Ziv compression algorithm [16, 17] to \( s \).

The original sequence \( s \) can be reconstructed from the compressed stream alone (using a constant space dictionary).

1.6 Kolmogoroff Complexity

When the class of sequence-generating automata is taken to be the whole set of Turing machine programs over the alphabet of the sequence, the size of the shortest one that generates the given sequence is said to be its absolute complexity.

Definition 7. Let \( (s)_{k=0}^{n-1} \) a sequence over alphabet \( \{0, 1\}^* \). Its Kolmogoroff complexity is defined to be the length of the shortest binary program (for a given universal Turing machine \( M \)) whose output is the given sequence \( s \).

Kolmogoroff complexity is not a recursive (i.e., computable) function. There is no algorithm to compute it (and there cannot be). This is in sharp contrast with our former complexity measures, all of which could be computed by more or less cumbersome methods.

1.7 Other Measures

There are other functions that do not fit so neatly the scheme we will introduce in Sec. 2. Among them, the following are widely used in cryptography.
1. Maurer’s universal test \( [8] \). It roughly measures the compressibility of a bit stream. It encompasses many of the classical pseudorandomness tests mentioned in \([8]\).

2. “Absolute” definitions of pseudorandomness based on assumptions that some problems like factoring and discrete logarithm are hard (a good survey of which can be found in \([6]\)).

**2 Sequence Complexity Hierarchy**

From the list given in Sec. 1 it is apparent that the choice of increasingly powerful automata classes gives rise to a hierarchy of complexity measures that can be useful in the assessment of cryptographical weaknesses (and strength) of KSGs.

With exceptions (most remarkable of them are the tests of Sec. 1.7), cryptographic complexity measures can be described by means of the following framework:

1. A family \( \mathcal{M} \) of sequence-generating finite automata over an alphabet \( \Sigma \) is chosen.
2. A “size measure” \( s : \mathcal{M} \rightarrow \mathbb{N} \) is defined.
3. A complexity measure \( c(\cdot) \) for sequences over an alphabet is given by the function \( c : \Sigma^* \rightarrow \mathbb{N} \) with
   \[
   c(x) = \min\{s(M) | M \in \mathcal{M} \text{ and output}(M) = x\} \tag{4}
   \]
4. A synthesis algorithm is (hopefully) given for finding some \( M \in \mathcal{M} \) matching that minimum.

It is a well-known fact that KSGs with good linear complexity profiles can actually be very weak cryptographically, failing miserably as far as other measures are concerned. Nevertheless, “simple” measures like linear complexity are still appealing because of their ease of use and nice theoretical properties.

It is to be expected that a wider class \( \mathcal{M} \) gives rise to a harder synthesis algorithm. This turns out to be the case for the complexity measures studied in Sec 1. A synoptical display of the parameters involved is given in Table 1. For a given complexity measure fitting the description above, the most simple-minded synthesis algorithm implies exhaustively searching all automata with bounded size until matching the sequence whose synthesis is demanded. The complexity of the search grows with the size of \( \mathcal{M} \).

Of course, this brute force approach never happens with the well-known measures mentioned before. A good example of this is the Berlekamp-Massey algorithm, in which the search is guided by a very precise heuristic (the next symbol in the sequence is made to match the output of the next LFSR synthesized by the algorithm in a deterministic way). This becomes harder as the class of automata involved becomes bigger and with less precise analytical properties. The extreme case is Kolmogoroff complexity, where even brute force search becomes
Table 1. Comparison of complexity measures

<table>
<thead>
<tr>
<th>Complexity measure</th>
<th>Automaton</th>
<th>Size</th>
<th>Algorithm</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC</td>
<td>LFSR</td>
<td>Length</td>
<td>Berlekamp--Massey</td>
<td>$L/2$</td>
</tr>
<tr>
<td>QS</td>
<td>Quadratic FSR</td>
<td>Length</td>
<td>Chan–Games</td>
<td>$\sqrt{L}$</td>
</tr>
<tr>
<td>MOC</td>
<td>Nonlinear FSR</td>
<td>Length</td>
<td>Blumer DAWG</td>
<td>$2\log L$</td>
</tr>
<tr>
<td>Entropy</td>
<td>Huffman tree</td>
<td>Length of compressed stream</td>
<td>Huffman coding</td>
<td>$L$</td>
</tr>
<tr>
<td>Kolmogoroff</td>
<td>Turing machine</td>
<td>Length of program</td>
<td>Non-recursive</td>
<td>$L$</td>
</tr>
</tbody>
</table>

infeasible, as no matching strategy neither prediction of the output is possible, even in principle.

On the other hand, the wider class $\mathcal{M}$ is chosen, the more discriminating power we get, and more powerful criteria for non-randomness are obtained.

3 Optimal Cryptographic Measures of Complexity

Quality assessment for cryptographic pseudorandom sequences is a very difficult task. The statement just proved shows that there must be some intrinsic difficulty to it. The ultimate test (i.e., pseudorandomness in the sense of [15]) is known to be passed successfully only by KSGs that are too slow for many practical purposes. This is a natural consequence of the stringent requirement that the sequence be indistinguishable from a truly random one by any polynomial-time test.

The question remains open: Is there some way to construct measures of complexity more general than the ones described in Sec. 1, but still easy enough to compute? We point to some possible methods of devising such a construction, to wit:

1. synthesis of cellular automata
2. other chaotic mappings
3. synthesis of DFA with some probability of error

We take a closer look at the first one.

4 Cellular Automata as PRSG

Proposals have been made [13] of using cellular automata as KSG. For the purpose, a cellular automaton (CA for short) is an array of cells storing the state of
the machine. Every time a transition is called for, every cell is updated with to the value of a nonlinear boolean function whose arguments are the contents of that same cell and its neighbours's, as depicted in Fig. 3. The function is called transition function or transition rule of the CA.

![Cellular automaton transition](image)

\[ a'_i = a_{i-1} \oplus (a_i \lor a_{i+1}) \]

**Fig. 3.** Cellular automaton transition

The transition rule is applied synchronously, i.e., every cell is updated according to the value of the transition function with its arguments evaluated prior to changing state.

It is known that very large numbers of cells, on the order of several thousands, have to be used to provide security [9]. This means that CA are quite weak devices for PRSG purposes, and that synthesis of a CA generating some given sequence could be possible.

Two approaches are considered here:

1. Finding a suitable transition rule that provides a given sequence as output.
2. Finding a suitable initial state for a fixed transition rule that results in generation of the given sequence.

Surprisingly enough, the second alternative is possible in some (bidimensional) CA with very simple transition rules but a huge number of cells. The reason is that bidimensional CA can be found which simulate a universal Turing machine [14], so any recursive sequence will be synthesized by properly choosing the initial state. Nevertheless, the problem of determining such state is non-recursive, which makes this approach of no practical value.

On the other hand, the first alternative seems intractable as well. Not every sequence can be so generated. Experiments were performed to synthesize simple (LFSR-generated) sequences with CA of short register lengths (up to 20). The algorithm for determination of the transition rule was exhaustive search of boolean functions of increasing number of arguments (up to five, so as to explore every one of the possible $2^{32}$ outcomes). Although results are inconclusive, the synthesis becomes infeasible very soon with the only increase of length of the LFSR generator, being the time spent on it roughly exponential.
Except the obvious strategies for shortening exhaustive search, the question of a guiding heuristic (such as the one found for LFSR synthesis) remains open. Of course, no such heuristic is expected to be found when the neighbours of a cell involved in transitions of a CA are fixed from the start. Some kind of fine-tuning of this number of neighbours is required, but no single way of performing this bumping of the transition function length has been found that suits the synthesis process.

5 Closing Remarks

The analysis of KSG via measures of complexity is a well-established technique in cryptography. The concept of complexity differs from the usual one in complexity theory: it is given by the size of a program, not the time or space it takes to work.

More discriminating measures of complexity are harder to compute. Although an optimal class of automata for pseudorandomness testing is unlikely to exist, finding a more general kind of finite-state device with a tractable synthesis problem is an interesting research problem. Cellular automata (with a suitable definition of complexity parameters) could be useful for this purpose, but further refinement of their families of parameters (restricting their transition rules) seems necessary to obtain a feasible synthesis algorithm.

References

Characterizing the Software Development Process: A New Approach Based on Kolmogorov Complexity

Carlos A.P. Campani\textsuperscript{1} and Paulo Blauth Menezes\textsuperscript{2}

\textsuperscript{1} IFM/ UFPel - CP 354, 96010-900, Pelotas/RS, Brazil
campani@ufpel.tche.br

\textsuperscript{2} II/ UFRGS - CP 15064, 91501-970, Porto Alegre/RS, Brazil
blauth@inf.ufrgs.br

Abstract. Our main aim is to propose a new characterization for the software development process. We suggest that software development methodology has some limits. These limits are a clue that software development process is more subjective and empirical than objective and formal. We use Kolmogorov complexity to develop the formal argument and to outline the informal conclusions. Kolmogorov complexity is based on the size in bits of the smallest effective description of an object and is a suitable quantitative measure of the object’s information content. We try to show that notion of complexity is a suitable measure and a tool for the characterization of the software development process. Following the paper conclusions, the limits of formal methods typifies the software development process as experimental and heuristical based, like, for example, the scientific development in physics and chemistry. Moreover, by our approach, we argue that software development is, in some sense, formally unpredictable. These conclusions suggest that software engineering is a scientific field not totally characterized by the typical work of engineering, but also by the experimental sciences methodology.

1 Introduction

One of the main goals of software development methodologies is to obtain optimum code implementation and correct programs by the easiest way and at low cost. The formal methodology places software development on a mathematical basis \cite{10}. On the other hand, software engineering methods and metrics are important to estimate efforts and costs of the software development process \cite{12,20,21}. They are very important because force the programmer to think about the software under development, practicing a better programming discipline and a more efficient project management. These methodologies make use of methods and models to obtain better programs at low cost of development much like the solution of problems in engineering.

\textsuperscript{*} This work is partially supported by: FAPERGS (Projects QaP-For, \Omega-\pi/UFPel) and CNPq (Projects HoVer-CAM, GRAPHIT, MEFIA) in Brazil.
In this paper, we discuss the limits of software development in many aspects: predictability of software development; formal specification and semantics; correctness of programs; optimization of programs regarding the code size; and choice of the best programming language to the specific target application. Based on these results, we are also proposing a new characterization of the software development process. We attempt to outline our point of view using *Kolmogorov complexity* as a tool to develop the formal argument and base our informal conclusions.

The idea of Kolmogorov complexity came from the concept of entropy [3,8]. Kolmogorov complexity is based on the size in bits of the smallest effective (computable) description of an object and is a suitable quantitative measure of the object’s information content [2,3,8,9,14,15]. It uses the universal Turing machine as specification method, achieving a measure independent of the specification method, up to a constant (invariance theorem). As a consequence of this, the complexity of an object is an intrinsic attribute of the object itself independently of a particular specification method.

In other words, descriptions are computer programs and the objects are interpreted as the outputs obtained running the program, or the problem solved by it. Kolmogorov complexity is simply the length of the smallest computer program which computes a given output.

A result shown in this paper is that Kolmogorov complexity is not computable, but semi-computable, and its estimate is an intractable problem. It suggests that the problem to achieve the best program implementation regarding the code size is very difficult.

Following the argument, we can show that formal methodologies have some limits, characterizing the formal methods as incomplete. Finally, we suggest these limits are a clue for the characterization of the software development process.

## 2 Kolmogorov Complexity

A typical problem in computer science is the evaluation of the complexity, that is, the evaluation of the computational resources needed to solve a given problem. The concept of complexity belongs to both categories: dynamic complexity and static complexity.

The dynamic complexity is related with the time execution complexity and with the space complexity [11]. The static complexity, on the other hand, is related with the quantitative measure of the information content of an object. It’s an important concept for computer science because, instead of deal with the time or space needed to run a given program, deals with the complexity of the object itself [16]. This new approach suggests that the phenomenon of computation isn’t only logical but also statistical, in a stochastic sense.

We accept that exists a specification method \( f \) that associates at least one object \( x \) with one description \( y \). Let \( X \) be the set of objects and \( Y \) the set of descriptions, then \( f(y) = x \), where \( x \in X \) and \( y \in Y \).
We represent the descriptions as binary strings. There is an enumeration of
the binary strings by the lexicographical order
\[
\left( \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\Lambda & 0 & 1 & 00 & 01 & 10 & 11 & 000 & 001 \\
\end{array} \right),
\]
(1)
where \( \Lambda \) is the empty string. The natural number \( s \) represents the \( s \)-th binary
string. If \(|.|\) denotes the string length, then \(|s| = \lfloor \log(s+1) \rfloor \), where \( \lfloor . \rfloor \) represents
the biggest integer number smaller or equal than a number and \( \log \) represents
the base two logarithm.

From now on, in this paper, we will not distinguish between binary strings and
natural numbers. Hence, we can define the “size” of an integer number. Moreover,
we can define functions over naturals \( \phi : \mathbb{N} \to \mathbb{N} \) as functions over binary strings
\( \phi : \{0,1\}^* \to \{0,1\}^* \). Then, we can represent objects and descriptions as both
natural numbers or binary strings.

We define the complexity \( C_f(.) \) over binary strings (or natural numbers),
relating to some partial function \( f \) (specification method), as
\[
C_f(x) = \min_{f(y)=x} |y|,
\]
(2)
where \( x \) is an object and \( y \) is a description. Hence, this complexity is defined
over all the set of partial functions. But, in this way, it wouldn’t be an objective
notion because the complexity of an object would depend on the specification
method adopted.

Kolmogorov complexity, on the other hand, uses the Turing machine as spe-
cification method (partial recursive functions). We consider a Turing machine \( M \)
which, with a given binary string \( p \) and natural number \( y \), computes the output
\( M_{p,y} = x \). We say that \( M \) interprets \( p \) as a description of \( x \) in the presence of
the side information \( y \).

We want to show how to construct a concrete computer to deal with our
definition of complexity. The machine has two tapes. The first tape is called
program tape and is an one-way finite read-only tape. The second tape is called
work tape and is a two-way infinite read/write tape. Initially, the first tape stores
the description \( p \) and the second stores the side information \( y \) literally. All other
fields on the work tape are filled with blanks. The machine has two heads, one
per tape. The machine can read or write a symbol (0 or 1), move the head left or
right, or delete symbols (all these operations are defined on the work tape). After
a finite amount of time the machine eventually halts with the output stored on
the work tape.

**Definition 1.** The conditional complexity \( C_M(x|y) \) of a number \( x \) with respect
to a given number \( y \) is the size of the smallest description \( p \) in such a way that
\( M_{p,y} = x \),
\[
C_M(x|y) = \min_{M_{p,y} = x} |p|.
\]
(3)
If doesn’t exist a description \( p \) of \( x \) then we say, by definition, \( C_M(x|y) = \infty \).
In a first view, seems that the conditional complexity \( C_M(\cdot|\cdot) \) remains dependent on the machine \( M \). Kolmogorov and Solomonoff observed that depends much few, up to a constant, because there are universal Turing machines, capable of simulating any other Turing machine whose description is supplied (for a proof of the existence of universal Turing machines see [11]). This is the important invariance theorem \([2,14,15,16]\).

Let \( M_0, M_1, M_2, \ldots \) be an enumeration of the Turing machines (see [16]). We can do it because the set of programs is enumerable. Let \( p_0, p_1, p_2, \ldots \) be an enumeration of the set of programs. Each machine \( M_i \) is defined by an “internal” program \( p_i \). We will call it the “hardware program”.

**Theorem 1.** (invariance theorem) There is a machine \( U \), called universal, in such a way that for any machine \( M \) and numbers \( x \) and \( y \), \( C_U(x|y) \leq C_M(x|y) + c \), where \( c \) depends only on \( M \).

**Proof:** For \( C_M(x|y) = \infty \) the proposition is trivially truthful. We can show an universal Turing machine simulating any other Turing machine. For example, suppose the machine \( U \) in such a way that \( U_{n0p,y} = M_{ip,y} \), where \( M_i \) is the \( i \)-th machine in the Turing machines enumeration. Suppose \( M \) is the \( n \)-th machine in the enumeration, that is, \( M_n = M \), then

\[
C_M(x|y) = \min_{M_{p,y} = x} |p| \tag{4}
\]

and

\[
C_U(x|y) = \min_{U_{n0p,y} = x} |1^n 0p| = \min_{U_{n0p,y} = x} |p| + n + 1 = C_M(x|y) + n + 1. \tag{5}
\]

That is, the upper limit of the complexity expressed with \( U \) is \( C_M(x|y) + n + 1 \). Then,

\[
C_U(x|y) \leq C_M(x|y) + n + 1. \tag{6}
\]

Taking \( c = n + 1 \) completes de proof, with \( c \) depending on \( M \).

Let \( f \) and \( g \) be two partial functions. If \( C_f(x|y) \leq C_g(x|y) + c \), we say that \( f \) minorizes \( g \). If \( C_f(x|y) \leq C_g(x|y) + c \), for every \( g \) in a subset of the partial functions, we say that \( f \) is an universal element of the subset. The invariance theorem proves that there exists an universal element in the set of the partial recursive functions (the universal partial recursive function).

Restricting the descriptions only to the effective (computable) descriptions permits that exists an universal specification method that obtains the minimal-size description with respect to all other methods (minorizes all other specification methods). As a consequence of this, the complexity of an object is an intrinsic attribute of the object itself independently of a particular description method \([16]\).

From theorem \( \text{1} \) we can trivially prove the corollary \( \text{1} \).

**Corollary 1.** (equivalence of additively optimum methods) Let \( U \) and \( V \) be two universal Turing machines. Then,

\[
|C_U(x|y) - C_V(x|y)| \leq c, \tag{7}
\]

where \( |.| \) denotes the absolute value of an integer.
That is, two optimum specification methods, defining the minimal description of an object, differ only by a constant. It’s true because both $\mathcal{U}$ and $\mathcal{V}$ simulate each other [16].

Fixing an universal Turing machine $\mathcal{U}$, called reference machine, we can say $C(x|y) = C_{\mathcal{U}}(x|y)$, called conditional complexity. If isn’t supplied side information, we can define the unconditional complexity of the number $x$

$$C(x) = C(x|\Lambda),$$

called plain complexity.

In few words, the plain complexity $C(x)$ of a number or binary string $x$ is defined as the size, expressed in bits, of the smallest program for the universal Turing machine $\mathcal{U}$ that computes $x$.

**Theorem 2.** There is a constant $c$ in such a way that $C(x) \leq |x| + c$ for all binary strings $x$.

proof: There is a machine $\mathcal{M}$ in such a way that $\mathcal{M}_p = p$ for all programs $p$. Then, for all binary strings $x$, applying the theorem 1, $C(x) \leq C_{\mathcal{M}}(x) + c = |x| + c$.

It’s true because exists a machine that performs the copy of the program itself to the output [2].

Kolmogorov complexity defines a new kind of information theory called algorithmic information theory [5,8,9,16]. In this paper, we use algorithmic information theory to show some limits of the software development process and to characterize it.

3 About Program Length

We advocate that program length is an important parameter for software development, because has influence on software project management. The use of Kolmogorov complexity shows that program length is related with the measure of the complexity of the problem solved by the program, independently of other factors like the programming language or programmers ability. The choice of the minimal program length, among other measures, is suitable to define the complexity and the minimal program length define an universal program complexity measure.

We want to show that our notion of the minimal program length is a suitable measure of the object’s complexity and a tool for the characterization of software development. We will discuss a few arguments that may be presented against our notion of complexity. Finally, we will show that Kolmogorov complexity isn’t computable and its estimate is an intractable problem.

3.1 The Reference Machine

Someone can argue that complexity depends on the reference machine $\mathcal{U}$. Let $\mathcal{V}$ be an universal machine not equal our reference machine, then (by corollary 1)

$$C(x) \leq C_{\mathcal{V}}(x) + \Delta,$$
where $\Delta$ is a measure error dependent on $V$. It’s importante to note that $\Delta$ is independent of $x$. Hence, we can rewrite $\Delta$ as a constant $c$ (or $cV$).

This case is very similar with the calculus. In the calculus, when you integrate a function you obtain a new function and an uncertainty expressed as a constant. Many times the solution of differential equations are functions with constants, defining an infinite number of level sets. In this case, more than defining the exact solution, we are interested on some mathematical properties of the solution. In our case, we are interested on the behaviour of the complexity asymptotically.

It isn’t always true that $C_U(x) < C_M(x)$. But, the meaning of the invariance theorem is that we can’t improve the complexity expressed on $M$ more than the complexity expressed on $U$, up to a constant $[16]$. Thus, the complexity expressed on the reference machine is an intrinsic attribute of the object. We call $U$ an universal specification method, because it minorizes all other methods in the set of specification methods.

### 3.2 Universal Method as an Objective Notion

Someone can argue that the universal method used in the proof of the invariance theorem isn’t an objective notion, because we can choose, for every string $x_0$, a machine $M_0$ in such a way that $C_{M_0}(x_0) = 0$.

A mathematical solution for this problem is to define complexity classes $[16]$. Two complexities $C_{M_0}$ e $C_{M_1}$ are equivalent, $C_{M_0} \equiv C_{M_1}$, if there is a constant $c$ in such a way that for every $x$

$$|C_{M_0}(x) - C_{M_1}(x)| \leq c$$

then, the relation $\equiv$ induces an equivalence class over the complexities set

$$[C_{M_0}] = \{C_M : C_M \equiv C_{M_0}\} .$$

We can prove it very easy, because

**Reflexive** For all $x$ and $M$, there is a $c$ in such a way that $|C_M(x) - C_M(x)| \leq c$ holds trivially $(|a - a| = 0)$;

**Symmetrical** If $|C_{M_0}(x) - C_{M_1}(x)| \leq c$ then $|C_{M_1}(x) - C_{M_0}(x)| \leq c$ ($|a - b| = |b - a|$);

**Transitive** If $|C_{M_0}(x) - C_{M_1}(x)| \leq c$ and $|C_{M_1}(x) - C_{M_2}(x)| \leq c'$ then $|C_{M_0}(x) - C_{M_2}(x)| \leq c''$, with $c'' = c + c'$ (by the triangle inequality $|a+b| \leq |a| + |b|$).

We can order these classes, $[C_{M_0}] \leq [C_{M_1}]$ if and only if, for every $x$, we have $C_{M_0}(x) \leq C_{M_1}(x)$. Thus, the set of complexity classes has a minimal element $[C_U]$ in such a way that for every $C_M$

$$[C_U] \leq [C_M] .$$

This minimal element exists because the complexity classes are disjoined sets and relation $\leq$, over the set of complexities, is a total order. We can use this minimal element as our reference to measure the complexity.


3.3 Acceptable Numberings

Another possible objection to our argument is that our definition of complexity depends on an arbitrarily chosen enumeration of the Turing machines.

Let \( f_0, f_1, f_2, \ldots \) and \( g_0, g_1, g_2, \ldots \) be two enumerations of the partial recursive functions, where the first is our plain enumeration. We can assert that \( f_i = g_{\theta(i)} \) and \( g_i = f_{\psi(i)} \), for \( i \geq 0 \). If both \( \theta \) e \( \psi \) are recursive functions, then the enumerations are recursively isomorphic and both are acceptable numberings (Gödel numbers) [16,19].

Suppose two enumerations recursively isomorphic. Let \( C(x) \) be the complexity measured using the first enumeration and \( C'(x) \) the complexity using the second. Then,

\[
|C(x) - C'(x)| \leq c,
\]

where \( c \) doesn’t depend on \( x \).

3.4 Choosing the Programming Language

We want to make clear that Kolmogorov complexity is a characterization of the object not matter what strategies or tools are selected by the programmer to implement the software.

It is true, related to the choice of the programming language, because the equivalence of additively optimum methods. For example, suppose we want to implement a program both in PROLOG and LISP [16]. We can assert that

\[
|C_{\text{PROLOG}}(x) - C_{\text{LISP}}(x)| \leq c,
\]

where \( c \) is independent of \( x \) (the problem solved by the program).

It means that Kolmogorov complexity is independent of the programming language.

3.5 Dealing with Modules

The use of pieces of program interconnected and reusables (program units, modules and components), a modern approach to software development, as a way to avoid complexity is another objection to Kolmogorov complexity. Someone can say that it is more important focusing our attention on the complexity of software interconnections. But, we answer saying that complexity is independent of these aspects because we can show that prefix complexity (complexity based on prefix universal Turing machines) is sub-additive. It means that the complexity of the whole problem is the same of the sum of the complexities of all pieces (sub-additively) [3,5,16]. Equality holds if the pieces are totally independent regarding the information content.

Suppose that if \( M_p \) is defined then \( M_q \) isn’t defined for all prefixes \( q \) of \( p \). We can obtain it by using prefix-free coding and we call these kind of programs auto-delimited programs [3,6,8,16]. If \( M \) receives as input an auto-delimited program we call \( M \) a prefix Turing machine.
An important property of the prefix-free coding is that if you get the concatenation of two strings, represented as a prefix-free code, you can recover both strings \[8,16\].

**Definition 2.** The prefix complexity \[K_M(x)\] is the size in bits of the smallest prefix-free description \(p\), in such a way that \(M_p = x\), where \(M\) is a prefix Turing machine.

This is the unconditional prefix complexity. Like with the plain complexity, we define a conditional prefix complexity \(K(x | y)\) as the size in bits of the smallest prefix-free description of \(y\) computes \(x\) in the universal prefix machine \(U\)

\[K(x | y) = \min_{U_p, y^* = x} |p|, \quad (15)\]

where \(p\) is a prefix-free description and \(y^*\) is the smallest prefix-free description of \(y\) \([3]\).

Define \(K(x, y)\), called associated complexity, as the size of the smallest autodelimited program that computes both \(x\) and \(y\) \([3,5]\). If \(p\) is the smallest prefix-free description of \(x\) and \(q\) is the smallest prefix-free description of \(y\) then \(pq\) or \(qp\) is the smallest description of \(x\) and \(y\). There is a prefix Turing machine \(M\) that receiving \(pq\) or \(qp\) recover both \(p\) and \(q\) (by the property previously mentioned) and compute \(x\) and \(y\). Hence, \(K_M(x, y) = |p| + |q|\). Applying the invariance theorem,

\[K(x, y) \leq K_M(x, y) + c = K(x) + K(y) + c. \quad (16)\]

By induction, we can generalize it, asserting that \(K(x_1, x_2, \ldots, x_n) \leq K(x_1) + K(x_2) + \cdots + K(x_n) + c\), and we showed our point. The prefix complexity is sub-additive and the complexity of the whole problem is the sum (sub-additively) of the complexity of its pieces.

If \(x\) and \(y\) are independent regarding the information content then \(K(x, y) = K(x) + K(y) + c\), because \(p\) can’t use what it knows about \(y\), using \(q\), to compute \(x\) more efficiently. In this case, \(K(x | y) = K(x)\). On the other hand, if \(x\) and \(y\) are dependent then \(K(x | y) \leq K(x)\).

It means that focusing the problem on the complexity of software interconnections doesn’t change the conclusions obtained with Kolmogorov complexity.

### 3.6 Computing Kolmogorov Complexity

If we think about \(C\) as a function over naturals, we may want to compute \(C(\_\_\_)\). But, unfortunately it’s a very hard thing to do \([9,16]\).

**Theorem 3.** The function \(C\) isn’t computable.

*proof: By the halting problem, we can’t decide if a program halts or not.*
These are the bad news. The good news are that \( C \) is semi-computable.

A function \( f : \mathbb{N} \to \mathbb{N} \) is semi-computable if there exists a recursive function \( \phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \), with \( \phi(t,x) \) monotonic on \( t \), in such a way that \( \lim_{t \to \infty} \phi(t,x) = f(x) \).

Let \( p \) be a program and \( t \) a natural number, then \( \overline{U}_p^t \) define the predicate that is true if the computation of \( t \) steps of \( p \) in \( U \) halts and computes the value \( \Phi_p \) and false otherwise with \( \Phi_p \) undefined. Using the Turing-Church thesis is easy to show that \( \overline{U}_p^t \) is decidible and \( \Phi_p \) is computable [9].

**Corollary 2.** (bounded halting problem) \( \overline{U}_p^t \) is decidible and \( \Phi_p \) is computable.

**Theorem 4.** The function \( C \) is semi-computable.

**proof:** We know that \( C(x) < \log(x) + c \), because a natural number may be represented by its binary code. Let \( U_p^t \) be the universal Turing machine which computes \( t \) steps of the program \( p \) simulating \( U \). Let \( C^t(x) \) be a value smaller than \( \log(x) + c \) and

\[
C^t(x) = \min\{|p| \leq t : U_p^t = \text{true}, \Phi_p = x\}.
\]

Then, the function \( C^t \) is computable, monotonic on \( t \) and \( \lim_{t \to \infty} C^t(x) = C(x) \).

By theorem [4] there is a total recursive function \( \phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \), in such a way that \( \lim_{t \to \infty} \phi(t,x) = C(x) \). This result means that, although Kolmogorov complexity isn’t computable, we can estimate it.

But, we have another problem. The estimate of \( C(\cdot) \) is an intractable problem (for intractable problems see [11]).

Suppose an enumeration of programs that sorts the set of programs lexicographically. Let \( \mathcal{M} \) be a Turing machine and \( \pi \) a program. Then, the computation of \( \mathcal{M}_\pi \) searches the enumeration for the first program \( p \) that computes \( x \). The algorithm must generate random strings and test a string for a valid program that is a minimal description of \( x \). We know that \( \mathcal{M}_\pi \) must search a set with cardinality \( 2^n \), where \( n \) is the size of the strings generated. At the present time, we don’t know other algorithm more efficient. The problem with this algorithm is to decide if a given program halts or not (halting problem).

Thus, the computational effort to find the smallest description of an object is enormous and impracticable. It suggests that the problem to achieve the best program implementation regarding the code size is very difficult.

We can imagine a contest. A large group of programmers are invited to code a minimal-size program to solve some given problem. After, we pick up the best solution, the winner. How much confident are we to declare the winner’s size code the minimal possible solution? Is it possible to implement a smaller program? The answer is “we can’t decide if a given program is the smallest program that solve some given problem”. We can estimate the minimal code size, but with enormous computational efforts.
4 Incompleteness of Formal Methods

Following our approach, we argue that formal methods have the property of incompleteness in the same sense that Gödel proved in 1931 the incompleteness of arithmetics. This conclusion is based on the work of Greg Chaitin about the incompleteness of formal systems. Chaitin constructed an information-theoretic version of Gödel’s proof, showing that a theorem with more Kolmogorov complexity than the system’s axioms can’t be proved (Chaitin’s theorem). It happens because there are proofs, in the formal system, with more complexity than the formal system itself. The proof of Chaitin’s theorem is based on the classical Berry’s paradox.

Gödel’s proof deals with the problem of the mechanization of proofs in arithmetics. He worked on an axiomatic system for the arithmetics based on the Peano’s Postulates.

Define a formal system as the pair \((F, a)\), where \(F\) is a set of inference rules and \(a\) is a set of binary coded axioms. We can construct a Turing machine \(M\) that receiving as input \(a\) and an integer \(t\) computes in \(t\) steps the set of propositions \(F^t(a)\). Hence, \(F(a) = \bigcup_t F^t(a)\) is the whole set of theorems that we can prove in the formal system [2].

Theorem 5. (Chaitin’s theorem) Consider a formal system \((F, a)\), where \(a\) is a set of axioms. Suppose that a proposition like “\(C(x) ≥ n\)” is in \(F(a)\) only if \(C(x) ≥ n\) is true. Then a proposition like “\(C(x) ≥ n\)” is in \(F(a)\) only if \(n ≤ |a| + c\), where \(c\) is a constant that depends only on \(F\).

\[C(s_k) = C(M_{0^k1a}) ≤ C_M(0^k1a) + c'' ≤ |0^k1a| + c'' = |a| + k + c'' = 18\]

\[= |a| + k + c'' + 1 = |a| + k + c' = 19\]

Taking \(k = c'\) we conclude from lemma 7 that the string \(s_{c'}\) with the smallest proof in \((F, a)\) with complexity greater than \(|a| + 2c'\) has, in fact, complexity fewer or equal than \(|a| + 2c'\), which is impossible. Then, \(s_k\) doesn’t exist for \(k = c'\). Not any string can be proved in \((F, a)\) with complexity greater than \(|a| + 2c'\). Taking \(c = 2c'\) we complete the proof.

If we think the formal verification process as a mechanical theorem proving system designed to prove correctness of programs, the problem appears. Such system is, in fact, a formal system and has the same limits. Everybody with experience in program development knows that verification of the correctness of
programs is a hard work. But, practical limitations are overwhelmed by theoretical limitations. Chaitin’s theorem shows that formal systems have limits relating to the complexity of the set of axioms. Hence, in our view, formal verification can only get evidence about correctness.

Let \((F, a)\) be a formal system designed to prove correctness of programs and let \(\pi\) be a program that implements the formal system. We assert that there is always a program \(p_0\), which is correct, but \(\pi\) can’t prove the correctness of \(p_0\). Hence, \(p_0 \notin F(a)\) or \(M_\pi(p_0)\) doesn’t halt neither at the halt true (accepting state) nor at the halt false (rejecting state).

Moreover, this criticism appears as well in the formal synthesis of programs, by the same reason. We argue that formal synthesis has the same kind of limits outlined in our argument.

On the other hand, programming diversity (or n-version programming) is an example of a successful tool to get more reliable software. In this approach, software quality is ensured by the experience of a large team of programmers working independently. It tries to get fault tolerance by the redundancy of independent parts. It suggests that the human experience is very important for software development.

5 Software Development Predictability

Other important aspect of software development is predictability. Software development predictability is the ability to anticipate the time of development, efforts and costs of the software development. Many authors argue about the importance of predictability as a metric to organize projects \([12, 20, 21]\). These authors assert that the length of the software source code is a fundamental parameter to estimate these efforts.

There isn’t a convention to measure the unit of work to be done in the software design and implementation. The most common method is to estimate the number of lines of code or the number of instructions to be written. But, then appear some problems: Must comments be counted? If exist more than one instruction per line, how do we count it? Moreover, different programmers code the same program with different lengths and with different productivities.

The average time programmers spend to write programs may be estimated by the average productivity of the programmers team. We can estimate the total number of lines of code and use it to estimate the necessary resources to obtain the software finished in a given period of time. After the end of the development process, we can calculate the productivity based on the original time estimate and the time to carry out the work.

Counting program length with bits, bytes or lines of code are equivalent choices, and represent simply different scales of measure.

We are proposing the use of Kolmogorov complexity as a new metric for software development, avoiding the related problems, because Kolmogorov complexity is an attribute of the object (problem) independent of the specification method (programmer, programming language, etc.).
If we accept Kolmogorov complexity as a suitable metric, we realize a problem relating with the non-computability and intractability of the complexity.

There is scientific research to estimate Kolmogorov complexity by many methods, for example genetic programming. Conte et alii. [7] show a method to estimate Kolmogorov complexity by a population of LISP programs which are selected, along generations, by the least size code criterion. In the implementation was used the LISP developed by Chaitin [6], because this interpreter has facilities to deal with undefined computations. The number of programs capable of to compute the string $x$ are all programs with length not significantly bigger than $|x|$. The approach followed is to fix a string $x$ and use a genetic algorithm to do a search on the space of programs for one with optimum length from an initial set of programs generated randomly. Each new generation is obtained from the previous by mutation. Good results are obtained with no more than 100 generations.

But, these kind of methods have limits when applied to complex, real-world problems. These difficulties are implied by the intractability of the complexity. We suggest that software development process is, in this sense, formally unpredictable.

Some authors value empirical and subjective estimates of efforts and costs, based on statistical methods. Höst & Wohlin [12] argue that the estimate of efforts applied on software development is the most important task to estimate costs and the time of development. They say that we can estimate subjectively the efforts to do some task. This estimate is made by interviews with many experienced programmers. The answers are analyzed by statistical methods using some criterions to achieve reliability. There is empirical evidence of the correctness of these kind of estimates.

Finally, exists a proof that the Kolmogorov complexities, measured by different universal specification methods, result in the same value up to a constant (equivalence of additively optimum methods - see corollary [1]). It suggests that the minimal code size of different implementations, in different programming languages, are equal up to a constant and all formalisms have the same limitations outlined in our main argument. It also includes all formalisms of formal specification.

6 Software Development: Art, Engineering, or Science?

According to the Webster’s Dictionary

art 1. human ability to make things; 2. skill; craftsmanship 3. any specific skill or its application . . .

engineering 1. a) the science concerned with putting scientific knowledge to practical uses, divided into different branches, as civil, electrical, mechanical or chemical engineering b) the planning, designing, construction, or management of machinery, roads, bridges, buildings, waterways, etc. 2) the act of maneuvering or managing.
science 1. the state or fact of knowing; knowledge 2. systematized knowledge derived from observation, study, and experimentation carried on in order to determine the nature or principles of what is being studied 3. a branch of knowledge or study, esp. one concerned with establishing and systematizing facts, principles, and methods, as by experiments and hypotheses . . .

Quoting Aho & Ullman [1], computer science is a science of abstraction, because it deals with the mechanization of abstraction. An important part of the field deals with how to make programming easier and software more reliable. Every other science deals with the universe as it is. Computer scientists, on the other hand, must create abstractions of real-world problems and manipulate it inside the computer. The authors argue that the process of abstraction, in computer science, is a process of finding approximate, simplified, models of the real-world problems. Often, finding a good model can be difficult because the fundamental limitations on the tasks computers can perform.

Someone can argue that software development is an art. According the definition of art, in this case, software development would be based on the programmers skills, instead of systematized knowledge. But, everybody agree that there is a lot of knowledge on the software development activity.

On the other hand, McCarthy [17] asserts computer science is the science of how machines can be made to carry out intellectual processes. Moreover, the basic results of theory of computability include the existence of limits on what tasks computers can perform. The author considers it to be the theory which attempts to formalize our understanding of computation, and a particular to make the art of verifying computer programs into a science.

According [18] software development should be an engineering, because it is the application of scientific principles toward practical ends. But, according the author, software development is a different kind of engineering, because software is so labor intensive that a significant amount of engineering energy must be focused on project goals in addition to product goals.

We divide, by didactic reasons, computer science into two categories. The first is the software development, systems design and programming, in other words, the mechanization of abstraction as pointed by Aho & Ullman. This category includes all applications of computers in the solution of real-world problems, like artificial intelligence, database design, etc. The second is the theory of computing, complexity theory, automata theory, recursive function theory and the foundations of computer science, which deals with the phenomenon of computation as the object of study, as pointed by McCarthy.

Computer science is, in some sense, an engineering, when we are concerned with the first category. And, on the other hand, it’s a science when we are concerned with the last. In this paper we are more interested in the first, relating with the software development process.

The models and methods used on software development must be effective, but we pointed along the paper some limits to the effectiveness of the process. We argue that absence of effectiveness is a clue for the characterization of the
7 Conclusions

We can summarize the conclusions as follows:

- Formal methods have theoretical limits imposed by the incompleteness of formal systems;
- The software development goal to obtain best code implementation, regarding the code size, is very difficult to achieve because the non-computability and intractability of the complexity;
- The software development process is formally unpredictable;
- The limits outlined in this paper are valid for all formalisms because the equivalence of additively optimum methods;
- Empirical estimates, based on statistical methods and the human experience, are successful to estimate efforts and costs of software development;
- As pointed, programming diversity (or n-version programming) shows that software quality may be obtained from the experience of a large team of programmers.

These results suggest that software development is more subjective and empirical than objective and formal. Following our conclusions, these limits typifies the software development process as experimental and heuristical based, like, for example, the scientific development in physics and chemistry.

We agree with the use of formal methods for software development and don’t encourage programmers to leave software engineering procedures in the same sense that mathematicians didn’t leave mathematic foundation theory by the simple fact that mathematics is intrinsically incomplete, as proved by Gödel. We only want to show that formal methods don’t have all the answers to the future of software development. We argue in support of the importance of empirical estimates, based on the human experience, in the whole software development process. These conclusions suggest that software engineering is a scientific field not totally characterized by the typical work of engineering, but also by the experimental sciences methodology.

References

Application of Signed Kolmogorov Hashes to Provide Integrity and Authenticity in Web-Based Software Distribution

Josef Scharinger
Johannes Kepler University, Altenbergerstraße 25,
A-4040 Linz, Austria
js@cast.uni-linz.ac.at

Abstract. When downloading software from websites located somewhere on the globe, doubt is in place on authenticity and integrity of the software obtained. Fortunately, cryptologic experience has shown that concepts based on digitally signed message digests can ensure integrity and authenticity in web-based software distribution applications.

It is the purpose of this contribution to introduce a novel approach for generating secure hash values (message digests) computed from input data dependent pseudo-random permutations. Essentially, input messages are processed sequentially using bytes of input data as keys to discrete chaotic Kolmogorov systems which permute an initial message digest in a cryptographically strong manner heavily depending on the input stream.

1 Introduction

In modern software engineering the life span of one specific software version is permanently decreasing. Therefore it is a vital interest of any software vendor to provide his customers with the latest versions available as efficiently as possible, a task most conveniently solved by direct download of new releases and updates from support sites over the WWW.

While this provides a comfortable and efficient way for software distribution and version upgrading, new risks for data integrity and authenticity are encountered. When purchasing software directly from an accredited dealer one will safely assume that the software obtained is authentic, intact and not infected by viruses, worms, trojan horses and the like. But when downloading an upgrade from a website located somewhere on the globe, doubt is in place on authenticity and integrity of the software obtained.

Authenticity can be achieved by the use of digital certificates. Once a vendor certificate is trustworthy installed on the local computer, any software downloaded over the web can be checked for authenticity using that certificate. Since signature schemes available today [11, 3, 8] are rather slow, it is impractical to sign Mega-Bytes of data. Instead a message digest [10, 7] (digital fingerprint, cryptographic hash) of the data is calculated, attached to the original data.
and signed with the vendor certificate. This approach enhances performance, increases cryptographic security and allows integrity checks in a unified framework. Therefore concepts for generating strong digital fingerprints are essential components for ensuring integrity and authenticity in web-based software distribution applications.

It is the purpose of this contribution to introduce a novel approach for generating cryptographic hashes that provide secure digital fingerprints for arbitrary input data. These hashes are derived as 256 bit message digests computed from input data dependent pseudo-random permutations. Initially, a $16 \times 16 = 256$ bit square array is filled with pseudo-random bits. Next, input messages are processed sequentially using bytes of input data as keys to so-called discrete chaotic Kolmogorov systems which permute the square array in a cryptographically strong manner heavily depending on the input stream. Finally, the state obtained in the $16 \times 16$ array of bits is read out to provide the 256 bit cryptographic fingerprint.

The remainder of this contribution will be organized as follows. In section 2 a detailed description of chaotic Kolmogorov systems will be given. Thereby, particular emphasis will be put on the cryptographic analysis of specific discrete versions that can constitute a basis for developing secure message digest functions later on. Next, in section 3 a novel approach for implementing cryptographically strong hash functions computed from input data dependent pseudo-random permutations will be specified. This approach given is analyzed with respect to the level of security provided in detail in section 4. Finally, section 5 recaps on the main findings of this contribution resulting in the conclusion that the novel approach described herein does indeed result in remarkably efficient and strong message digests.

2 Chaotic Kolmogorov Systems

2.1 Continuous Kolmogorov Systems

Continuous Kolmogorov systems \cite{1, 4, 6, 14} act as permutation operators upon the unit square. Figure 1 is intended to give a notion of the dynamics associated with a specific Kolmogorov system parameterized by the partition $\pi = \left(\frac{1}{3}, \frac{1}{2}, \frac{1}{6}\right)$. As can be seen, the unit square is first partitioned into vertical strips which are then stretched in the horizontal and squeezed in the vertical direction and finally stacked atop of each other. Just after a few applications (see figure 1 from top left to bottom right) this iterated stretching, squeezing and folding achieves perfect mixing of the elements within the state space.

Formally this process of stretching, squeezing and folding is specified as follows. Given a partition $\pi = (p_1, p_2, \ldots, p_k)$, $0 < p_i < 1$ and $\sum_{i=1}^{k} p_i = 1$ of the unit interval $U$ and stretching and squeezing factors defined by $q_i = \frac{1}{p_i}$. Furthermore, let $F_i$ defined by $F_i = 0$ and $F_i = F_i - 1 + p_{i-1}$ denote the left border of the vertical strip containing the point $(x, y) \in E$ to transform. Then the continuous Kolmogorov system $T_\pi$ will move $(x, y) \in [F_i, F_i + p_i) \times [0, 1)$ to the position
Fig. 1. Illustrating the chaotic and mixing dynamics associated when iterating a Kolmogorov system.

\[ T_\pi(x, y) = (q_i(x - F_i), \frac{y}{q_i} + F_i). \]  

2.2 Discrete Kolmogorov Systems

In our notation a specific discrete Kolmogorov system for permuting a data block of dimensions \( n \times n \) is defined by a list \( \delta = (n_1, n_2, \ldots, n_k), 0 < n_i < n \) and \( \sum_{i=1}^k n_i = n \) of positive integers that adhere to the restriction that all \( n_i \in \delta \) must partition the side length \( n \). Furthermore let the quantities \( q_i \) be defined by \( q_i = \frac{n}{n_i} \) and let \( N_i \) specified by \( N_1 = 0 \) and \( N_i = N_{i-1} + n_{i-1} \) denote the left border of the vertical strip that contains the point \((x, y)\) to transform. Then the discrete Kolmogorov system \( T_{n, \delta} \) will move the point \((x, y) \in [N_i, N_i + n_i) \times [0, n)\) to the position

\[ T_{n, \delta}(x, y) = (q_i(x - N_i) + (y \mod q_i), (y \div q_i) + N_i). \]

The restriction to integral stretching- and squeezing factors is necessary to keep resultant points at integer positions within the \( n \times n \) grid. Use of the \( \text{div} \) (division of positive integers \( a \) and \( b \) delivering \( \lfloor \frac{a}{b} \rfloor \)) and \( \text{mod} \) (remainder when dividing positive integers) operation ensures that points in \( n \times n \) are mapped onto each other in a bijective and reversible manner.

It is straightforward to check that in the case \( n = 16 \) a total of 55 different partitions \( \delta_i \) \((0 \leq i \leq 54)\) can be found to define permutation operators \( T_{16, \delta_i} \) on \( 16 \times 16 \) 2D arrays. Mapping byte values \( b \in [0..255] \) to valid partitions \( \delta_i \) (e.g. according to \( i = b \mod 55 \); see [12] for more alternatives), permutations \( T_{16, \delta_i} \) can be specified by bytes. Interpreting a message as a sequence of bytes, every message byte may thus be taken as a round-key to this permutation operator. This way a secure hash algorithm that generates strong 256 bit message
digests as the result of message dependent pseudo-random permutations will be constructed.

2.3 Properties of Cryptographic Relevance

Kolmogorov systems tend to permute elements of the state space in a chaotic non-linear and apparently random fashion. After a sufficient number of iterations it becomes extremely hard for an observer to deduce the initial state of a Kolmogorov system from its final state. To be more specific, Kolmogorov systems offer very unique properties of cryptographic relevance that are explained in more detail in the sequel.

Ergodicity

Ergodicity is important for a system that is to be applied in cryptography because it stands as a synonym for confusion. Informally speaking and expressed in terms of permutation systems, ergodicity stands for the property that almost any initial point will move to almost any other position in state space with equal probability as the system evolves in time. In other words there is no statistical way to predict the initial from the final position or vice versa.

Ergodicity of continuous Kolmogorov systems has been proven long ago [1]. As for discrete Kolmogorov systems, we have no knowledge that anyone has succeeded in defining them in a way such that ergodicity can be shown. In the sequel we derive necessary and sufficient conditions on the number of iterations necessary to ensure ergodicity of discrete Kolmogorov systems as introduced by equation 2. Note that this way a constructive proof of ergodicity is achieved.

In the following we restrict attention to the practically most relevant case of \( n = p^n \) being an integral power of a prime \( p \). The discrete Kolmogorov system \( T_{n, \delta} \) is defined by the list \( \delta_r = (n_{1r}, n_{2r}, \ldots, n_{kr}) \) of length \( kr \) containing the positive integers to be used as key in round \( r \). As mentioned before there are the restrictions \( 1 \leq i \leq kr, 0 < n_{ir} < n, \sum_{i=1}^{kr} n_{ir} = n \) and the constraint that all \( n_{ir} \in \delta_r \) must partition the side length \( n \).

Furthermore let the stretching and squeezing factors \( q_{ir} \) to use for vertical strip number \( i \) in round number \( r \) be defined by \( q_{ir} = \frac{n}{n_{ir}} \). This results in quantities \( q_{ir}, q_{ir} \geq p \) that also have to be integral powers of \( p \) because of the divisibility assumption made.

Consider an arbitrary point \((x, y) \in [N_{ir}, N_{ir} + n_{ir}) \times [0, n)\) in vertical strip number \( i \) to be transformed in round number \( r \) under the influence of the key \( \delta_r \) (see equation 2 and figure 1). Coordinates \( x \) and \( y \) can then be expressed by \( q_{ir} \)-adic representations of length \( t_{ir} = \lfloor \log q_{ir}, n \rfloor \) by \( x = \sum_{j=1}^{t_{ir}} x_{jr} (q_{ir})^{t_{ir} - j} \) and \( y = \sum_{j=1}^{t_{ir}} y_{jr} (q_{ir})^{t_{ir} - j} \). Similarly \( N_{ir} \) can be expanded according to \( N_{ir} = \sum_{j=1}^{t_{ir}} N_{ir} (q_{ir})^{t_{ir} - j} \) and \( x - N_{ir} \) may be expressed as \( x - N_{ir} = \sum_{j=1}^{t_{ir}} x_{jr} (q_{ir})^{t_{ir} - j} \). Obviously \( x \) is the sum of \( x - N_{ir} \) and \( N_{ir} \).

To clarify these relations, the following illustration should be helpful.
According to equation 2 application of $T_{n,\delta_r}$ will move the point $(x, y)$ to a new position $(x', y') = T_{n,\delta_r}(x, y)$ with coordinates $x' = q_{ir}(x - N_{ir}) + (y \mod q_{ir})$ and $y' = (y \div q_{ir}) + N_{ir}$, as made clear by the subsequent figure.

Suppose that lists $\delta_r$ are chosen independently and at random. Neglecting the constraint $N_{ir} \leq x$ which follows from the fact that $N_{ir}$ is the left border of the vertical strip containing the point $(x, y)$ for a moment, the proof of ergodicity becomes straightforward. $N_{ir}$ adds random $q_{ir}$-bits to all the $q_{ir}$-bits of $y'$ yielding a random value for the new $y$-coordinate in one step. Cyclically shifting the least significant position of the $y$-coordinate to the least significant position in the $x$-coordinate and shifting these random $q_{ir}$-bits towards more significant positions in the $x$-coordinate ensures that after at most an additional $\max_{i=1}^{k_r} t_{ir} \leq m$ iterations the transformed point can move to almost any other position in state space with equal probability. Thus ergodicity is achieved after at most $m + 1$ iterations.

Now let us pay attention to the constraint $N_{ir} \leq x$. A moment of thought reveals that the worst non-trivial point that will need the largest number of rounds until being able to move to any position has a $x$-coordinate of 0 and a $y$-coordinate where just $y_{t_{ir}}$ is different from zero. Then it takes at most $m + 1$ iterations until the second-least significant $q_{ir}$-bit in the $x$-coordinate is set and the least significant $q_{ir}$-bit in $N_{ir}$ (and also in the $x$-coordinate!) may assume any random value. By shifting $q_{ir}$-bits towards more significant positions in the $x$-coordinate every iteration causes one additional position in $x$ to become random and by adding $N_{ir}$ the same applies to the $y$-coordinate. This way it is guaranteed that after another at most $m - 1$ iterations ergodicity is achieved after at most $2m$ steps in total.

The preceding discussion can be summarized as follows:

**Theorem 1.** Let the side-length $n = p^m$ be given as integral power of a prime $p$. Then the discrete Kolmogorov system $T_{n,\delta_r}$ as defined in equation 2 is ergodic provided that at least $2m$ iterations are performed and lists $\delta_r$ used in every step $r$ are chosen independently and at random.

In the discussion above we have noted that the restriction $N_{ir} \leq x$ to observe in every step significantly increases the number of iterations necessary until an

---

1 This is a common assumption whenever proving specific properties of iterated cryptographic schemes. Round keys are generally supposed to be random and independent.
initial point can move to any other position. Particularly points with small (zero) $x$-coordinate need a long time until exhibiting ergodic behaviour. However, a simple trick can help a lot in reducing the number of iterations necessary to achieve ergodicity of the underlying system: after every discrete Kolmogorov permutation round just apply a cyclic shift by $\frac{n}{2} - 1$ to the elements in the $n \times n$ array. This corresponds to adding $\frac{n}{2} - 1$ modulo $n$ to every $x$-coordinate and helps points with initially small $x$-coordinates to move to any other position in a reduced number of rounds. Additionally this simple trick also solves the problems associated with the fixed points $(0,0)$ and $(n - 1, n - 1)$ so that not just almost all points can move to almost any position but really all of the $n \times n$ points will have ergodic behaviour.

**Exponential Divergence** Exponential divergence is essential for a system that is to be applied in cryptography because it stands as a synonym for diffusion. Informally speaking and expressed in terms of permutation systems, exponential divergence implies that neighboring points contained in the same subspace of the state space (e.g. points of the same vertical strip corresponding to the same block of the defining partition) diverge at an exponential rate. This way even highly correlated points in input blocks will quickly loose correlations and structures present in input data will soon disappear.

Exponential divergence of continuous as well as discrete Kolmogorov systems follows immediately from their defining equations. In the sequel we will focus on a detailed analysis of the exponential divergence behavior associated with discrete Kolmogorov systems resulting in a simple formula which gives the maximum number of rounds necessary until minimum differences in inputs propagate to the most significant position.

Once again we restrict attention to the practically most relevant case of $n = p^m$ being an integral power of a prime $p$. The discrete Kolmogorov system $T_{n,\delta}$ shall be defined by the list $\delta_r = (n_{1r}, n_{2r}, \ldots, n_{kr})$ of length $k_r$ to be used as key in round $r$ under the restrictions specified in section 2.3. Stretching and squeezing factors $q_{ir}$ to use for vertical strip number $i$ in round number $r$ are defined by $q_{ir} = \frac{n}{n_{ir}}$ ensuring that quantities $q_{ir}$, $q_{ir} \geq p$ also have to be integral powers of $p$.

A moment of thought reveals that points that will need the largest number of rounds until exhibiting exponential divergence have the same $x$-coordinate and just differ at the most significant digit in the $q_{ir}$-adic representation of their $y$-coordinate. So we consider arbitrary points $(x, y)$ and $(x, y + dy)$ with $dy = dy_{1r}(q_{ir})^{t_{ir} - 1}$ contained in $[N_{ir}, N_{ir} + n_{ir}) \times [0, n)$ in vertical strip number $i$ in round number $r$. Coordinates can then be expressed by $q_{ir}$-adic representations of length $t_{ir} = \lceil \log_{q_{ir}} n \rceil$ by $x = \sum_{j=1}^{t_{ir}} x_{jr}(q_{ir})^{t_{ir} - j}$ and $y = \sum_{j=1}^{t_{ir}} y_{jr}(q_{ir})^{t_{ir} - j}$. Similarly $N_{ir}$ can be expanded according to $N_{ir} = \sum_{j=1}^{t_{ir}} N_{jr}(q_{ir})^{t_{ir} - j}$ and $x - N_{ir}$ may be expressed as $x - N_{ir} = \sum_{j=1}^{t_{ir}} x_{jr}(q_{ir})^{t_{ir} - j}$. The following figure illustrates these relations.
According to equation 2, application of $T_{n,\delta_r}$ will move these points to new positions $(x', y')$ and $(x', (y + dy)')$ with coordinates specified in the subsequent figure.

It has been noted before that $q_{ir} \geq p$ implies $t_{ir} \leq m$. As depicted above the first at most $m$ iterations do not exhibit any exponential divergence at all. On the contrary there is exponential convergence since the influence of $dy$ is reduced by at least a factor of $p$ in every iteration. But as soon as $dy$ has been cyclically shifted to the least significant position in the $x$-coordinate (via cyclic shift from the least significant position in the $y$-coordinate), exponential divergence of the trajectories starting at points $(x, y)$ and $(x, (y + dy))$ is observed. $dy$ is shifted towards the more significant positions in the $x$-coordinate ensuring that after at most $2m - 1$ iterations even minute deviations in inputs will have propagated at least once to the most significant position in the $x$-coordinate.

The preceding discussion can be summarized as follows:

**Theorem 2.** Let the side-length $n = p^m$ be given as integral power of a prime $p$. Then the discrete Kolmogorov system $T_{n,\delta_r}$, as defined in equation 2, exhibits exponential divergence of points contained in the same blocks defined by partitions $\delta_r$ ensuring that after at most $2m - 1$ iterations arbitrary non-zero deviations between initial points have propagated at least once to the most significant position in the $x$-coordinate.

**Mixing Property** The mixing property is important for a system that is to be applied in cryptography because it stands as a synonym for confusion as well as diffusion. Informally speaking and expressed in terms of permutation systems, fulfillment of the mixing property implies that any subspace of the state space will dissipate uniformly over the whole state space. Obviously this is an even stronger requirement than ergodicity because it does not only imply that almost any point will move to almost any position in state space with equal probability but also that distances between neighboring points within certain subspaces will become random as the system evolves in time.
Attention is again restricted to the practically most relevant case of $n = p^m$ being an integral power of a prime $p$. The discrete Kolmogorov system $T_{n, \delta_r}$ shall be defined by the list $\delta_r = (n_{1r}, n_{2r}, \ldots, n_{kr})$ of length $k_r$ to be used as key in round $r$ under the restrictions specified in section 2.3. Stretching and squeezing factors $q_{ir}$ to use for vertical strip number $i$ in round number $r$ are defined by $q_{ir} = \frac{n}{n_{ir}}$ ensuring that quantities $q_{ir}, q_{ir} \geq p$ also have to be integral powers of $p$.

Basically, mixing requires two things. First points within arbitrary subspaces must be de-correlated. Next all points must be able to move to any position in state space with equal probability. In section 1 it has been shown that after at most $2^m - 1$ rounds differences $dx \neq 0$ or $dy \neq 0$ separating arbitrary points $(x, y)$ and $(x + dx, y + dy)$ will have at least once contributed to the most significant position in the $x$-coordinate of the transformed points. This situation may be illustrated as follows ($dx_1 \neq 0$):

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{t_{1r}}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$dx$</th>
<th>$dy$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dx_{t_{1r}}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

The important point about this situation is that in this round those points initially separated by $dx, dy$ must be in different blocks of the partition $\delta_r = (n_{1r}, n_{2r}, \ldots, n_{kr})$ as can be shown by contradiction. Knowing $dx_{1r} \neq 0$ implies $dx \geq q_{t_{ir} - 1} \geq p^{m-1}$. Assume that points would be in the same block of the partition. This would multiply the difference in $x$-coordinates by $q_{ir}$ resulting in a difference $dx' \geq q_{t_{ir}} \geq p^m \geq n$. Obviously such a difference in $x$-coordinates is impossible when operating on a grid constraining coordinates to remain in the range $[0..n - 1]$.

This implies that within the iteration of $2m$ rounds the trajectories of arbitrary initial points have to pass through different blocks with indices $i_1$ and $i_2$. In other words: iteration of $2m$ rounds ensures that arbitrary points get separated and de-correlated by different offsets $N_{i_1r}$ and $N_{i_2r}$ yielding differences $dN_{i_1i_2r} = \|N_{i_1r} - N_{i_2r}\|$ in offsets that are different from zero and random at least in their least significant positions. Following the reasoning used in section 2.3 such differences will go ergodic in at most $2m$ additional rounds, taking a total of at most $4m$ rounds for differences to get ergodic.

Now we can combine this with results derived for points in section 2.3. Every round adds randomness to the subsequent positions of points $(x, y)$ by means of $N_{ir}$ provided that lists $\delta_r$ are chosen independently and at random and it has been shown that iteration of $2m$ rounds ensures that any point can move to any position with equal probability. Combining arguments for points and differences finally leads to the conclusion that iteration of $4m$ rounds ensures fulfillment of the mixing property of the underlying discrete Kolmogorov system.

The preceding discussion can be summarized as follows:

**Theorem 3.** Let the side-length $n = p^m$ be given as integral power of a prime $p$. Then the discrete Kolmogorov system $T_{n, \delta_r}$ as defined in equation 2.3 is mixing...
provided that at least $4m$ iterations are performed and lists $\delta_r$ used in every step $r$ are chosen independently and at random.

2.4 Conclusion

Summarizing the preceding discussion, a simple law on the number of rounds necessary to ensure that all the essential cryptographic properties of discrete Kolmogorov systems are fulfilled can be stated as follows:

**Theorem 4.** Let the side-length $n = p^m$ be given as integral power of a prime $p$. Then the discrete Kolmogorov system $T_{n,\delta_r}$ as defined in equation 2 fulfills the properties of ergodicity, exponential divergence and mixing provided that at least $4m$ iterations are performed and lists $\delta_r$ used in every step $r$ are chosen independently and at random.

Equipped with this solid knowledge it will be shown in the remainder of this contribution that discrete Kolmogorov systems can successfully be applied to develop strong and efficient hash functions to ensure communications integrity.

3 Permutation Hashes Based on Chaotic Kolmogorov Systems

To provide integrity and authenticity in secure communications applications at reasonable computational costs, efficient and strong cryptographic hash functions are needed. Our approach to compute a message digest based on discrete chaotic Kolmogorov systems runs as follows.

First a $16 \times 16$ square array of bits is initialized with 256 pseudo-random bits (128 zeros, 128 ones) taken from the after-comma binary expansion of some “magic” constants ($\pi$, $e$, golden ratio $\phi$, $\sqrt{2}$, $\sqrt{5}$, etc.) as done in almost any cryptographic hash function. Taken line-by-line or column-by-column, this provides the initial 256 bit message digest $MD_0$.

After initialization, in every step $t = 1, 2, \ldots$, the message digest $MD_{t-1}$ is updated by processing the message in blocks $W_t$ of 256 bit each. Since message lengths are usually not a multiple of 256, padding the last block with arbitrary constant bits may be necessary.

Now these 256 message bits are XORed with the current 256 bit message digest to obtain $X_t = W_t \oplus MD_{t-1}$. This step ensures that any block contains approximately an equal number of zeros and ones, regardless of the message block (which could be entirely zero etc.).

To maximize input avalanche effects, the 8 32-bit words $X_t(i)$ ($0 \leq i \leq 7$) are processed according to a linear recurrence relation. First a forward dissipation step is done according to $Y_t(0) = X_t(0)$, $Y_t(i) = aY_t(i-1) + b \mod 2^{32} \oplus X_t(i)$ with parameters $a$ and $b$ set accordingly (see e.g. [9] for a large variety of suitable parameter settings) to give pseudo-random sequences $Y_t(i)$. This is followed by a backward dissipation step (with index $i$ decreasing) according to $Z_t(7) = Y_t(7)$, $Z_t(i) = aZ_t(i+1) + b \mod 2^{32} \oplus Y_t(i)$. 
After preprocessing the message block $W_t$ to obtain the block $Z_t$, the actual hashing step takes place. The 256 bit of $Z_t$ are used to provide 32 key bytes $Z_t(i,j)$ ($0 \leq i \leq 7$, $0 \leq j \leq 3$) to permute the message digest $MD_{t-1}$ stored in the $16 \times 16$ array of bits using the corresponding discrete Kolmogorov system. For several reasons, a cyclic shift by 7 positions follows each of the 32 rounds involved in calculating the updated message digest $MD_t$.

Figure 2 summarizes one round when calculating data dependent chaotic permutation hashes based on chaotic Kolmogorov systems. Iterating this procedure for all blocks of the input message and finally reading the $16 \times 16$ 2D array line-by-line or column-by-column delivers the 256 bit message digest of the message to hash in a very efficient and elegant manner as pseudo-random message-dependent permutation of the initial message digest $MD_0$.

\[ W_t \]
\[ X_t = W_t \text{ XOR } MD_{t-1} \]
\[ Y_t(0) = X_t(0); \quad Y_t(l) = aY_t(l-1)+b \mod 32 \text{ XOR } X_t(l) \]
\[ Z_t(7) = Y_t(7); \quad Z_t(l) = aZ_t(l+1)+b \mod 32 \text{ XOR } Y_t(l) \]

Fig. 2. One step in calculating data dependent chaotic permutation hashes based on discrete Kolmogorov systems.

4 Security Analysis

Informally speaking, a message digest scheme is called secure, if it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

\[ MD_{t-1} \]
\[ \text{Cyclic shift (7 positions)} \]
\[ T_{16, (2\{0,0\})} \]
\[ \text{Cyclic shift (7 positions)} \]
\[ MD_{t} \]

It has been shown that fulfillment of essential cryptographic properties by discrete Kolmogorov systems with side length $n$ needs iteration of at least $4 \times \log_2 n$ steps. This property can be achieved in less steps (see arguments given in proving ergodicity), if each iteration is followed by a cyclic shift by $\frac{n}{2} - 1$ positions. Additionally this resolves problems related to fixed points $(0,0)$ and $(n-1,n-1)$.
Any change to a message in transit will, with very high probability, result in a
different message digest, and the signature will fail to verify.

Security of cryptographic hashes calculated from discrete Kolmogorov per-
mutations is extremely well based on the properties given in section 2.3. Addi-
tionally, we have done extensive evaluations on the cryptanalytic quality of the
data dependent chaotic permutations generated by discrete chaotic Kolmogorov
systems with respect to confusion, diffusion and fixed point distribution proper-
ties as well as differential and linear characteristics. These quantitative results
will be summarized in the sequel.

4.1 Quantitative Security Analysis

Number of Different Message Digests Since any 256 bit message digest
generated is a permutation of an initial message digest having 128 zeros and
128 ones, the number of different message digests is \( \binom{256}{128} > 2^{251} \) and thus far
beyond the number of particles in our galaxy [13]. Therefore it can be considered
extremely unlikely that two different input messages lead to the same message
digest.

Confusion We have done extensive \( \chi^2 \)-testing to verify the claim that data
dependent chaotic permutations generated are uniformly distributed regardless
of the initial distribution. Results justify the assumption that the permutations
generated are statistically indistinguishable from a uniform distribution, a result
expected due to the mixing property associated with any chaotic Kolmogorov
system.

Diffusion We demand that even for very similar messages completely differ-
ent permutations will be generated. Checking average diffusion distances the
diffusion distances observed rapidly converge to the optimum diffusion distance
expected for completely un-correlated permutations.

Singleton Cycles Distribution Suppose we are given a list of \( n \) elements,
the resulting \( n! \) permutations can be listed according to the number of singleton
cycles in groups with no fixed element, one fixed element, two fixed elements, ...,; all \( n \) elements fixed. Assuming that \( n \) tends to infinity the probability \( p_k \) that a
random permutation has exactly \( k \) fixed elements is \( p_k = \frac{1}{e k!} \). It was stunning
how close the singleton cycles distributions observed matched the distribution
expected for a random permutation. Therefore there was no evidence that per-
mutations generated have a singleton cycles distribution significantly different
from a random permutation.

Differential Analysis Differential cryptanalysis [2] analyzes the effect of par-
ticular differences in input pairs on the differences of the resultant output pairs.
In the ideal case of a random permutation of 256 elements the most likely non-
trivial differential characteristic has a probability of $\frac{1}{2^{255}}$. Deducing from our experiments we claim that the most likely differential characteristic observed rapidly converges to that optimum value.

**Linear Analysis** Linear cryptanalysis studies linear relations between bits of inputs and corresponding outputs. When deriving linear relations one chooses a subset of the input bits and the output bits, calculates the parity (XOR) of these bits for each of the possible inputs and counts the number of inputs whose subset’s parity is zero. Ideally this number does not differ from half at all. Deducing from our experiments it can be assumed that all non-trivial linear characteristics have a probability rapidly converging to the optimum value $(\frac{1}{2} \pm 0)$ that can be expected for random permutations.

## 5 Conclusion

Concepts for generating strong digital fingerprints are essential components for ensuring integrity and authenticity in secure communications applications. In this contribution we have introduced a novel approach for generating cryptographic hashes (digital fingerprints) that can be characterized as follows:

- **Efficiency**: Time-consuming operations (multiplications) are only needed in preprocessing the input block. Implementation of the chaotic permutations can be made extremely fast. Since all stretching and squeezing factors are restricted to be integral powers of 2, all the operations involved in computing a new position for points in the array can be done by just using additions, subtractions and bit shifts.

- **Security**: Permutation hashes generated fulfill all requirements essential for secure hashes. There is an enormous number of different fingerprints and permutations generated are indistinguishable from random permutations with respect to confusion, diffusion and fixed point distribution as well as differential and linear characteristics. Note that this observation is perfectly in line with the proven cryptanalytic properties of chaotic Kolmogorov systems (ergodic, mixing, exponential divergence).

Combining security arguments with the ease and flexibility in implementation possible for discrete Kolmogorov systems it is expected that signed Kolmogorov hashes can be of considerable value in applications intending to provide integrity and authenticity in web-based software distribution.

**References**


Toward a Formalisation of Evolutionary Hypermedia Systems Based on System Theory

Lina García-Cabrera¹, María José Rodríguez-Fórtiz², and José Parets-Llorca²

¹ Departamento de Informática, E. P. S., Universidad de Jaén
Avda. Madrid, 35. 23071 –JAEN (Spain)
lina@ujaen.es

² Departamento de L. S. I. ETS Ingeniería Informática, Universidad de Granada.
Avda. Andalucía, 38. 18071 –GRANADA (Spain)
jparets@ugr.es

Abstract. Main development and use activities of Hypermedia-Systems evolve through time. Hypermedia-systems need models that support the evolutionary nature of their building, maintenance and navigation processes. The System Theory and cognitive models offer a better perspective of web-systems and succeed in abstracting their structure, information and behaviour. We assume that an Evolutionary Hypermedia System must be modelled by means of a set of interconnected and interacting systems that allow: a) from the author’s viewpoint, complete and flexible control of the development and maintenance of hyperdocuments; b) from the reader’s point of view, an understandable navigation that allows easy access to and selection of information. The Model allows an explicit representation of the semantic content which allows us to structure the information-system and determines its possibilities of change, updating and evolution. In addition, the model is flexible enough in offering the necessary mechanisms to incorporate and represent the author’s conceptual-domains in order to characterise the information-domains.

1 Introduction

Traditional hypermedia reference models ⁴, ⁷, ⁸ “tend to focus on abstracting the connectivity of hypermedia –links- from its underlying information -nodes- rather than abstracting structure from functionality” ¹², i.e., these focus more on edition and document navigation through prefixed links than on the dynamic construction, evolution and maintenance of the document. The traditional skeleton of hypermedia models, based on a set of hierarchical levels that can be translated into a sequential and static methodology, is not the best approach in representing complex and evolving realities, where construction, maintenance and navigation are confused by their strong interrelationships. In these models there is no correspondence between structure and functionality. In our opinion, a functional systemic perspective is more

¹ This research is supported by a project –MEIGAS- by the Spanish CICYT (TIC2000-1673-C06-04) which is a subproject of the DOLMEN project (TIC2000-1673-C06).
© Springer-Verlag Berlin Heidelberg 2001
suitable and hypermedia systems must be conceived under an evolving model with the following assumptions:

1. Hypermedia systems need a functional systemic perspective [9], [11], that is, a hypermedia system can be conceived as a set of interacting systems in continuous evolution.

2. The model must help and make flexible the construction, maintenance and navigation of the hypermedia systems. These three key aspects are exposed to continuous changes and updates the model should be able to integrate.

3. An explicit semantic representation must permeate the model. The possibilities of which structuring and further changes, adaptations or evolution will depend on the level of explicitness of this semantic representation. The building process of a hypermedia system must be based on a cognitive model [5].

4. A cognitive model benefits the users—author and reader—during development and use activities: construction, maintenance and navigation.
   - The author can make an incremental design process of his/her hyperdocuments.
   - Collaboration between authors is possible.
   - Effective maintenance is possible when the process of underlying reasoning and decision-making carried out or Design Rationale [13] is represented.
   - The reader can have a contextual access that facilitates his/her knowledge and comprehension.

5. The model must offer a flexible semantic representation that allows the author a characterisation of his/her own information domains by means of his/her own ontologies.

Guided by these objectives, we present here a Semantic-Evolutionary Model (SEM-HP). Section 2 will present a general architecture of the proposed Model and an example, which will be used in further explanations. After that, in section 3, the important concepts of the model are defined and explained. Section four provides an extensive description of the Knowledge and Navigation Systems. Section 5 goes back to the architecture in order to explain a more detailed view of the functionality and evolution of the Systems. Finally, section 6 summarises the conclusions about the presented model, current research and further developments.

2 Architecture of Evolutionary Hypermedia Systems

Stemming from the previous assumptions a Semantic-Evolutionary Model is proposed in order to support HyPermedia Systems, SEM-HP, with the following three systems—figure 1-: The Knowledge System, the Navigation System and the Learning System.

The Knowledge System is in charge of the storage, structuring and maintenance of the different pieces of information. It memorises the knowledge acquired about the information system that is represented. This knowledge will guide the design and structuring processes of the information system. It will determine the possibilities of transformation and change of this structure throughout its evolution. The Navigation System helps the reader in his/her interaction with the information system. Using the knowledge base and the reader activity through time dynamically, this system determines—firstly—the accessible information and—secondly—its interaction possibilities.
Finally, The *Learning System* optimises the knowledge acquisition process from the hypermedia system adapting navigation to the information needs and to the knowledge gained by the reader.

![Fig. 1. Semantic-Evolutionary Model based on Systems. Different systems interact among themselves -black arrows-. The Reader interacts with the Navigation and Learning Systems, while the author interacts with the three systems -grey arrows-.](image)

Each System redefines, to some extent, the knowledge base provided by the Knowledge System, which is stable for the reader but dynamic for the author or authoring tool. Each System is supported by itself and contributes additional information. This information will determine what pieces of information can be consulted and under what prism. The different systems interact among themselves and their interaction produces, in a dynamic way, adaptations within them. In order to clarify the explanation and show the possibilities of the approach we use a concrete conceptual and information domain example about the *Solar System*—figure 2—.

### 3 The Conceptual Perspective of the SEM-HP

In order to highlight the evolving aspects of the model we will start by explaining the meaning of the four most important concepts of our model: *Information Items (II)*, *Conceptual Structure (CS)*, *Restrictions (RT)* and *Evolving Actions (ACε)*. In their explanations other basic concepts will appear, and the interactions between them will be shown. We will also see how Conceptual Structure and Restrictions stress the cognitive and evolving aspects of the hypermedia system.
3.1 Information Items

A hypermedia system is an information system made up of different pieces of information—the information items—that can be referred to, used and composed. These pieces can be expressed in any language and can be provided by the authoring system, a computer application or a database. The author can associate properties to these items. These properties include the type of information and the function of the item in a context.

An information item is referenced by one or more concepts or labelled ideas. These concepts will be part of a Conceptual Domain—our example has 12 concepts represented by ellipses; i.e., Planets, Stars, etc—created by the author during the development and maintenance of the hyperdocument. The set of information items identified by the concepts in a Conceptual Domain will be called Information Domain—the example contains 13 items represented by a square; i.e., Sun, C2, etc—.

Many information items can refer to the same concept or set of concepts. In this case each information item will play a different function in a context. We call this function or intention of the information item its Role. The Role is useful for the author because it provides the function of the item, and for the reader because it guides the

---

2 Here the term “item” is preferred to “chunk”, more widely used in the literature, for two reasons: a) historical fidelity because this was the word used by Vannevar Bush in his memorable and forerunner paper “As We May Think” [1]; and b) we consider that the term “information item” represents better the idea of an own-entity piece of information that is implied by a conceptual unit.

3 More than one semantic model in research literature simply builds documents as compositions of data represented and presented as a style template, that is to say, in the form of a database. Unfortunately, many documents cannot be adapted to this simple skeleton.
intention of a link. For instance, the Sun concept -one of the Stars- is referenced by different items, which play the roles of photos or chemical-composition. Apart from concepts and roles an information item has additional properties: type of language of the content -text, sound, graphics, image, animation, execution or hypermedia-, edition aspects -authors, date, revision date, quality, visitors,...- and Specialisation Level or version. This property is related to the user's degree of knowledge and allows the possibility of items with different depth levels, which can be selected by individual readers. For instance, some properties of the Po2 item are Portugal, map, image, PlanetEducation, 26feb00, teenagers.

3.2 Conceptual Structure

The set of concepts of a Conceptual Domain constitutes a directed graph, in which nodes and links are labelled with semantic meanings –a semantic net [14]. The graph represents the Conceptual Domain –concepts and associations- of the information system, named Conceptual Structure (CS). The different information items –documents- can be associated –labelled- with one or more concepts of the CS –i.e., <Stars, def, S1>, see figure 2-. These items are also nodes of the CS. In order to allow provisional and incomplete development, items which are not related to any concept can also be included. Therefore CS is defined as: CS = (C, II, A_c, A_i), where C is the set of concepts, II is the set of information items, A_c is the set of labelled conceptual associations and, A_i is the set of labelled associations between concepts and information items.

We distinguish between Reference and Dependency Conceptual Associations. Reference Conceptual Associations –i.e. <Earth, rotate, Moon>- are domain dependent and must be defined by the author for each particular conceptual domain, i.e. the author provide his own ontologies [17]. These ontologies -concepts and associations between concepts- define a dictionary of keywords which is used by the author in order to provide the structure, and by the reader in order to select material.

In addition, Dependency Conceptual Associations, which are domain independent and have a generic character, can be considered: aggregation (partOf), instantiation (isA) and specialisation (aKindOf). The dependency partOf allows hierarchies between concepts –i.e. <Solar-System, part-of, Planets>- . The dependency aKindOf allows the composition of information items –i.e. <Stars, kind-of, Nova>-. For instance, Nova and Supernova have an association aKindOf with Stars. Then, a composed item, which is labelled with the generic concept –Stars-, can be constructed by grouping all items associated with the children concepts. The dependency isA allows the definition of a concept using more generic concepts -i.e. <Stars, isA, Sun>-.

Conceptual associations allow the definition of the Concept Environment, i.e. the set of concepts which are related to another concept. In the example, the environment

---

4 We prefer the term “association” instead of “link” because links have a clear static meaning in current models and links are more diffuse in the research literature. The term association reflects the fact that this connection between information items responds to relationships between the concepts represented by them more than to circumstantial reasons -as usually occurs in links-. 
of Stars is made up by concepts such as Solar-System, Nova or Sun. The notion of environment allows some interesting operations which are known in the literature as *queries based on the structure*:

- Which concepts add more information to another concept.
- Which concepts are derived from another concept.
- Which concepts produce or cause another concept.
- Which concepts are one level higher or lower in the conceptual structure.
- Which concepts are separated from another concept by a distance d.
- Which documents are related to some conceptual domain.

The previous conceptual associations allow the dynamic creation and evolution of computed documents, i.e. the authors can construct new documents by means of this explicit semantic structure. *Restrictions* about conceptual associations also guide the authors during the construction and maintenance of the Conceptual Structure because they can forbid some structures and associations -see below- in a concrete information domain.

### 3.3 Restrictions

Restrictions (RT) guide the development, maintenance and navigation of hypermedia systems. They are supplied by different Systems, and are always applied -as we will see later - by these systems. They limit *associations* between concepts in the CS and constrain associations of information items that can be used during navigation. Dynamically, way a set of restrictions will hold for each information item and they will limit the set of associated items. We will call this set the *Item Framework*. Two types of restrictions can be distinguished:

1. Derived from the semantic structure of the information system. Obviously, navigation will be restricted inside the world conceived and designed by the author. These restrictions will be applied by the *Knowledge System (KS)* and can be basic, defined as a functional part of the KS, or can also be defined by the author. Some examples of basic restrictions are:
   - Each association of the CS must connect two concepts or a concept and an item.
   - Each arc and node of the CS must be labelled.
   - Two nodes in a CS cannot have the same label.
   The author can also include additional restrictions which determine what associations between concepts are possible. In order to represent these restrictions, formulas in temporal logic are used. This formalism also allows checking if the CS is valid at any moment. Some examples are:
     - The concept Stars can be connected to the concept Planets by means of the association rotate.
     - The association rotate must be acyclic.
     - A concept X can be connected with concept Countries if the concept Countries is previously reached from the concept Earth.

2. Derived from the navigation itself and providing different navigation 'styles' which can be performed using the same semantic structure:
the type of navigation: a group of restrictions that constrain more or less the navigation paths of a Conceptual Structure;

- the navigation carried out by user through time or functional history. The functional history is the set of operations performed by the user during a work session, i.e. the information items selected by the reader and their order;

- considerations about security and access control: user identification, restrictions in accessing the Conceptual Structure, item roles and item versions.

The possibility of adding restrictions implies adaptations and changes in the hypermedia system. These restrictions are described formally using graph theory and in a temporal logic language—a more detailed use of these formalisms can see in [6]—which supports expressions as: “if this and that then...”, “if before ... and after ... then show...”, “take into account whether the reader knows these or those concepts “, “if the reader has made this tour.... then these and those items can be shown”. Like Stotts and Furuta [16] we consider that a hypertext is an interactive document which provides a dynamic structure. This assumption implies the need for temporal logic in expressing what link sequences can be followed during browsing. These authors propose temporal logic as formalism in checking the properties of a hypertext. In our approach we also use temporal logic as an inherent way of expressing restrictions. Consequently, this kind of rules determines, at all times, which pieces of information can be reached and which are the information items that can be searched. These rules are provided by the hypermedia author and are indirectly selected by the reader when he/she specifies a navigation type or navigates through the system. In the example, the items labelled with the concepts Nova or Supernova and the items subordinated to these concepts should be hidden to a user who does not know the definition of Stars concept.

3.4 Evolving Actions

All systems include a set of evolving actions (AC_e) that allow changes to be made and propagated in the hypermedia system. An evolving action can belong to three different types:

1. Actions that redefine some aspects the system. Obviously the basic restrictions, defined by the system, discussed below, RT_s, cannot be changed.

2. Actions that control the propagation of these changes inside the system itself.

3. Actions that control the propagation of these changes outside the system, i.e. in the other systems of the SEM-HP.

When these actions are carried out they change the corresponding elements of the hypermedia system. Because integrity should be guaranteed in any case, these operations should be carried out following a set of meta-restrictions. The specification of these meta-restrictions implies a meta-level in the definition of the Systems.
4 The Systems of the SEM-HP

Each of the systems of the SEM-HP can be defined by: a) one or more artefacts which represent its particular vision of the conceptual and information domain, b) a set of restrictions RT which control the construction and guarantee the consistency of these artefacts and, c) a set of evolving actions ACe that allow changes to be made and propagated in the hypermedia system. In next subsections we will define and describe the different systems and their components. For better understanding, the previous definitions are included in the appendix.

4.1 Knowledge System

The main objective of the Knowledge System is the storage, structuring and maintenance of the different pieces of information. It is made up by a Memorisation Subsystem and a Presentation Subsystem.

The Memorisation Subsystem allows the storage of selected knowledge for each Domain Information –pages or documents-. It memorises information concerning the whole Conceptual Domain –concepts and conceptual associations- (definition 4), which is managed in a particular information system. The elements to be managed are:

1. The Conceptual Structure (definition 8) which allows information items (definition 1) to be catalogued. The CS is formalised by a directed graph, CS = (C, II, Ac, Ai), where C is the set of concepts, II is the set of information items, Ac is the set of labelled conceptual associations, Ai is the set of labelled associations between concepts and information items.

2. The Information Items: the different pieces of information that can be used to construct hyperdocuments. These information items will be expressed in one or more possible language/s -such as text, sound, graphic, image, animation, execution or hypermedia- and will have to be catalogued under one or several concepts of the domain. They will also be labelled with one or several roles into a particular context. They will have certain edition properties.

Because CS is constructed by the authors, dynamically, some evolution actions ACem such as add-concept, delete-association, modify-association, add-item, etc. have to be included. The actions must verify a set of restrictions RT in order to maintain the consistency of the CS. These restrictions can be basic ones RTs, defined as a functional part of the MS, or can also be defined by the author RTa – as described in the 3.2 section-. Therefore, the Memorisation Subsystem is defined as MS = (CS, RT, ACem), where CS is the previously defined, directed and labelled weakly connected graph that represents the conceptual domain of a hypermedia system, RT is the set of restrictions and ACem is a set of evolving actions -see next section-.

The Presentation Subsystem determines the set of possible views of a specific Conceptual and Information Domain. To some extent it establishes the possible views of the hypermedia documents which can be built with the items of the Memorisation Subsystem. The Presentation Subsystem, using as basis the CS of the Memorisation System, allows a selection of a subset of the concepts and associations included in CS. This graph, CP, a subgraph of CS, CP = (Cp, Ip, Acp, Ap), will be presented to the
reader. The Conceptual substructure chosen by the author must respect, absolutely, all the restrictions (RT) set in the Memorisation System. Each time, the author change the substructure, the system must check that the new conceptual and information Domain selected verifies the restrictions. For instance, figure 3 shows the subgraph CS_p chosen by the author, taking into account the Earth out associations.

![Diagram of CS_p selection of CS.](image)

**Fig. 3.** CS_p selection of CS.

Therefore, the *Presentation Subsystem* is defined as \( PS = (CS_p, RT, AC_{ep}) \), where CS_p is a subset of the original CS, RT is the set of the same restrictions of the Memorisation Subsystem and AC_{ep} is a set of evolving actions that allows the author to limit or reduce the CS.

As a result, the Knowledge System stores the pieces of knowledge of the conceptual worlds that the author will use in his/her documents. This System permits the specification author restrictions RT_a. Using these restrictions the system can help the author in creating and maintaining –guaranteeing its consistency- their conceptual and information domains.

### 4.2 Navigation System

The *Navigation System* permits browsing and remembering the memorised knowledge, adapting it to the characteristics and interaction of the reader. The Navigation System permits the ordering of, in some form, the Conceptual Structure and the Information Domain associated to it, both offered by the Presentation System.

We can consider navigation as the execution of a particular presentation. The Navigation System has to take into account the following information at all times:
- First, the information item where the document reader is located at any moment,
- Second, the *conceptual environment* of the information item,
- Third, item *information framework*, i.e. the restrictions set that is true for an information item.

Therefore, the Navigation System, using as basis the CS_p of the Presentation Subsystem, can add more restrictions in order to follow more restricted paths in the subgraph. These restrictions or navigation rules RT_n are expressed formally using temporal logic. Considering the CS_p and temporal restrictions, a Petri net can be automatically constructed. As demonstrated in [10] and in [15], Petri nets give an operational
semantics to temporal logic formulas allowing operational navigation. The algorithm which constructs a Petri net from temporal logic formulas is explained in [10]. Summing up, the Navigation System is defined as $NS = (CS_p, RT_n, PN, AC_{en})$, where $RT_n$ is the set of restrictions specified by the author by means of temporal logic, PN is the Petri Net and $AC_{en}$ is the set of evolving actions for adding, deleting or modifying navigation restrictions.

**Fig. 4.** a) A Petri net from the $CS_p$, b) Petri net from $CS_p$ and one navigation restriction $RT_n$

The Navigation System models evolution using predicate temporal logic. It provides a meta-level with evolution actions which manage and change the navigation restrictions. Navigation restrictions can be added, deleted or modified, and the meta-restrictions of these operations can be established. In a similar way to the Knowledge System, the consistency must be guaranteed during the evolution of the Navigation System. In this system, changes can be produced in the navigation restrictions, $RT_n$, defined by the author, and therefore, in the PN obtained from them. For instance, from the conceptual substructure $CS_p$, the Navigation System can produce the Petri net of the figure 4a. The author can add more restrictions: “the reader can only reach the $Portugal.map$ item if he has visited the $Countries.cities$ item”. Then the Navigation System must generate a new Petri net –figure 4b-. Of course, all limitations are not possible, for instance, all items selected in $CS_p$ have to be reached and all conceptual associations have to be fired –the system must verify this meta-restriction-.

### 4.3 Learning System

The last element of our model is the Learning System, which modifies navigation by taking into account the type of information that the reader wants to achieve -the goals of the reader- and/or knowledge that he wants to acquire or learn –achievements-. Now, we are beginning to work in the development of this System as we wanted first to have a prototype of the model.

### 5 The SEM-HP Systems Functionality and Evolution

Up to now we have described the pieces of information that our model can use in building a document, the properties that characterise them, their content, the concep-
tual structure and the restrictions that control the structuring and navigation process. In addition, the set of Evolving Actions and their preconditions, the Restrictions, provide conscious support for every one of previous components. Different formalisms – the most suitable for each system- are used in order to specify the evolving actions and their meta-restrictions.

The author, an expert in a domain, represents his complex domain/s of knowledge using the Memorisation Subsystem. He/she creates his concepts, $C$, and associations between concepts, $A_c$. This knowledge will be used in characterising the different information items II; the author associated these items with concept/s $A_i$.

As a result, he/she builds the Conceptual Structure CS. In addition, the author defines its restrictions in order to guide the constructions of the CS. During this process, the Memorisation Subsystem must always guarantee its consistency. Two aspects of this system can change, the CS –the graph- and the restrictions defined by the author. Graph Theory is used to represent the evolving actions of the graph and their associated meta-restrictions. Changes in restrictions defined by the author, $RT_a$, must be defined by means of meta-restrictions.

When the author changes the CS –add, delete or modify a concept, item or association- the system must check:

1. The CS verifies the restrictions defined by the system and the associations satisfy the set of restrictions defined by the author. The RT acts as a set of restrictions for the actions, only if the action matches these restrictions, will it be carried out -internal propagation of changes-.
2. The subgraph used by the Presentation Subsystem, CS$_p$, is consistent with changes in the CS. If a concept or association has been deleted in the CS, the PS must also delete this concept or association in the CS$_p$ -external propagation of changes-.

When the author redefines –add, delete or modify- one associative restriction $RT_a$, the system must check:

1. The set of axioms about associations is valid, by means of predicate temporal logic.
2. The CS verifies the new set of restrictions, using graph theory. The system must detect the associations that do not satisfy one or more restrictions and delete them -internal propagation of changes-.
3. The CS$_p$ –the subgraph selected by the PS- verifies the new set of restrictions by means of graph theory. The system must detect the associations that do not satisfy these restrictions and delete them -external propagation of changes-.

In addition, the author can select a particular subgraph CS$_p$ from one Conceptual Structure CS using the Presentation Subsystem. In a similar way to the Memorisation Subsystem, the consistency must be guaranteed during the evolution of the Presentation Subsystem. In this system, changes can be produced in the subgraph selected CS$_p$. When the CS$_p$ is changed –the author select another set of concepts and associations- the subsystem must check:

1. The CS$_p$ verifies the restrictions defined by the system and the associations satisfy the set of restrictions defined by the author.
2. A new view or presentation is defined. In this case, the author must define again the navigation restrictions. This change is not a real evolution, the author is designing a new view of the information and, therefore, new navigation possibilities, but if these possibilities are defined in an incremental way, the system can aid the author in the design process -external propagation of changes-. 
Finally, the author defines their navigation restrictions $RT_n$ and the Navigation System must guarantee the consistency again. When the author redefines –add, delete or modify- a navigation restriction, $RT_n$, the system must check:

1. The set of restrictions that establish the order of navigation is consistent. Predicate temporal logic is used to specify the evolution operations over the restrictions, and their associated meta-restrictions.
2. The navigation restrictions have changed. Changes in a restriction can imply the modification of other restrictions. The PN based on the navigation restrictions must evolve, generating it again -internal propagation of changes-.

---

Fig. 5. Evolution: the evolving actions and their propagation.

To sum up, restrictions defined by the system, $RT_s$, or by the author, $RT_a$, are associated to the conceptual structure $CS$ (1). Evolution can be carried out in the conceptual structure, $CS$ (5), in $RT_a$ by means of predicate logic (6) and in $RT_n$ using predicate temporal logic (8). When $RT_a$ is modified $CS$ could also change (7). PN evolves being reconstructed from $RT_n$ (4). The evolution in the Memorisation Subsystem is also propagated to the Presentation Subsystem (2) and, later, to the Navigation system (3).

6 Conclusions and Further Work

Traditional hypermedia reference models shows that they are not able to represent the development, maintenance and navigation processes of an information system in continuous evolution. We have proposed a SEM-HP model composed of some interrelated and interacting systems: Knowledge –made of Memorisation and Presentation Subsystems-, Navigation and Learning, where an explicit representation of the provided knowledge is carried out.

Each System redefines or restricts the knowledge base -which is stable for the reader but dynamic for the author or authoring tool- provided by the Memorisation Subsystem by means of a set of Restrictions. In addition, the SEM-HP model supports different formalisms –graph theory and temporal logic-, which allow the specification of the evolving actions and the propagation of the changes in order to maintain the integrity of the systems. The Learning System is optional but, if present, it offers an optimisation of the knowledge acquisition process, which is very useful in educational systems. The explicit representation of the semantic structure drives the development,
maintenance and navigation processes of information systems. Consequently each system, basing on semantics, evolves –restructures the knowledge base- and makes the rest of the systems evolve.

Using the SEM-HP model and its specification formal, we are working in the construction of a prototype in Java and XML. In the near future we will improve the model specifying and formalising the Learning System.

References

Appendix: Glossary

1. An **Information Item** is any piece of identified information, which represents a conceptual unit in the information system. Each information item has a set of properties describing the type and functionality of the information it contains.

2. A **Property** of an information item is an associated attribute which describes the type, function and behaviour of the information that the information item contains.

3. A **Concept** is an idea, thought or abstraction which can be labelled by the author in order to make explicit his knowledge and understanding, i.e. a concept is a labelled idea.

4. A **Conceptual Domain** is the set of concepts to which the different information items in a hypermedia system may refer.

5. An **Information Domain** is the set of information items identified by concepts belonging to a certain Conceptual Domain.

6. An information item may play different **Roles** in the context of an information system. From the author’s point of view, an item may play a certain role in the context, but for the reader, it follows a link with the aim of reaching a certain type of information about a specific concept.

7. The **Specialisation Level** or version is a property of an information item that determines the level of specialisation of the information contained in the item.

8. A **Conceptual Structure CS** of a Conceptual Domain is a graph of labelled concepts which maintains information about a) associations between concepts, and b) associations between concepts and information items.

9. A **Reference Conceptual Association** is a labelled association between two concepts, members of a Conceptual Domain.

10. A **Dependency Conceptual Association** is a labelled association that is independent of the considered Conceptual Domain {partOf, kindOf and isA}.

11. A **Concept Environment** is the set of concepts that are related to a specific concept.

12. **Restrictions** are the set of conditions or rules that constrain the conceptual and information associations. They guarantee the consistency of the different artefacts of the Systems and carry out the function of preconditions to evolving actions.

13. An **information item framework** is the set of restrictions that holds when an item is achieved. It limits or constrains the set of information items that can be further associated with it.

14. An **evolving action** is the set of operations that can change the artefacts and restrictions of the different Systems. An evolving action must verify a set of restrictions and, in this way, guarantee the consistency of the System and carry out propagation of the change inside and outside the System.
15. The **Memorisation Subsystem** is a subsystem of the Knowledge System. It establishes the raw material used in building the hypermedia system. It includes two main components: information items and conceptual structure. Other basic components of this subsystem are the dictionaries of concepts, reference associations between concepts, and roles.

16. The **Presentation Subsystem** is one of the subsystems of the Knowledge System of a hypermedia system. It allows the selection of subsets of a Conceptual Structure in order to determine the hypermedia document which will be shown by means of the Navigation System.

17. The **Knowledge System** is one of the Systems that models a hypermedia system. It provides information items, their categorisation and the basic rules to establish their possible associations. It is made up by the Memorisation and Presentation subsystems.

18. The **Navigation System** is one of the Systems that models a hypermedia system. It constrains or filters the set of possible presentations with the aim of choosing a subset of them. It uses the restrictions provided by the Presentation Subsystem and the selfsame Navigation System.

19. The **Learning System** is one of the Systems that models a hypermedia system. It allows the evaluation and modification of the Navigation System, taking into account the goals and achievements proposed by the reader at each moment.

20. A **Goal** is a set of information items that the reader wants to achieve.

21. An **Achievement** is a set of pieces of knowledge that the reader wants to acquire or learn. They can be defined using the Conceptual Structure.

22. A **Hypermedia System** under a SEM-HP model based on systems is a set of interconnected and interacting systems called a Knowledge System -composed by the Memorisation and Presentation Subsystems-, a Navigation System and a Learning System, which allows: a) easy and flexible development and maintenance of hypermedia documents, b) representation of the conceptual structure and dependencies between them, c) more than one representation of the information system –of a set of possible representations - and, d) dynamic navigation where multiproposal navigation with structural contextualisation is possible.

23. A **Hypermedia Document** or Hyperdocument built with a SEM-HP model is a subset of information items and possible associations between them determined by the Navigation System according to a set of restrictions which it verifies at each moment.
Optimization Problems Categories

Liara Aparecida dos Santos Leal¹, Paulo Blauth Menezes², Dalcidio Moraes Claudio¹, and Laira Vieira Toscani²

¹ Mathematics Department
PUCRS – Porto Alegre, Brazil
{liara, dalcidio}@mat.pucrs.br
http://www.mat.pucrs.br/
² Computing Institute
UFRGS – Porto Alegre, Brazil
{blauth, laira}@inf.ufrgs.br
http://www.inf.ufrgs.br/

Abstract. This work presents a categorical approach to cope with some questions originally studied within Computational Complexity Theory. It proceeds a research with theoretical emphasis, aiming at characterising the structural properties of optimization problems, related to the approximative issue, by means of Category Theory. In order to achieve it, two new categories are defined: the OPT category, which objects are optimization problems and the morphisms are the reductions between them, and the APX category, that has approximation problems as objects and approximation-preserving reductions as morphisms. Following the basic idea of categorical shape theory, a comparison mechanism between these two categories is defined and a hierarchical structure of approximation to each optimization problem can be modelled.

1 Introduction

Certainly a very important point in the computer systems design and analysis is concerning the computational complexity. Since the introduction of the NP-completeness theory due to Cook [6], in the early 1970’s, a wide variety of problems from mathematics, computer science, operational research and general engineering are now known to be NP-complete.

When it comes to NP-hard optimization problems, there is also another parameter of interest. An optimization problem can be defined as the task to maximize or minimize some objective function, over a feasible solutions space. Generally these solutions can not be efficiently searched due to their too large number, in such way that finding an optimum solution results a computationally intractable problem [10]. In many cases, a solution close to the exact solution can be satisfactory. However, in practice, optimization problems are used to presenting diverse behavior related to approximation degree, could be classified highly approximable until it is proved impossible to approximate [8]. Various sorts of approximation are presented by Hochbaum [13].
The notion of *reductibility* between optimization problems allows to formally establish the meaning that one optimization problem is harder in approximation than other, inducing a partial order among problems in the same approximation class. A complete problem to a class is formally equivalent to that of a maximum problem with respect to that order \[2\]. According to Blum at al \[4\], reductibility allows the identification of complete problems, which are universal to each class. Their universality amounts to the fact that the existence of an algorithm for one of them would provide polynomial time algorithm for all problems in that class. Thus, the fundamental “P=NP?” question reduces to ascertaining the completeness. In the approximation context, Trevisan \[19\] considers that a reduction must preserve approximation in such way that some relations between quality measures of algorithms are satisfied.

This work presents a categorical approach to cope with some questions originally studied within Computational Complexity Theory, focusing to those aspects which could be named “qualitative” or “structural”. The Structural Complexity’s goal is to classify the problems in such way that it obtains an hierarchy in relation to the intrinsic problem difficulty degree. The basic idea follows that one presented by C. Rattray \[17\] establishing that the only structure that an object has is by virtue of its interaction with other objects, besides that, in any approximating situation, the approximations are what encode the only information that the system can analyse.

Nowadays it is well known that Category Theory is considered one of the most worthy tools in Scientific Computation, specially in the Theoretical Computer Science field. According to Asperti and Longo \[1\], one of the paramount aspects justifying the use of the Category Theory is the basic premise that every kind of mathematically structured object comes equipped with a notion of transformation or construction acceptable, that is, a morphism preserving the structure of the object. This same idea is powered in \[16\], who identifies five “basic doctrines” aimed at providing an initial intuition of the elementary categorical concepts, which are applied from the notion of a system as a category.

In this paper, a categorical view for combinatorial optimization problems is presented in such way that the notion of *reduction* from a problem to another one appears, naturally, in the conceptual sense of *morphism* between two objects. It proceeds a research with theoretical emphasis, aiming at characterizing the structural properties of optimization problems, related to the approximative issue, by means of Category Theory. In order to achieve it, two new categories are defined: the OPT category, which objects are optimization problems and the morphisms are the reductions between them, and the APX category, that has approximation problems as category objects and approximation-preserving reductions as morphisms.

In this context, basic properties of approximative algorithms theory have been investigated in order to better understand system approximation and design with respect to optimization problems area. Following the basic idea of categorical shape theory due to Cordier and Porter \[7\], a comparison mechanism between these two categories is defined and a hierarchical structure of approxi-
Optimization to each optimization problem can be modelled. Category theory is likely to be useful in providing frameworks within which to explain, for instance, basic notions such as “completeness”, “approximation schemes” and “complexity hierarchies”. Also, the idea of defining an optimization problem through a functional relation has proved to be convenient since, at first sight, the OPT and APX categories behave somehow as the slice category – a particular kind of comma category. The importance of this result is due to the fact that the comma category presents a desirable topos structure.

This paper is organized as follows. After presenting an introduction to the complexity of optimization problems and to the classification of NPO problems with respect to approximability, in section 3 category theory is presented as a suitable mathematical foundation to deal with the structural aspects of optimization problems. In section 4 and 5 are introduced the optimization and approximation categories. Finally, in section 6, the connections with categorical shape theory are presented.

A preliminary and short version of this paper appeared in [14].

2 Preliminaries

We assume that the basic concepts of computational complexity theory are familiar. We are following the notation of Garey and Johnson [10], which is universally accepted. Their book is well-known as a very good reference on the theory of NP-completeness, presenting a rich compendium of NP-complete decision problems. It also provides a great introduction to the area of approximation algorithms, although it is quite a bit outdated in this subject. Fortunately, the book by Ausiello et al [2] presents a similar compendium containing valuable information on the approximability of more than 200 problems, and being updated continuously via the WWW [9]. For a better understanding of most of the complexity definitions used in this paper we refer to one of the books on the subjects (see, for example, [2,15,5]). We now give some standard definitions in the field of optimization and approximation theory.

2.1 Optimization Problem

Here we present a functional definition to optimization problem, in order to characterize further properties by a categorical approach.

Definition 1. An optimization problem \( p \) is a triple \( p = (I, S, Opt) \), with

\[
Opt : I \times S \rightarrow Z^+
\]

where \( Z^+ \) is the positive integers set, and

1. \( I \) is the set of instances for \( p \);
2. \( S \) is the set of feasible solutions for \( p \);
3. Opt is the composite function between:
   – the measure function \( m : I \times S \rightarrow P(Z^+) \), which, for every instance \( x \in I \), maps a positive integer number \( m_x(y) \) to each feasible solution \( y \in S_x \), and
   – the objective function \( \text{Obj} : P(Z^+) \rightarrow Z^+ \), such that
     \[
     \text{Obj}(M_x) = m^*(x)
     \]
where \( M_x = \{m_x(z) \mid z \in S_x \} \in P(Z^+) \) for each instance \( x \in I \), and \( m^*(x) \) stands to the optimum measure to the instance \( x \).

The figure 1 illustrates this definition.

Fig. 1. Optimization Problem

2.2 NPO Class

On the analogy of the theory of NP-completeness, it there has been more interest in studying a class of optimization problems whose feasible solutions are short and easy-to-recognize. To this aim, suitable constraint have to be introduced.

Definition 2. An NP optimization problem \( p \) is a triple \( p = (I,S,\text{Opt}) \), such that
1. The set of instances $I$ is recognizable in polynomial time;
2. Given an instance $x$ of $I$, all the feasible solutions of $x$ belonging to the set $S_x$ are short, that is, a polynomial $p$ exists such that, for any $y \in S_x$, $y \leq p(x)$.
Moreover, it is decidable in polynomial time whether, for any $x$ and for any $y$ such that $y \leq p(x)$, $y \in S_x$.
3. The objective function $m$ is computable in polynomial time.

**Definition 3.** The NPO class is the set of all NP optimization problems, and the PO stands to the class of NPO problems that admits a polynomial algorithm to find their optimum solution.

Similarly to “P=NP?” question, also it is not known if “PO=NPO”.

By means of the notion of reductibility (See **Definition 17**) between optimization problems it is possible to define hardness to NPO class.

**Definition 4.** Given a reduction, an NPO problem $p$ is said to be NP-hard respect to that reduction, if for all NPO problems $p'$ we have that $p'$ reduces to the problem $p$.

### 2.3 Performance Ratio and Relative Error

In order to define classes of NPO approximation problems we need some notions of quality measures to a feasible solution. There are many of them, and the most widely applied are performance ratio and relative error.

**Definition 5.** Let $p$ be an NPO problem. Given an instance $x$ and a feasible solution $y$ of $x$, the performance ratio of $y$ with respect to $x$ is defined as

$$R(x, y) = \min \left\{ \frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)} \right\}$$  \hspace{1cm} (1)

**Definition 6.** The relative error of $y$ with respect to $x$ is the ratio

$$E(x, y) = \frac{|m^*(x) - m(x, y)|}{\max \{m^*(x), m(x, y)\}}$$  \hspace{1cm} (2)

Of course a strict relationships exists between performance ratio and relative error of a feasible solution. It has that

$$E(x, y) = 1 - R(x, y)$$  \hspace{1cm} (3)

Both the relative error and the performance ratio range between 0 and 1: the relative error is as close to 1 (and the performance ratio is as close to 0) as the feasible solution is far from the optimum one.
2.4 Approximation Algorithm

We consider only approximation algorithms with a performance guarantee, that is, in this approach it is required that, for all instances of the problem, the performance ratio of the solution found by the algorithm is, at least, bounded by a suitable function. In particular, the most interesting cases are those in which this function is a constant.

Definition 7. Let \( p \) be an NPO problem and let \( A \) be an algorithm that, for any instance \( x \) of \( p \), returns a feasible solution \( A(x) \) of \( x \). Given an arbitrary rational \( \varepsilon \in (0, 1) \), we say that \( A \) is an \( \varepsilon \)-approximation algorithm for \( p \) if, for any instance \( x \), the relative error of the feasible solution \( A(x) \) with respect to \( x \) verifies the following inequality:

\[
E(x, A(x)) \leq \varepsilon
\]  

(equivalently, \( A \) is an \( \varepsilon \)-approximation algorithm if \( R(x, A(x)) \geq 1 - \varepsilon \))

2.5 Approximation Scheme

It was early observed that NPO problems display different kinds of behavior with respect to approximation:

- Some problems exist that are not approximable for any rational \( \varepsilon \).
- Some problems exist that are approximable for some rational \( \varepsilon \).
- Some problems exist that are approximable for any rational \( \varepsilon \).

To the latter the infinite sequence \( \{A_\varepsilon\} \) of approximate algorithms is called a polynomial approximation scheme. However, the better is the approximation, the larger may be the running time. In most cases, it can approach the optimal solution of a problem arbitrarily well, but at the price of an increasing computation cost. In other cases, it may construct approximation schemes whose running time is polynomial both in the size of the instance and in the inverse of the required degree of approximation.

Definition 8. Let \( p \) be an NPO problem. An algorithm \( A \) is said to be a polynomial approximation scheme for \( p \) if, for any instance \( x \) of \( p \) and for any rational \( \varepsilon \in (0, 1) \), \( A(x, \varepsilon) \) returns a feasible solution whose relative error is at most \( \varepsilon \).

Definition 9. A polynomial approximation scheme \( A \) is said to be a fully polynomial approximation scheme for \( p \) if its time complexity is polynomial both in the size of the instance \( x \) and in the inverse of \( \varepsilon \).

2.6 Approximation Problem

A similar approach for providing a characterization for the optimization problems can be applied to approximation problem. In this case we have modified the set of measures from the positive integers set \( Z^+ \) to the set of rational intervals \( IQ \), in order to encapsulate the possible error from the approximate solution.
Definition 10. An approximation problem $p$ is a triple $p = (I, S, Apx)$, with

$$Apx : I \times S \rightarrow IQ$$

where $IQ$ stands to the set of rational intervals, and

1. $I$ is the set of instances for $p$;
2. $S$ is the set of feasible solutions for $p$;
3. $Apx$ is the composite function between:
   - the measure function $m : I \times S \rightarrow \mathcal{P}(\mathbb{Z}^+)$ defined early.
   - the objective function $Obj : \mathcal{P}(\mathbb{Z}^+) \rightarrow IQ$, such that
     $$Obj(M_x) = [m \ast (x) - \varepsilon, m \ast (x) + \varepsilon]$$

where $M_x = \{m_x(z) | z \in S_x \} \in \mathcal{P}(\mathbb{Z}^+)$, for each instance $x \in I$, and $\varepsilon$ is a rational positive number characterizing the performance quality of approximation.

See figure 2 below.

Fig. 2. Approximation Problem
2.7 Approximation Classes

The different behavior of NP-hard optimization problems with respect to their approximability properties will be captured by means of the definition of approximation classes, that is, classes of optimization problems sharing similar approximability properties and, if \( P \neq NP \), then these classes form a strict hierarchy whose levels correspond to different degrees of approximation.

**Definition 11.** \( APX \) Class: An NPO problem \( p \) belongs to the class \( APX \) if an \( \varepsilon \)-approximate polynomial-time algorithm \( A \) for \( p \) exists, for some \( \varepsilon \in (0, 1) \).

**Definition 12.** \( PAS \) Class: An NPO problem \( p \) belongs to the class \( PAS \) if it admits a polynomial approximation scheme.

**Definition 13.** \( FPAS \) Class: An NPO problem \( p \) belongs to the class \( FPAS \) if it admits a fully polynomial approximation scheme.

![Fig. 3. Approximation Classes](image)

As it is possible to see in the figure 3, clearly the following inclusions hold:

\[
NPO \supseteq APX \supseteq PAS \supseteq FPAS \supseteq PO
\]

3 Mathematical Foundations

This section gives a brief overview of important categorical concepts, in an informal way. But, why *category theory*?
After all that it has previously been said in the introduction, and according to Barr and Wells \[3\], there are various views on what category theory is about, and what it is good for. Category theory is a relatively young branch of mathematics stemming from algebraic topology, and designed to describe various structural concepts from different mathematical fields in a uniform way. Indeed, category theory provides a bag of concepts (and theorems about those concepts) that form an abstraction of many concrete concepts in diverse branches of mathematics, including computing science. Hence, it will come as no surprise that the concepts of category theory form an abstraction of many concepts that play a role in structural complexity.

According cited in \[16\], four achievements of the categorical development can be singled out: (i) concepts were clarified by phrasing them in categorical terms; (ii) uniform definitions of constructions were given in categorical terms; (iii) abstraction led to simplification and generalization of concepts and proofs; and (iv) complicated computations were guided by general categorical results. In the following, we briefly review the basic terminology and notation.

3.1 Category

Quoting Goldblat \[12\]: ”A category may be though of in the first instance as a universe for a particular kind of mathematical discourse. Such a universe is determined by specifying a certain kind of objects, and a certain kind of arrow that links different objects.”

Definition 14. A category $C$ is specified by a collection $\text{ob} \ C$, disjoint sets $C(A,B)$ for $A, B \in \text{ob} \ C$, and an associative operation $\circ$, such that

1. $(f \circ g)$ is defined for $g \in C(A,B)$, $f \in C(C,D)$ if and only if $B=C$;
2. for each $A \in \text{ob} \ C$, there exists $1_A \in C(A,A)$ such that $(1_A \circ f) = f$ and $(g \circ 1_A) = g$, whenever the composition is defined.

Members of $\text{ob} \ C$ are called $C$-objects and members of $C(A,B)$ are $C$-morphisms for $A, B \in \text{ob} \ C$.

3.2 Functor

A functor is a mapping from one category to another that preserves the categorical structure, that is, it preserves the property of being an object, the property of being a morphism, the typing, the composition, and the identities. Functors are the mathematically type of transformation between categories, and form a categorical tool to deal with ”structured” objects.

Definition 15. A functor $F: \mathbf{A} \rightarrow \mathbf{B}$ for the categories $\mathbf{A}$ and $\mathbf{B}$ maps $\text{ob} \ A$ into $\text{ob} \ B$ and sets $A(A,B)$ into $B(FA,FB)$ such that it preserves

1. units, ie. $1_{FA} = F(1_A)$, for each object of $\mathbf{A}$;
2. composition, ie. $F(f \circ g) = (Ff \circ Fg)$, whenever $(f \circ g)$ is defined.
3.3 Comma Category

**Definition 16.** Let $C$ be a category, and $A$ any object of $C$. The **comma category** $C \downarrow A$ is the category of objects over $A$ such that it has $C$-morphisms with codomain $A$ as objects, and as morphisms from $f : B \rightarrow A$ to $g : C \rightarrow A$ the $C$-morphisms $k : B \rightarrow C$, where $g \circ k = f$.

3.4 Universality

The notion of *universality* is fundamental to the category theory. According D. Ellerman [11], the foundational role of category theory is to characterize what is important in mathematics by exhibiting its concrete universality properties, not to provide some alternative construction of the same entities. The concrete universal for a property represents the *essential* characteristics of the property without any imperfections, and category theory provides the concepts to isolate the universal instance from among all the instances of a property. All the objects in category theory with universal mapping properties such as *limits* and *colimits* (see, for example, [1]) are concrete universals for universal properties. Thus the universal objects of category theory can typically be presented as the limit (or colimit) of a process of filtering out to arrive at the *essence* of the property.

4 Optimization Problems Category

In this section a categorical approach to optimization problems is presented in such way that the notion of *reduction* from a problem to another one appears, naturally, in the conceptual sense of *morphism* between two objects. Reductibility provides the key-concept to this approach. The recognition that the only structure that an object has is by virtue of its interaction with other object leads to focus on structural aspects of optimization problems.

The introduction of an appropriate notion of reductibility between optimization problems allows to formally state that an optimization problem is as hard to approximate as another one. In particular, the notion of approximation-preserving reductibility orders optimization problems with respect to their difficult of being approximated. *Hard problems* are the maximal elements in a class, with respect to this order, and capture the *essential* properties of that class. In this sense, NP-hard problems are *universal* to NPO class. In the following it will be clear the convenience of a functional representation of a optimization problem.

**Definition 17.** A reduction between the optimization problems $p = (I, S, Opt)$ and $q = (I', S', Opt')$ is a pair of functions $(f, g)$, where $f : I \rightarrow I'$ and $g : I' \times S' \rightarrow I \times S$ are such that the diagram in the figure commutes:
Definition 18. The optimization problem category OPT has NPO optimization problems as objects and reductions between optimization problems as morphisms.

OPT is really a category since reductions are computable functions satisfying the reflexive and transitive properties. Besides, the OPT category behaves somehow as the slice category — a particular kind of comma category [3]. The importance of this result is due to the fact that the comma category presents a desirable topos structure [12]. A topos is a cartesian closed category with an special object called subobject classifier. Topos theory can be viewed as a categorically-formulated generalization of set theory to abstract sheaf theory. This is an interesting direction to further work.

5 Approximation Problems Category

After we have given a first step in the categorical approach with the definition of the optimization problems category, it is natural to pursue in this direction, aiming at extending to approximation problems. Now, considering the notion of approximation-preserving reduction as morphisms between approximation problems, it is possible to define an approximation problems category.

Definition 19. An approximation preserving reduction is defined as a reduction between optimization problems adding some conditions that guarantee some property related with approximation.

Definition 20. The approximation problems category APX has approximation problems as objects and approximation preserving reductions as morphisms.

Analogously to OPT category, is easily verified that APX is really a category.
6 OPT Category × APX Category

Very often we wish to find a mathematical model of a structure in order to explain its properties and predict its behavior in different circumstances. Related to approximability issue to optimization problems, it is likely that the categorical shape theory would be such a model. It does provide a comparison mechanism to establish the meaning of an approximation system, identifying the universal properties in the category theory sense, in order to describe how an object “best approximating” another object. This section has been motivated from the previous work due to C. Rattray [16,17]. The basic idea of categorical shape theory [7] is that, in any approximating situation, the approximation are what encode the only information that it can analyze.

In the context of categorical shape theory it has:

1. a category $B$ of objects of interest;
2. a category $A$ of archetypes or model-objects;
3. a “comparison” of objects with model-objects, ie. a functor $K: A \rightarrow B$.

**Definition 21.** Given category $A$ of archetypes, category $B$ of objects of interest, and a comparison $K: A \rightarrow B$, an approximation to an object $B$ in $B$ is the pair $(f, A)$, where $A$ in $A$ is an archetype and $f: B \rightarrow KA$.

A morphism between approximations $h: (f, A) \rightarrow (g, A')$ is a morphism $h: A \rightarrow A'$ of the underlying archetypes, such that $K(h) \circ f = g$, ie. the triangle

commutes.

Approximations with their morphisms form a category $B \downarrow K$, the slice category of $K$-objects under $B$. The cone-like form of the morphisms in $B$ giving the approximations for some object $B$, suggests that taking the limit object of the diagram would result in an archetype $A^*$ “as near as possible “ to $B$. See figure 4 below.

The notion of “most closely approximates” is given by an universal object.
Definition 22. Let $K : A \rightarrow B$ be a comparison functor. An archetype $A$ of $A$ is said to be $K$-universal for an object of interest $B$ of $B$ if there exists an approximation $(f, A)$ to $B$ such that, for each approximation $(g, A')$ to $B$, with $A'$ in $A$, there exists a unique morphism $h : A \rightarrow A'$ in $A$ with $g = K(h) \circ f$.

Definition 23. Category $A$ is said to be $K$-universal in $B$ if every object of interest of $B$ has a $K$-universal archetype in $A$.

Categories OPT and APX are specialization of categorical shape theory, in such way that the OPT category stands to the category of objects of interest $B$, APX category stands to the category of archetypes $A$, and $K : APX \rightarrow OPT$ is a comparison mechanism related to a approximation method (for instance, a relaxation). Through this theory it is possible to identify the best approximation to a optimization problem $B$, if it exists. In this context, the APX category is $K$-universal in OPT, unless $P=NP$. In this context, other properties are been investigated.

7 Conclusions

The main objective of this work is to develop a formal theory to approximative algorithms, considering them as a feasible alternative to intractable problems in such way that integrates the Structural Complexity’s conceptions to the fundamentals of suitable semantic model. The recognizing that the notion of reductibility between problems substantiate in a specialization process of Category Theory, led to an investigation on structural aspects of approximation classes through of categorical approach, focusing over the approximation preserving reductions between optimization problems. Structural complexity theory is often concerned with the inter-relationship between sets in a complexity class and inclusion relationships between complexity classes.

However it seems that an attempt of organizing all there results in a unified framework as general as possible is lacking. The aim of this paper is to make a first step in this direction. Starting from the observation that, intuitively,
there are many connections among categorical concepts and structural complexity notions, we have defined two categories: the OPT category of optimization problems and the APX category of approximation problems.

In order to establish a formal ground for the study of the properties of those categories, we also present some basic definitions, following the literature.

A comparison of the OPT and APX categories has been motivated from previous work by C. Rattray [16,17], based on categorical shape theory. The study that we have started in this paper is an attempt in this direction. Along the same line, we think that in order to establish connections among optimization problems and their approximability properties, it may be fruitful to find relationships with other results drawn from other approaches, at the same level of abstraction, such as the one developed in [18].

Another direction for interesting research would be to proceed to a more systematic study of both categories OPT and APX related to identifying elements to be a topos (or quasi-topos) structure.

Acknowledgments

This work was partially supported by: FAPERGS – Fundação de Amparo a Pesquisa do Rio Grande do Sul, PUCRS – Pontifica Universidade Catolica do Rio Grande do Sul and UFRGS – Universidade Federal do Rio Grande do Sul, Brazil.

References

Abstract. The different aspects of classical neural nets are treated here at the light of Systems Theory. First, we consider McCulloch-Pitts-Blum formalisms, which are regarded as biological counterparts of logical machines (automata). Systems Theory inspired the introduction of state transition and functional matrices, to treat problems of analysis, synthesis, stability and oscillations. The so called McCulloch Program I is completed by the development of systems technics to allow the neural synthesis of arbitrary probabilistic automata. Next, nets to integrate sensorial functions, as well as intermodal integration and effector action are considered, which corresponds to a level higher than the "molecular" neuron-like computing element. Finally, at the highest level, modules for the so called generalized robotic-like behavioral systems are presented, in line with the known McCulloch-Problem II. This includes a diagram for a behavioral system under a command and control subsystem of the type of the reticular formation of vertebrates.

1 Introduction

Though it is generally accepted that the key paper and the pioneer in formal neural nets is the ’45 McCulloch’s and Pitts ”A Logical calculus...” [1] it is at least surprising the little real use that the results in it has been made of strictly. Even when scanning McCulloch’s production afterwards, we find practically no use of the results except to illustrate concrete embodiments of theoretical neuronal circuits inspired by neurophysiology or at higher levels, but again here, just the properties of very simple neuronal units were into play. McCulloch complained that ”in my field the theory of Automata is becoming purely mathematical automata theory”. And that was and is true. Because the very nature of their paper, because its generality and because there are jumps in meanings assigned to equations and jumps of levels of descriptions, from microscopic to very global, the paper is of very little use for explaining real brains in formal terms. They actually did not explain brains; they explained logic in terms of neurons,and subsequently,automata in the same terms. That is, they provided for the first ”granular” or modular decomposition of automata and practically, gave rise to
the Algebraic Automata Theory, that went on its own. But as in any other formal representation, the meanings attached to the equations were always outside the theory.

Anyway this was a very interesting and fruitful way to proceed in the interplay of formal systems tools and neurophysiology, and that was what was done in the later developments of theory: to tend to a formal biological counterpart of automata and computers, proceeding from granular, biology-like representations (the so-called McCulloch Program 1).

The original theorems were, in no way, obvious and clearly demonstrated, that is, even since there is always a "waste" of formal neurons when synthesizing even simple operations, that waste was too large. For example, to perform a simple contrast detection of the type of the exclusive or, they were required two layers of neurons already. That is simply because the primitive formal unit was a linear machine followed by a threshold.

![Fig. 1. Illustration of interaction of afferents in McCulloch-Blum neurons.](image)

Interaction of afferents is the simplest logical formulation for the presynaptic inhibition found in Rana Pipiens by Shypperheyn. In essence, it consists that
fibers reaching a neuron bifurcate in a way that they may drastically inhibit other fibers to the cell, prior to the synapsis. In what follows we shall use a clearer description [2]. A simple illustration is in figure 1(a) where each fibre, when simulated, completely blocks propagation in the other, before it reach the synapsis. For \( t = 1 \), it is easily verified that the cells computes the "exclusive or" of \( x_1 \) and \( x_2 \). The problem is to synthesize any nonlinear logical function by allowing the interaction, which in fact corresponds to a new layer of logic. The systematic approach is illustrated in figure 1(b), where \( \alpha_i, \alpha_{ij}, \alpha_{ijk}, \ldots \) are the synaptic weights, and, \( \theta \) is the threshold.

The firing condition is then:

\[
\sum \alpha_i x_i + \sum \alpha_{ij} x_i x_j' + \sum \alpha_{ijk} x_i x_j' x_k' \geq \theta
\]

The number of degrees of freedom for a unit with a fixed threshold for \( M \) inputs fibers, is

\[
M + M(M-1) + M \left( \frac{M-1}{2} \right) + \ldots + M \left( \frac{M-1}{M-2} \right) + M = M 2^{M-1}
\]

which is larger that \( 2^M \), and therefore shows the redundant effect of the interaction of afferents.

The synthesis problems is then to find the minimal neurophysiological paradigm for a neuron computing an arbitrary function of their inputs.

The Blum-McCulloch [3] procedure is to assign a threshold to each minterm of the inputs, which the condition that minterm \( x'_1, x'_2, \ldots, x'_m \) must have a zero threshold. There are systematic ways to proceed in the synthesis. But more important are the implications of this apparent dendritic redundancy (or dendritic profusion).

From the logical point of view, the redundancy appears from the possibility that a neuron can compute a whole set of different logical functions as its threshold changes. That is, the limiting case of no-redundancy is when the neuron computes \( 2M + 1 \) different functions, including tautology and contradiction, which is the maximum number of logical functions that a neuron having a fixed anatomy can compute. The number of different thresholds is precisely \( 2^{M+1} \). But this is the more unreliable neuron, as we shall see.

In the fifties, McCulloch and Von Neumann were engaged in the problems of reliability. For neural nets, McCulloch and later Winograd and Cowan [4], treated the stability of the function of a neuron when its threshold changes. They try to solve the problem by a multilayered network with interaction of afferents (multilayered redundancy) or by the classical multichannel redundancy of Von Neumann.

As it can be seen, dendritic profusion provides for an additional mechanism for logical stability, since for all (or part) of possible threshold values the same logical function can be assigned. The two limiting cases (maximal functionality, minimal logical stability and viceversa) are illustrated in figure 1(c) and fig-
Figure 1(d) for the simplest case of the "exclusive or". It is also interesting how this type of arguments help to clarify the role of retinal lateral interaction cells \[5\].

In 1965, the neurophysiology group of MIT was after the problems of how more economically and efficiently can a network (granular if one wishes) store dynamic patterns of activity for later retrieval. The basic unit was the formal neuron with interacting afferents, capable of computing any logical function of its inputs.

First, there was the problem of what is the maximum number of oscillating modes than a net of \(N\) neurons can engage in. A mode of oscillation is a circular sequence of states for the net. Schnabel \[6\] found them to be, for a net of \(N\) neurons:

\[
N_0 = \sum_{k=2}^{2N} (k - 1) \binom{2N}{k}
\]  

(3)

a number which increases very rapidly with the number of neurons. In fact, it applies to any finite automaton of \(2^N\) states.

A second question was whether this astronomical number (for \(N\) a little large) could be stored and retrieved in-from a fixed anatomy. Da Fonseca showed that it is non practical and started to explore the potentiality of non-linear feedback shift registers as alternatives for dynamic storage \[7\]. His conclusions were practical in the sense that the corresponding networks had a much larger possibility to be embodied somewhere in the nervous systems. In fact, formal nets as they were at the mid sixties sowed such a logical potentiality that all deterministic and most probabilistic automata where particular cases, as it happened Da Fonseca’s dynamic storage models.

It was already felt that logical and computational systems tools available fell too short to provide a frame where to proceed from the nervous system description beyond perhaps sensorial systems and primary cortex. By the early seventies it was already quite well established that any behavior that can be expressed as a result of the interaction of one or various probabilistic or deterministic automata had a counterpart in a logically minimal network having a neurophysiological look of threshold and dendrites interaction. From the automata theory point of view, there were only one point missing, due to an error in a theorem on the synthesis of arbitrary probabilistic automata by McCulloch and Moreno-Diaz \[8\]. For a net of \(N\) probabilistic neurons it is not possible to synthesize an arbitrary probabilistic automaton, so it looked that other neurophysiological paradigm was behind the logical systems constructs.

2 Neuron-like Analysis and Synthesis of Arbitrary Automata

Formal net theory as it is contemplated as potential biological counterpart of logical machines, suffered from the sixties, the influence of automata theory, influence which has been both limiting and enriching. It has limited the scope
and practical applicability because the very nature in "extenso" of anything derived from automata theory to refer the real world. And enriching, because the new points of view permitted to state new questions to the theory.

The theory has stopped its vertical development, though lateral branches are still allow to grow to provide for concrete applications and perhaps for new formal theorems.

In a rather general sense, there is one type of questions that mostly matters to modular or granular brain theory, from top to bottom and even from bottom to top. These questions have to do with the "level of description". It is something like stablishing the "ceiling" from where to go down in the interpretation - explanation. Or in the way up being aware of the power of the tools that will allow to reach a certain height in the description.

For the case of the already classical theory of formal neural nets, the ceiling was apparent to be that of the formalisms of probabilistic arbitrary automata. Without considering in fuzzy representations for the moment, we shall consider, with quick notation, the way from cell bodies, thresholds, afferent and efferent interactions to probabilistic automata and viceversa.

Without paying any attention to the physiological significance of subjects such as temporal and spatial integration (linear or non linear, space-time equivalence, multiplicative effects that can be expressed by nets), we remind the following definitions:

1. A McCulloch-Pitts formal neuron (F.N.) is a computing device capable of a linear boolean function of its inputs followed by a thresholds and a delay.
2. A F.N. with interaction of afferents (F.N.A.) is a F.N. capable of computing any boolean function of the inputs.
3. A probabilistic F.N. is a F.N.A. where each input configuration has a probability \(0 \leq p \leq 1\) of firing it.

As mentioned before there are two types of general theorems for the theory.

1. Analysis Theorems (constructive)
   (a) An arbitrary neural net, of \(N\) deterministic neurons and \(M\) external inputs with any type of feedback, is equivalent to a functional matrix of \(2^N \times 2^N\), and therefore, to a deterministic automaton. There is a straightforward procedure to go from the functions performed by each neuron to the expression of the functional matrix.
   (b) The same for probabilistic neural nets with feedback, with respect to probabilistic automata.
2. Synthesis Theorems (constructive)
   (a) Any arbitrary deterministic automaton of \(2N\) states and \(M\) logical inputs lines is equivalent to a neural net of \(N\) neurons with feedback, and \(M\) external inputs. The neurons must have, in general, interaction of afferents. There is a straightforward procedure to obtain the net, given the functional matrix of the automaton.
   (b) There is not, in general, a probabilistic neural net which is equivalent to an arbitrary probabilistic automaton.
There are three main conclusions of this explicit connection between formal neurons and automata. First, there is a quick way to show that all possible modes of oscillation for a net of \( N \) neurons with feedback, though for \( N \) a little large, the connectivity of the net is very complicated. Obviously, a number of inputs lines, \( M \), is required such that \( 2^M \) equals or surpasses Schnabel’s number and the proof proceeds as follows:

- First, for each mode of oscillation, construct a transition matrix. States not involved in the oscillation are made to enter a transient ending in one state of the mode. Second, assign mutually exclusive input configuration to each transition matrix. Build the functional matrix and proceed to the neural net according to theorem
- The second conclusion is that, since linear or non-linear shifts registers for memory are particular cases of automata having relatively simple functional matrices, new effective and rapid ways for their synthesis were available.
- And third, there is the question of why negative theorem 2(b).

The fact of an axon-axonal interaction in real neurons at least at the level of neurons in the retina had became recognized by the early seventies, the interaction being much faster than that which involves the cell body.

![Diagram](image-url)

**Fig. 2.** Illustration of analysis and synthesis of arbitrary probabilistic classical neural nets.

As it happened with dendritic interaction in Blum’s formulations, axon-axonal interaction could account for the missing ”layer” which a minimal syn-
thesis theorem required for arbitrary probabilistic automaton. First, it was recognized that the limitations came for considering that firing probabilities for each neuron were independent. They are not, in general. It matters what type of dependence will produce the appropriate results more simply. It was found that some type of hierarchical dependence at the axonal level will solve the problem so that neurons higher at the hierarchy, influence the rest, not viceversa [9]. Figure 2 shows the ways in analysis and synthesis. Arrows to the right signify analysis theorems, and to the left, synthesis theorems. For each case the minimal theorems are constructive, that is, they provide for effective ways to perform the analysis and synthesis.

A much more relevant paper in what it refers to methodology in Brain Theory is Pitt’s and McCulloch’s ”How we know Universals...” [10], as some authors have pointed out. Alternative nets are however possible and more economical, still in agreement with anatomy, to generate visual and auditory invariants (such as chords). In these nets, the quasi-beginning and the quasi-end (temporal) of a chord are the logical prerequisites for homoteties in a spatial representation of time.

The basic idea is that there must be two paths for invariances, one which computes the invariant parameters from the ”apparitions” and a second which applies the parameters to those apparitions. That is, the net computes the size the apparition, and then normalizes it. The computations which are carried in parallel. Details on that are found in.

The results pertinent to granular automata are that the logical consequence of the prerequisites is a net having layers of linear (analog) computation followed by layers of local conditional branching (if-then-else), alternating in a double sequence.

These conclusions lead to a additional theorems for granular neuron-like decomposition of arbitrary automata. Let $LAC$ be a local linear analog computing layer and $LCB$ a layer of local conditional branching. The Pinoso Theorems are then stated as follows for generalized formal neural nets and consequently, for automata.

**Theorem 1.** For a Net of $N$ generalized neurons, with afferent interaction, without feedback there is a $LAC \rightarrow LCB \rightarrow LAC \rightarrow LCB$ structure that duplicates it. For one dimension, in the continuum, the function of neuron at site $y$, is, for a receptive field inputting $I(x)$, given through these four computational layers:

\[
\begin{align*}
LAC \ M(z) &= \int_x W(x, z) I(x) dx \\
LCB \ N(z) &= \mu [M(z) - \theta(z)] \\
LAC \ P(y) &= \int_z W(y, z) N(z) dz \\
LCB \ Y(y) &= \mu [P(y) - 1]
\end{align*}
\]
where $\mu$ is the step function, and $W$ the corresponding weights of the analog linear layers.

**Theorem 2.** For a generalized neural net with feedback there is also a $LC \rightarrow LCB \rightarrow LC \rightarrow LCB$ structure that duplicates it, but taking into account time effects.

The demonstration ends in:

\[
LAC M(z, t) = \int_x I(x, t)W(x, z)dx + \int_y O(y, t)W(y, z)dy
\]

\[
LCB N(z, t) = \mu[M(y, z) - \theta(z)]
\]

\[
LAC P(z, t) = \int N(z, t)W(y, z)dy
\]

\[
LCB Y(z, t) = \mu[P(z, t) - 1]
\]

where $O(y, t) = \int_0^t K(y, t)Y(y, t)dt$ and $K$ is the kernel for the temporal axonic effect of the “neurons” (perhaps a simple delay).

**Theorem 3.** Since any arbitrary automata is equivalent to a neural net with feedback plus a neural net without feedback, any automata is equivalent to a cascade of two-layered machines of structure as given by Theorem 1 and Theorem 2.

These theorems are constructive, i.e. provide for effective ways to go from one to another. Figure indicates the equivalences developed by Theorem 1, Theorem 2 and Theorem 3. For probabilistic or fuzzy machines, everything applies mutatis mutandis. They are illustrated in figure 3.

3 Nets to Integrate Sensory, Intermodal, and Effector Functions

3.1 Intermodal Transformations

Intermodal transformations refer to transformations between data from different sensory modalities or different representations (modes) for the same sensors modality. Under these conditions patterns belonging to one sensory modality could be, under circumstances, handled and “recognized” by computing structures which belong to a different sensory modality [11].

The first requisite is to transform the original sensory data to quasi-geometrical representation spaces. In artificial sensory systems we can think of a variety of instances of how a representation space can be constructed, for visual or for other input. From each representation, space mechanisms acting as “operators” on the space, provide for outputs carrying some type of decision or classifications which are required for the overall diagnosis of the sensory situation. This overall outputs enter a ”modal, or sensory integration” process, to produce that overall diagnosis,
Fig. 3. Illustration of equivalence layered computation classical neural nets and arbitrary automata.

by applying some decision rules where the "blanks" in the rules are to be filled with the data provided by each sensory modality. Intermodal transformations "acta" among the representation spaces, that is, interconnect them.

As for the central nervous system, it is clear that specific sensory pathways became multifunctional as they proceed upwards towards more central areas, in a way that such structures which are specific can, under circumstances be used by different sensory modalities. This is so in emergency situations, e.g. under traumatic lesions. In other words, functional specificity is less dogmatic and more circumstantial as more central zones are considered in a sensory pathway. This characteristic must be taken into account when the high degree of reliability of nervous tissue is to be explained.

A simplified representation of various medium level sensory cortex functions, as they are normally accepted in brain theory, is shown in figure. Here a typical process of sensory integration is represented,, which goals are the diagnostics of an overall external situation.

Sensory data from each modality are preprocessed in specific pathways, before they reach the cortex, in a way such that a translation to an specific language takes place. This language is the one understood by the processors at the primary sensory cortex. It is important that the messages being transmitted to the primary sensory cortex are already represented in a language which probably is much more complex that the usual pulse code normally accepted in neurophysiology. Primary sensory cortex classifies and labels the corresponding sensory
apparitions. This classification and labelling are a consequence of mapping in representation spaces and of applying classification and decision rules which are either acquired by learning or built-in. One of the implicit properties which are normally ascribed to primary sensory cortex processor is their almost total specificity, in a way such that practically no possible intermodal transformations take place, but just a posterior multimodal integration to converge in the overall diagnosis.

This is a consequence of admitting that the crucial mechanism to attain diagnoses convergence is a high level multimodal integration, with no previous intermodal talk. From the behavioral point of view, multimodal integration can be thought to be directed towards two essential goals: a) To accelerate and optimize in time and resources, the diagnostic and identification process, by means of the use of intermodal clues (that is, labels, which by learning or instinct, are associated to belong to a same externo-internal global situation). b) To separate classes of situations which are not separable by means of a single sensory modality, so increasing the discriminant power of the sensory system.

![Diagram for multimodal integration.](image)

**Fig. 4.** Diagram for multimodal integration.

The diagram of figure 4 fulfils the requirements corresponding to the above goal. Obviously, this diagram can be sophisticated by adding a series of refinements in the processors corresponding to a primary sensory cortex and to associative or integrating cortex, such as a sensory model of the environment, expressed in a more or less abstract language. Also, the existence of centrifugal (feedback) pathways can be included to adapt the resolution and subsequent
computational costs of the processors. The concepts of intermodal transformation permit, however, a qualitative change in said diagram.

If we assume that a lesion in the specific pathways happens, according to figure 4, the processor corresponding to the primary sensory cortex will became out of use, since there are not inputs. However, the experiments of J. Gonzalo show that is not strictly the case. That is, the practical absolute specificity of the processors in the primary sensory cortex tends to disappear in the case of lesion of the specific afferents, after a learning process. Also, the draw backs which result form primary sensory cortex lesions can be diminished after training, such that the eliminated function is partly assumed by the remaining cortical structures. The above can, in principle, be expressed in terms of a global behavioral goal, as it was done for multimodal integration. The assumption is as follows. The exists and adaptive sharing of the specific structures of different sensory modalities, such that, after learning intermodal information can be utilized to increase reliability and to tend to optimize the use of the structures which remain after lesions or when there is a low information input in a sensory pathway.

Fig. 5. Diagram for intermodal interactions to achieve reliability.

A possible systems solution is diagrammed in figure 5. Intermodal transformations (IT) occur between the processors specific to each sensory modality. There are two cases in which, computerwise, we may think of intermodal trans-
formations coming into play: 1) When there is an overflow of sensory information from a sensory pathway \((A)\) to their more central specific processors, and there are dimensions which are free (non utilized) in the representation spaces of other sensory modalities \((B, C)\). This overflow may appear as a consequence of a traumatic reduction in the specific processors functionality of \((B, C)\). The transformation proceed from \((A)\) to \((B, C)\). 2) When, because a traumatic lesion or circumstantial effects, there is a low sensorial flow from the sensory pathways of a modality \((A)\), which free dimensions in their representation space. These free dimensions can be occupied by data through transformations from other sensory modalities \((B, C)\). And also, the are computerwise possibilities to define intermodal transformations among representation or "features" spaces.

### 3.2 Modules for Some Robotics Systems

Warren McCulloch provided simplified block diagrams of the central nervous system under the demand by Sutro, of the Charles Stark Draper Laboratory of MIT, who at that time (1965) [12] leadered a project for a robot vision. L. Sutro provided "engineering" versions of them, under McCulloch supervision. In both cases, there was a very large amount of interconnection, and, for the robotic counterpart, there were clearly identified blocks containing cameras, associative computer, computer of effector sequences, effectors and other, all around a decision computer. But there were a number of brain theoretical (and robot engineering) questions about structures, communications protocols, nature of messages and mostly how we can imagine machines that performs those functions.

Actually, the description of the actual behavior of nervous system is very close globally to the problems faced by a generalized robot. The central issue is how sensing, perception, planning decision and action are, first, performed and thereafter integrated. The solution is not unique and could be broken down for specific tasks. But, obviously, we will like the system to do more, something like a generalized artificial robot to compare with the natural ones. Figure 6 shows our very much simplified version of the complex architecture of McCulloch’s and Sutro’s proposals, but which allows for some descriptions in terms of recent tools from artificial perception, artificial intelligence and robotics. The diagram includes the command and control reticular formation like subsystem.

First, the overall system presents a set of "modes of behavior" that mimic the accepted models of behavior of the vertebrates. The selections of a particular mode is performed by the command and control system, based mostly in present sensorial information (S.D.) and the status of the system. An external input (E.I.) is allowed from the external world (in practice from an operator’s console) to modulate the selection of a mode.

Information concerning the selected mode (M) is sent to the sensors which are to be tuned to optimize the sensors data acquisitions pertinent to the mode of action. It is also sent to the component labelled Planning in Present World Representation. Finally, the selected mode commands and controls the process of establishing goals according to the mode, the process of planning and the execution of the plan, by taking into account continuous highly processed sensory
Fig. 6. Diagram for a behavioral system under a command and control subsystem of the reticular formation type.

data (S). Actualized world representations are sent back through W.R. lines when the mode changes.

There are direct lines from sensors to effectors (line R) which are equivalent to reflexes. Line E provides for high level instructions to the effectors according to the plan of action, which are to be decoded into concrete motor-effector actions.

The basic function of a command and control systems is to commit the whole system to one overall mode of behavior belonging to a not very large set. This is what enables it to behave as a well integrated unit instead of a loose connection of separate sensors, effectors and processors. In this sense, a command and control computer is a close paradigm to the operation of the reticular formation in vertebrates. All modes of behavior are mutually exclusive for such a computer. First it receives relatively unprocessed information from all the sensors situated in sensory and effector subsystem. Second, it gives signals to which control, tune and set the filters of all external inputs. In McCulloch words, ”this is the structure that decide what to look and having looked, what to heed”. It also controls all the information flow from and to higher level computers.

From a structural point of view, the command computer must have a modular architecture, or, at least, it must simulate it. The basic idea is that a set of computing units (C.U.) is such that each computing module receives information only from a reduced part of the overall, little processed sensory inputs.

Each computing unit is capable of both general diagnostics about overall input situations and of specialized diagnostic according to the values of a concrete subset of the input lines.
A crucial point is that a consensus of the diagnostics—which corresponds to the selection of a single mode behavior—must be reached by the computing units in a relatively short time. This requires a very strong crosstalk among the computing units, which is a peculiar feature of the so-called cooperative processors. There are two basic properties of the computing modules, that can be stated easily by means of the terminology common in expert systems.

In fact, we can look at the computing units as if they were simplified expert systems working on their own data bases and with their own inferences engines on their specialized domain sensory inputs. But they are capable also of giving up before the evidence in diagnostics by other units which show to have more relevant information for the case. This “giving up” must be understood in the sense of a recruiting of the rest of the modules by those having more confidence about their diagnostics. As it was stated by McCulloch, modules having the information pertinent to the case “cry louder”, and doing so, they recruit the rest. The result of this strong crosstalk must be that the system converges in one mode, in the sense that practically all the units decide the same mode of behavior, though with perhaps different degree of confidence.

Modularity and division of expertise, with overlapping, among the computers units are the necessary addenda to achieve convergence. This architecture is supposed to provide for two main goals: first, to speed up the decision process by which a mode of behavior is selected; second, the system is supposed to present high reliability, in such a way that it will arrive into an appropriate consensed mode, even when some of the expert units are destroyed [14].

This second aspect, that is, the reliability intrinsic to distributed expertise, precludes any decision based upon a single majority organ, because its malfunction will imply total inoperativity. That is, the conclusion that a consensus has been reached cannot be the output of any special testing unit receiving its inputs from the expert unit. Instead, the decided modes must be appropriately labelled according to their procedence to prevent mixing, and be present in a non-computing structure, that is, a set of wires—or axons—or, in other words, in a kind of decision bus.

From this, it becomes clear that reaching a rapid consensus in the mode of behavior at the command and control computer is absolutely necessary for the rest of the system to operate coherently, because otherwise, the various higher and lower subsystems to be controlled, will have a high probability to pick up operating instructions from the decision bus, which belong to different exclusive modes of behavior, such that a kind of neurotic situation will be created.

For the world representation processes, both in files and the one presently used it is needed a multisensorial environment mapping prior or after transforms. There are two basic ways to structure multisensorial information which in turn, admit different levels of representation, from geometric to highly symbolic, at least for artificial systems. These two ways have a correlate with representations which tend to be optimal for discriminating among environmental patterns or representations to optimize the acquisition of clues for actions. These correspond to:
– Integrated representation, both at low level of acquisition and as high levels of processing sensory data.
– Step by step representation, in which integration only occurs at high levels -that is in symbolic or language- structures.

In other words, and putting aside for the moment all natural systems, we may either represent the sensed environment by means of a multidimensional space where all sensory modalities are present with their own resolution at low level, while all high level processing is performed directly in this space, or we can construct a high level intersensorial representation space by previously extracting properties, classifying and labelling each sensory modality separately.

The problem of the realization of neuron-like nets to cover the decision mechanisms which tend to drive a whole system into one mode of behavior is still open, and it is a fruitful area for the application of system theoretical methods [15].

References


Biologically Based CAST-mechanism for Visual Motion Analysis

Alexis Quesada-Arencibia, Roberto Moreno-Díaz jr., Miguel Aleman-Flores

Departamento de Informática y Sistemas, Universidad de Las Palmas de Gran Canaria, Campus de Tafira, E35017, Las Palmas, Spain
{aquesada,rmorenoj,maleman}@dis.ulpgc.es

Abstract. In this paper we present a working vision system for estimating size, location and motion of an object by using a set of randomly distributed receptive fields on a retina. The approach used here differs from more conventional ones in which the receptive fields are arranged in a geometric pattern. From the input level, computations are performed in parallel in two different channels: one for purely spatial properties, the other for time-space analysis, and are then used at a subsequent level to yield estimates of the size and center of gravity of an object and the speed and direction of motion. Movement analysis refining is implemented by a lateral interaction (spatial) and memory (temporal) schemes in which direction and speed are used to build a trajectory. The different parameters involved (receptive field size, memory weighting function, number of cells) are tested for different speeds and the results compared, yielding new insights on the functioning of the living retina and suggesting ideas for improving the artificial system.

1 Introduction

The ability to detect an object’s size, location and movement is essential for a visual system in either a biological or man made system [1], [2]. In man made systems these parameters are required for robotics, intelligent operations and many other applications [3]. In order to obtain food, to escape out of the sight of a predator, to follow the individuals of the species or any other vital function, the location of the objects of the environment, the determination of their sizes and the identification of their trajectories are essential aspects of the visual systems of many animal species. The formal model used here for estimating size, center of gravity, direction and speed, is based on an idea by Leibovic [4], analogous to the Montecarlo methods [5], and uses a set of randomly distributed receptive fields on a retina. As part of an ongoing project we report in this paper our results on a variety of movement trajectories and how the estimates depend on a range of sizes and the number of receptive fields.
2 Mathematical Model

The vision system is designed following previously developed models [6] and their subsequent improvements [7] that intend to obtain, from the image of an object located on an artificial retina, an estimation of the center of gravity and size of this object, and from the successive images that arrive to the retina during the several temporal samples, an approximation of its direction and speed in order to be able to follow it as it moves. In these and the following sections, the pixels will be considered as the quantification units, as the photoreceptors are in the natural system. The model is outlined as follows.

2.1 Center of Gravity

To locate the center of gravity of a uniformly lit object, a set of square receptive fields of 10x10 pixels is randomly generated over the retina. A receptive field responds when part of the object covers its area completely, providing two outputs that correspond to its horizontal and vertical co-ordinates. Thus, the number of fields that react are counted and their co-ordinates are added in two different summing boxes, one for each co-ordinate. The sum of the values of each co-ordinate is divided by the number of receptive fields, thus obtaining the averaged co-ordinates, that will be an approximation of the position of the center of gravity.

2.2 Size

In order to obtain an approximation of the size of an object, a series of randomly distributed receptive fields is generated all over the retina as in 2.1. The size of the object is then calculated by dividing the number of receptive fields covered by the object, by the total number of receptive fields generated.

2.3 Motion

To detect motion, we use receptive fields configured as illustrated in Fig. 1. Each such field contains eight sub-fields. The latter are formed by linear arrays of pixels oriented vertically, horizontally or diagonally and intersecting on a center pixel. Each array radiating from the center pixel forms a sub-field, giving eight in all. The receptive fields can be of different sizes within a square pixel frame. As before in 2.1 and 2.2, they are distributed randomly over the retina.

The initial state of a receptive field is shown in Fig. 1 (left). Only the center pixel is capable of responding as indicated by the +. The other pixels are inhibited as shown by the -. When a stimulus, such as the edge of an object, passes over the center pixel, the receptive field changes to the state at the right of Fig. 1. As the edge moves over the receptive field it activates the pixels over which it passes. Since time is digitised in intervals $\Delta T$, the number of pixels over which the edge has passed in any one subfield in time $\Delta T$ is a measure of its speed. When activated, each sub-field
generates its own output which has two components: one is its direction (including sense, i.e. up, down, etc) and the other is speed (number of pixels activated in time $\Delta T$). The output from all the subfields with a given direction on the retina go to a summing box for that direction. Then an object moving over the retina will tend to produce the largest summing box output for the direction closest to its motion. This can be seen from the following illustration.

![Fig. 1. Change of the RF’s weights as explained in text](image)

Suppose a square in the orientation of the RF is moving upwards. If one considers how a leading corner can move in different positions relative to the vertical edge of a RF over all the RF’s on the retina, one easily finds that the upward vertical summing box gives the largest output with lesser outputs from summing boxes for the upward diagonals. The presence of responses in more than one direction is somewhat analogous to the “aperture problem” which has been discussed extensively [1], [2]. Although that also happens in the natural systems, as cells react not only to a very specific direction, but also to a whole range of directions around the preferred one, it represents a problem when trying to determine approximately the direction of motion with a certain degree of accuracy.

With complex shapes and movements there can be ambiguities in addition to the “aperture problem”. To minimise these we subtract from each subfield in a RF the outputs of the 2 orthogonal subfields. This is somewhat analogous to lateral inhibition and enhances the value of the correct direction. A further improvement of the estimate of direction can be made by averaging the outputs of the direction summing boxes over $k\Delta T$, where $k$ is generally 2. All these operations are performed on line on a real time basis.

Once the direction of the object has been found, its speed is calculated by dividing the total number of pixels over which the border has moved in that direction by the total number of activated subfields in $\Delta T$.

3 System Description, Setup and Results

3.1 The Visual Environment

The system has been implemented on a Pentium™ II, 500 MHz, 64 MB RAM computer in which an acquisition and digitisation board (IC-ASYNC, Imaging
Technology™) has been installed. This image board processes b/w images. A monochromatic progressive camera CV-M10 with 640x480 pixels resolution is used, though the recorded images are 256x256 pixels and 256 grey levels. Software has been developed under Visual Basic 5.0. The input image is then on a first stage processed by a set of squared receptive fields randomly distributed on the image surface. All artificial vision systems being extremely sensitive to illumination conditions, our image acquisitions take place in a closed, strictly controlled structure. This structure has a working surface of aprox. 1 m², and is 2m high. The camera is on top of the structure, located at 1.5 m from the surface where the object is moving. This object, square shaped, is built using Lego™’s Robotics Invention System®, a construction kit that includes a programmable microprocessor and several sensors (light, touch, rotation) and motors that allow us to program, control and monitor its behaviour on line.

![System general diagram](image)

**Fig. 2.** System general diagram

### 3.2 Results on Size and Speed Estimation

Once the system was calibrated (3 pixels = 1.14 cm), a data base of several simple trajectories at different speeds was built to begin an exhaustive study of the influence of the several possible parameters involved. These trajectories can be seen in Fig. 3.
In the following table (Table 1), the results for size and initial position of the center of gravity (CG) for two different sizes of the object are shown. Size is expressed as a % of the retina covered by the object (number of pixels in brackets) and position expressed in (x,y) co-ordinates, with the origin on the lower left-hand corner. The number of RF’s was 27830, of size 10x10. There were slight variations in the estimates depending on the position and orientation of the object.

Table 1. Results for size and initial position of CG for two different objects

<table>
<thead>
<tr>
<th>Object size</th>
<th>Computed size</th>
<th>CG Position</th>
<th>CG Comp. Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.258 (1480)</td>
<td>2.291 (1501)</td>
<td>(137,88)</td>
<td>(139,88)</td>
</tr>
<tr>
<td>4.378 (2869)</td>
<td>4.230 (2772)</td>
<td>(53,42)</td>
<td>(53,42)</td>
</tr>
</tbody>
</table>

These time independent descriptors are calculated in real time, that is to say, they are computed and plotted as the camera acquires and the computer processes information from the object (20 frames per second). The results are quite satisfactory in size estimation and the location of the CG. The estimated speed depends on the size of receptive fields, as has already been noted above in the explanation of our model. This is shown in the following table for a vertical movement under constant speed, where RFS stands for Receptive Field Size meaning the length of each side of the squared receptive fields. In this case, the number of RF’s was 34992.

Table 2. Estimated speed for a vertical movement when RFS is incremented (Real speed: 6.85 cm/sc)

<table>
<thead>
<tr>
<th>Size (pixels)</th>
<th>RFS 3</th>
<th>RFS 5</th>
<th>RFS 7</th>
<th>RFS 10</th>
<th>RFS 15</th>
<th>RFS 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cms)</td>
<td>1.14</td>
<td>1.90</td>
<td>2.66</td>
<td>3.80</td>
<td>5.70</td>
<td>7.61</td>
</tr>
<tr>
<td>Estim. Speed (cm/sc)</td>
<td>02</td>
<td>03</td>
<td>03</td>
<td>03</td>
<td>03</td>
<td>03</td>
</tr>
<tr>
<td>% error</td>
<td>-0.29</td>
<td>+1.60</td>
<td>+1.60</td>
<td>+1.60</td>
<td>+1.60</td>
<td>+1.60</td>
</tr>
</tbody>
</table>
Table 3. Estimated speed for a vertical movement when RFS is incremented (Real speed: 22.64 cm/sc)

<table>
<thead>
<tr>
<th>Size (pixles)</th>
<th>RFS 3</th>
<th>RFS 5</th>
<th>RFS 7</th>
<th>RFS 10</th>
<th>RFS 15</th>
<th>RFS 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cms)</td>
<td>1.14</td>
<td>1.90</td>
<td>2.66</td>
<td>3.80</td>
<td>5.70</td>
<td>7.61</td>
</tr>
<tr>
<td>Estim. Speed (cm/sc)</td>
<td>7.56</td>
<td>15.68</td>
<td>19.78</td>
<td>22.57</td>
<td>22.59</td>
<td>22.59</td>
</tr>
<tr>
<td>% error</td>
<td>-66.60</td>
<td>-30.74</td>
<td>-12.63</td>
<td>-0.30</td>
<td>-0.3</td>
<td>-0.3</td>
</tr>
</tbody>
</table>

It can be seen that at the lower speed (Table 2), the estimate of speed stabilises with a 5x5 RF size. But at the higher speed it only stabilises with a 15x15 RF size (Table 3). Doing the same experiment when the object is following a diagonal trajectory, yields the next table of results. In this case, it has to be kept in mind that a $2^{1/2}$ factor has to be introduced when converting distance from pixels to centimetres.

Table 4. Estimated speed for a diagonal movement when RFS is incremented (Real speed: 8.13 cm/sc)

<table>
<thead>
<tr>
<th>Size (pixles)</th>
<th>RFS 3</th>
<th>RFS 5</th>
<th>RFS 7</th>
<th>RFS 10</th>
<th>RFS 15</th>
<th>RFS 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cms)</td>
<td>1.14</td>
<td>1.90</td>
<td>2.66</td>
<td>3.80</td>
<td>5.70</td>
<td>7.61</td>
</tr>
<tr>
<td>Estim. Speed (cm/sc)</td>
<td>7.78</td>
<td>8.23</td>
<td>8.23</td>
<td>8.23</td>
<td>8.23</td>
<td>8.23</td>
</tr>
<tr>
<td>% error</td>
<td>-4.30</td>
<td>+1.23</td>
<td>+1.23</td>
<td>+1.23</td>
<td>+1.23</td>
<td>+1.23</td>
</tr>
</tbody>
</table>

Table 5. Estimated speed for a diag. movement when RFS is incremented (Real speed: 19.20 cm/sc)

<table>
<thead>
<tr>
<th>Size (pixles)</th>
<th>RFS 3</th>
<th>RFS 5</th>
<th>RFS 7</th>
<th>RFS 10</th>
<th>RFS 15</th>
<th>RFS 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cms)</td>
<td>1.14</td>
<td>1.90</td>
<td>2.66</td>
<td>3.80</td>
<td>5.70</td>
<td>7.61</td>
</tr>
<tr>
<td>Estim. Speed (cm/sc)</td>
<td>7.17</td>
<td>18.06</td>
<td>19.56</td>
<td>19.56</td>
<td>19.56</td>
<td>19.56</td>
</tr>
<tr>
<td>% error</td>
<td>-62.65</td>
<td>-5.93</td>
<td>+1.87</td>
<td>+1.87</td>
<td>+1.87</td>
<td>+1.87</td>
</tr>
</tbody>
</table>

As before, the estimate of speed stabilises with smaller RF size (5x5) at the lower speed (Table 4), vs. the larger RF size (7x7) at the higher speed (Table 5). The next experiment was meant to try to obtain a trajectory description from the speed and direction channel not using the information yielded by the position channel (which is not time dependent since the position of the CG of the object is recalculated on every frame). Thus, under the explained conditions and using the trajectories mentioned at the beginning of this section, the first results can be seen in Fig. 5a, where the black line corresponds to the trajectory followed by the center of gravity of the object as computed by the spatial channel of the system, and the grey one is the estimation of trajectory drawn by using the speed and direction estimations of the spatio-temporal channel.

As noted earlier in the discussion of our model the position estimates are improved by averaging over $2\Delta T$. More generally we explored a longer averaging time and
looked at different weighting functions $P(t)$ for the past intervals. The position estimate is then given by the following expression (3.1):

$$R(i, t) = S(i, t) + \left( \sum_{j=1}^{TV} P(-j) \cdot S(i, t - j) \right) - S(i - 2, t) - S(i + 2, t), \quad i = 0..7 \quad (1)$$

Where $R(i, t)$ is the output for direction $i$ at instant $t$ after the original calculation $S(i, t)$ is modified by both the lateral interaction scheme and the memory-averaging operation. $TV$ is the size, in units of past time, of the memory, and $P(-j)$ represents the weights by which the outputs at those past instants are affected. Several different weighting functions to determine $P(-j)$ have been used, whose graphs are shown in Fig. 4.

![Fig. 4. Profiles of the memory weighting functions](image)

Some results are shown in Fig. 5(b) and (c), where the picture on the left shows the trajectory estimation using two previous instants memory, and the one on the right 10 memory instants.

![Fig. 5. An estimation of trajectory drawn after the data yield by the speed channel (grey), compared with the output from position channel (black). 5(b) and (c): The effect of excessive memory influence in the trajectory estimation using data from the velocity channel: (b) using 4 previous instants, (c) using 10](image)
3.3 The Effect of Varying the Number of Cells

This round of tests is ended with the study of the effect of the variation of the number of receptive fields, in determining speed and the estimated trajectory (Fig. 6). Thus, fixing a speed of 22.64 cm/sc we get the following results for a number of cells ranging from 46656 (which corresponds to one RF per input pixel) to 100 (0.2% of the previous figure) and for the usual variation of the receptive field size from 3 to 20 pixels/side.

As can be seen, in order to get a reasonable figure for the estimated speed we need to have at least 2000 receptive fields and a size of at least 10x10 pixels distributed on the retina. Better results are yielded by increasing the number of receptive fields. But with more than 11644 receptive fields there is no practical improvement in the error with a 10x10 pixels receptive field. This suggests that with these numbers of RF the system has a high built-in fault tolerance that allows us to compute values very close to the correct speed estimation (at least with an error margin between 0.31% to 1.37%). At the same time, the computational costs of our system can be controlled. The results in the previous graph refer to a vertical trajectory. The system yields similar results for horizontal and oblique trajectories. With different speeds the number of receptive fields and their sizes for the best estimates will be different.

Fig. 6. Percent error in the estimated speed vs. Receptive Field Size as function of the number of Receptive Fields

4 Bases of a Pre-attentive Model. Formal Considerations

A schematic representation of the bio-inspired visual motion analysis system we have explained is shown in Fig. 7.
Fig. 7. Representation of the proposed visual motion analysis system

Information is processed by the different integration layers and flow to higher levels. Obviously, these intermediate layers do not work as simple information boosters, but are the basis through which information is transformed in a higher-semantic content representation. Thus, in the visual motion analysis system we have presented, the local information processed by the first level of cells is transformed in such a way that when reaches higher levels the system manages descriptors such as the position and size of the object (CG-Size Channel) and the trajectory and speed of the object (Velocity Channel).

One critical point to be considered when we try to develop a real-time system capable of processing the information required by different channels is the computational cost. In this sense we try to obtain the best performance working on three different aspects:

- **Hardware Performance**: We always try to use new generation computer systems, both fast and reliable, which are capable to reach high information processing speed.
- **Parallelism**: The use of multiprocessors systems allow us to paralelise the different algorithms, so that the different channels can work in parallel (similar to biological systems).
- **Additional Mechanism**: We also work on the study of mechanisms through which the necessary information processing to perform on the image which is present in the artificial retina can be done in a more efficient way, that is, controlling two interrelated and critical variables: time and computational cost.

Following this research line we have proposed the bases of a pre-attentive model that takes the information processed by the trajectory-speed channel which has already been explained in details as input data.
This idea is inspired by the study performed on a specific type of synapsis that takes place in the nervous system of many living creatures: Presynaptic Inhibition (PI). PI is one of the most interesting inhibitory mechanisms found in the nervous system and consists of the strong suppression of a neuron’s response before the stimulus reaches the synaptic terminals mediated by a second, inhibitory, neuron. This mechanism has been observed in motorneurons and in several other places of nervous systems, mainly in sensory processing.

The modulation function achieved by this type of synapsis has led to think that it could play an important role as a ‘First-order’ attention focusing mechanism.

More precisely, we propose a mechanism in which information regarding the trajectory of the object (trajectory-speed channel) could be used as feedback information to the intermediate layers, where the PI mechanism would select the sets of stimuli that will be derived to higher brain areas, which in turn is equivalent to a ‘higher-level filter’ of information, in the sense of filtering the possible semantic content of the information that is allowed to reach later stages of processing.

Thus, if we suppose the existence of a fovea (area in retina in which the resolution is maximal) which implies an intensive information process and consistently a high computational cost, our idea is to develop a system in which information extracted by the trajectory-speed channel is used as feedback information that via PI would allow the system to relocate the fovea, being the intensive information process applied only to the ‘relevant’ part of the image (Fig. 8).

Fig. 8. Schema of the pre-attentive model
This mechanism could be proposed as a ‘First-order’ attention focusing mechanism. Note that it has nothing to do with ‘willness’ or purpose of behaviour, it is closer to an automatic response depending on outside stimuli.

5 Conclusions

The system, as it is shown, provides reliable and fast results in the spatial channel: the size and center of gravity values that we wanted to measure are within less than 2% error margin on average. A complete description of speed and direction of movement parameters by the velocity channel, however, being a more complicated model, takes longer time but yields good results provided we control several parameters affecting the output, namely number and size of receptive field, memory and lateral interaction. The main conclusions we obtain from the analysis of the system’s behaviour after varying several parameters are explained in the following paragraphs.

The basic operations needed with our model make it possible to carry them out in parallel, in a two-channel fashion.

The random distribution of receptive fields obviates the necessity of having a deterministically organised system and it seems that the natural system which has inspired the model does not use a perfectly established network either.

Seen as a whole, and comparing the location of the CG of the object calculated separately by both channels (e.g. the black and grey lines in Fig. 5), the results give us some perspective on the usefulness of channel processing in artificial visual systems which could also be of interest in trying to think of a rationale for the same kind of computations in natural perceptual systems. Though an estimate of the position of the object can be calculated from the velocity channel, it is not as good nor as fast in its delivery as it is when presented by the purely spatial CG-size channel, and is greatly dependent on the number of receptive fields that are used. The system could work properly for all descriptors (CG, size, position, velocity), in certain cases, with only the output from the velocity channel, but if precision is needed, then a second channel must be included. In either case, speed and direction of motion can be finely estimated.

From the data obtained in the tests varying the total number of receptive fields one conclusion is the following: a reliable system for computing speed and trajectory of a moving object can be built whose computational complexity and cost (in terms of number of processors, distribution and size of receptive fields) can be controlled to fall within certain desirable ranges. The built-in parallelism of the system allows us to play with those parameters avoiding the increase of the percent error in estimated speed. Thus loss of processors (loss of receptive fields) need not dramatically affect the performance of one subsystem (such as the movement detection and analysis system) provided there is a minimum overlap and the RF sizes are big enough to cope with a range of speeds. A possible extension on which we are currently working, is a system containing several subchannels (for, e.g., effectively computing different speeds or different other descriptors) which might be more reliable and less costly (both in complexity and computational operations) than a single do-it-all channel, even when the number of processors of this last channel could be less than the total sum of them in the former scheme.
In the final part of the paper we have proposed the bases of a pre-attentive model that could play an important role in the implemented system from two points of view:

- One the one hand, as already said, as an attention mechanism. We should not forget that the sense of vision is, in most of species, the one that processes the biggest amount of information coming from the outside, being the most important in guiding its behaviour. Thus, the ability to discriminate certain parameters and locate what part of all that information is significant is crucial for both natural and artificial complex systems.

- On the other as a mechanism that feeds the system with information that allows it to reduce the computational cost derived by the process carried out on the image falling on the artificial retina. These could be done by the definition of ‘areas of interest’, allowing the system to perform a ‘selective’ information process.

References

Towards a Model of Volume Transmission in Biological and Artificial Neural Networks: A CAST Approach

Carmen Paz Suarez Araujo\(^1\), Pablo Fernandez Lopez, and Patricio Garcia Baez\(^2\)

\(^1\) Department of Computer Sciences and Systems, University of Las Palmas de Gran Canaria, 35017 Las Palmas de Gran Canaria, Canary Islands, Spain
cpsuarez@dis.ulpgc.es

\(^2\) Department of Statistics, Operating Research and Computation, University of La Laguna, 38071 La Laguna, Canary Islands, Spain
patricio@csi.ull.es

Abstract. At present, a new type of process for signalling between cells seems to be emerging, the diffusion or volume transmission. The volume transmission is performed by means of a gas diffusion process, which is obtained with a diffusive type of signal (NO). We present in this paper a CAST approach, in order to develop a NO diffusion model, away from a biologically plausible morphology, that provides a formal framework for the establishing of neural signalling capacity of NO in biological and artificial neural environments.

It is also presented a study which shows implications of volume transmission in the emergence of complex structures and self-organisation processes in both biological and artificial neural networks.

Finally, we present the diffusion version of the Associative Network (AN) \textcite[6]{6}, the Diffusion Associative Network (DAN), where a more general framework of neural learning, which is based in synaptic and volume transmission, is considered.

1 Introduction

Cellular communication is responsible for neural information transmission and processing, as well as learning and cognitive and behavioural processes in the brain. This rapid and precise communication is made possible by two signalling mechanisms: Axonal conduction and synaptic transmission. Nowadays, the latter is the basic mechanism for excellence, for signalling between cells and for many of the brain processes.

In the synaptic transmission process it can distinguish between basic elements and constraints, which are mutually related. The basic elements are the neurotransmitters, the receptors and the contacts between nerve cells. The constraints...
will be the unidirectional transmission, from pre-synaptic to post-synaptic neuron, the release of neurotransmitters and receptors fit for these neurotransmitters in the pre-synaptic and post-synaptic sides respectively.

At present, a new type of process for signalling between cells seems to be emerging, that is to say, *diffusion or volume transmission*. Volume transmission is performed by means of a gas diffusion process, which is obtained with a diffusive type of signal. The main differences between synaptic and volume transmission is that the latter is membrane permeable and multidirectional because it disregards the spatial constraints on neurotransmitter activity [2].

The presence of a molecule in the brain that acts as a diffusive signal that affects the diverse neighbouring cellular processes opens up new perspectives regarding the study of brain function. Nitric oxide (NO) was soon found to be a more likely candidate, because of its extreme isotropical diffusibility in both aqueous and lipid environments, which allows a rapid three-dimensional spread of the signal irrespective of the presence of membranes [11].

In the absence of relevant experimental data for understanding how NO functions as a neuronal signalling molecule, we have developed a NO diffusion computational model. Several NO diffusion models have been published [4], [5] and [8]. They consider the spread of NO from sources whose morphology are not biologically validated and even some mathematical anomalies and biologically implausible results can be introduced for these approaches.

We present in this paper a CAST approach in order to develop an NO diffusion model, away from a biologically plausible morphology, that provides a formal framework to establish the neural signalling capacity of NO in biological and artificial neural environments. We will address this problem using problem-solving by models [1]. In addition to this, our model also has repercussions on establishing hypothesis regarding the existence of neural mechanisms and the design and interpretation of biological experiments on NO behaviour and effects on brain structure and function.

Furthermore, our study is different as and for first time, we show the emergence of complex structures and self-organisation processes in both biological and artificial neural networks.

Finally, we present the diffusion version of the Associative Network (AN) [6], the *Diffusion Associative Network (DAN)*, where a more general framework of neural learning, which is based in synaptic and volume transmission, is considered.

## 2 NO Diffusion Computational Model: A CAST Approach

The main objective of this study is to move towards a better understanding of the underlying neural mechanisms of cellular communication in the brain, their relationship with learning and structure emergence in biological and artificial neural networks.
Our research and this section are focused on a new kind of neural signalling transmission, volume transmission, whose underlying mechanism is a diffusive signalling molecule (NO) working as a retrograde cellular messenger. We will be specifically taking the first step towards developing the Theoretical Framework (TF) belonging to the NO diffusion messenger global study framework (GSFNO) [2].

We present a CAST approach in order to develop the NO diffusion computational model, which provides new neural signalling transmission models. We will address this problem using problem-solving by models [1]. Taking into account the structure of a problem-solving process, we can say that our research belongs to the modelling process stage.

Using the different types of models proposed by Pichler [1], our initial $M_0$ model is of the generator type. By following the model-transformation application method, we obtain a dynamics $M_f$ type model which will be used to model the NO diffusion.

The diffusion process is closely linked to the transport phenomena. This phenomena takes place when, due to the gradient of a physical magnitude, another is displaced and invades and affects its setting in a structural and/or functional way. Its behaviour does not depend on any type of molecular theory and can be studied from a phenomenological point of view. Thus, we propose to use physical phenomenological laws of diffusion, Fick’s laws regarding isotropic mediums, Eq. (1) [12], and theoretical and experimental biological studies into the NO generation pattern [2], [4], [9] and [10].

$$F = -D \frac{\partial C}{\partial x}.$$ 

where $F$ is the rate of transfer per unit area of section, $C$ the concentration of diffusing substance, $x$ the space co-ordinate measured normal to the section, and $D$ is the diffusion coefficient.

The fundamental differential equation of diffusion in an isotropic medium is derived from (1)

$$\frac{\partial C}{\partial t} = D \left( \frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2} \right).$$

where $x, y, z$ are the space coordinates.

The dynamics of the diffusing messenger have not yet been experimentally defined. Taking into account the fact that NO release does not need presynaptic specialisation, the whole surface of an NO neuron is a potential source. This implies that these dynamice respond to a volumetric morphology. We propose a cylindrical morphology, which considers the neuron as the basic NO module emitter. This obeys the quantal and non-local nature of the NO emission, the possibility of NO emission in multiple areas of the neuron, which leads us to a parallel emitting source structure situated throughout it, as well as to the NO self-regulatory capacity.
Towards a Model of Volume Transmission in Neural Networks

We can therefore consider a long circular cylinder throughout which diffusion is radial. Thus, the $M_0$ is given by the Eq. (2) in terms of the cylindrical coordinates, modified by the kinetics of NO decay. They are expressed by means of an inactivation function, used to model the NO loss through various chemical reactions. This has been taken to be exponential decay since there is not real data to suggest any other function. Thus: $1/2life = ln(2/\gamma) \equiv 8$.

$$\frac{\partial C}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left( rD \frac{\partial C}{\partial r} \right) - \gamma C. \quad (3)$$

Using the model-transformation application method, we obtain a dynamics type model $M_f$, in terms of Bessel functions $[7]$, suitably chosen so that the initial and boundary conditions are satisfied.

$$C(r,t) = \sum_{n=1}^{\infty} A_n J_0(\xi_n r) e^{-\lambda_n D t}. \quad (4)$$

where

$$A_n = \frac{2}{r_1^2 J_1^2(r_1 \xi_n)} \int_0^{r_1} f(r) J_0(\xi_n r) r dr. \quad (5)$$

If the initial concentration distribution $f(r)$ is maintained at zero concentration and the surface $r = r_1$, then $J_0(x)$ and $J_1(x)$ are Bessel functions of the first kind of order zero and one, $\lambda_n - (\gamma / D) = \xi_n^2$. Where $\lambda_n$ is the separation variable, and the diffusion constant is $D = 3 \times 10^{-5} cm^2/s. [11]$.

Another phenomenological approach, which is not only mathematically valid, but also biologically coherent, used in our model has been to consider the NO synthesis process as a pulse function. This coherence obeys to experimental results which infer that NO once synthetised, it freely diffuses $[13]$.

2.1 Implications in Neural Networks

The proposed volumetric transmission model gives results that are in line with experimental results, as well as with hypothesis about NO diffusion behaviour, and its possible implications neural architecture and learning processing both in biological and artificial neural networks.

Fast Diffuse Neural Propagation (FDNP) The Fig. 1 show NO concentration profiles at various times, $t = 0.1s$ and $t=0.2s$, according to the distance to the NO neuron and temporal concentration profiles at various distances, $r=100 \mu m$, $r=150 \mu m$. The local and highly dynamic nature of NO distribution can be seen there, which is limited by the diffusion radius that is defined by the half-life of the diffused NO. Therefore, we can observe the existence of a FDNP. In Fig. 2, it can likewise be seen that the presence of diffused NO is slightly stronger in $t=0.2s$ than in $t=0.1s$ in the areas located over 70 micrometres.
Fig. 1. Profiles of the concentration of NO at various times, $t = 0.1 \, s., \, t = 0.2 \, s.$

Fig. 2. Temporal concentration profiles at various distances, $r = 100 \, \mu m., \, r = 150 \, \mu m.$
Diffuse Neighbourhood. A further result that confirms the reliability of the proposed model is its behaviour in relation to the formation of neural neighbourhoods and the emergence of complex structures. One of the most important NO implications for ANNs is the emergence of Diffuse Neighbourhood (DNB) produced by a diffusion neural association. In this DNB, there will be transmission of some information from the NO neuron to all the neighbouring cells without any synaptic connections being established among them. We can define two kinds of DNB that are directly influenced by the NO neuron: The first-rate DNB, defined on the same neural level, (see fig. 3), and the second-rate DNB on the adjacent anterior and posterior neural levels [2].

The emergence of DNB and three-dimensional spread of the retrograde messenger effect can produce volume plasticity. The consequence of this volume plasticity is that correlation will be reinforced between elements which are coactive within a given time window and are within a critical volume, without being necessarily physically connected. This is a step towards obtaining simple neural structures and towards speeding up the learning and network operation. The effect demonstrated by this behaviour may be able to produce self-organisation in the neural network and to build neural columns, in the same way as in the cortex, which has a columnar organisation [2].

In this paper, we describe the existence of first-rate DNB generated by NO diffusion, according to the proposed model. Fig. 4 shows the DNB corresponding to Fig. 3b, for the times of the NO concentration profiles in Fig. 11 as well as for the parameter experimental values, half-life time, $\gamma$, and maximum diffusion radius, $r_1$ de 200 $\mu$m. and 5 s respectively. These are the parameters that define, for excellence, the form and scope of the diffuse neighbourhood (DNB).

The NO synthesis and, therefore, its diffusion through the neural tissue, are not a static or isolated process and are in fact a dynamic and multiple one. On
the other hand, taking into account the dense framework of the biological neural network and, therefore, the one that the artificial neural network wishes to maintain, the number of neurons that are emitting NO in narrow time bands can be very high, and therefore multiple numbers of DNB are produced. This involves complex structure formation, due to the formation of complex diffuse neighbourhoods, emerging both from the high time and space neighbourhood overlapping. This phenomena is demonstrated by our model, considering the lineal superposition between the various emerging DNB. Figs. (5)a and (5)b show the complex structure formation when 2 and 3 DNBs emerge as a result of various diffusion process with a 0.1s time interval between them.

In these neural settings, where a great number of diffuse neighbourhoods are involved, it is very useful to establish the NO neuron direction and therefore the neural assemblies which influence a neuron. That is to say, it allows us to determine which are the diffuse neighbourhoods that influence the behaviour of a specific neuron. This is highly important in the retrograde character of the neuro-messenger. Our model has repercussions for establishing hypothesis on the existence of underlying neural mechanisms to detect NO neuron direction. It has this potential and provides a mathematical formalism that allows us to study the direction of the NO neuron. This formalism is depicted in the so-called "NO neuron detector functions" expressed in (6).

\[
\begin{align*}
    f_1(x, y) &= |C(x, y) - C(x + dx, y)| \\
    f_2(x, y) &= |C(x, y) - C(x + dx, y + dy)| \\
    f_3(x, y) &= |C(x, y) - C(x + dx, y + dy)| \\
    f_4(x, y) &= |C(x, y) - C(x - dx, y + dy)|
\end{align*}
\]  

(6)

The meaning of these functions is determined by the meaning of an increase being produced in them. Taking into account the equations given in (6) and Fig. (1), an increase may be inferred in some of the previous functions, for example, it establishes that the concentration of diffused substance, in (x, y) y (x+dx, y) is changing. This change in the concentration will be caused by diffusion phenomena located in the direction of \( f_1 \), which implies that the neuron(s) responsible for the generated volumetric transmission, are localised in the direction indicated by this functions. We can apply similar arguments to the rest of the directions determined by functions \( f_2, f_3 \), y \( f_4 \). Therefore and by means of the behaviour of the detector functions throughout time, a localisation of the diffusion phenomena can be performed.

**Learning Processing.** The most appealing characteristic of ANNs is their learning capacity. In this way the ANNs behaviour is emergent of structural changes directed by learning laws which provide plasticity to these neural structures. A natural substratum for these characteristics seems to be present in the diffusive behaviour of NO and in their implications in neural structure and processing.

The learning mechanism both in biological and artificial neural networks has been explained by means of synaptic plasticity using the concept of neurotrans-
(a) DNB at $t = 0.1$ s. 
(b) DNB at $t = 0.2$ s.

**Fig. 4.** Diffusion Neighbourhoods for a NO neuron at position (0,0).

(a) 
(b)

**Fig. 5.** Emergence of complex structure for overlapping of several DNBs corresponding to several NO neurons involved in diffusion processes with a delay of 0.1 s. between them a) overlapping of DNBs produced by two NO neurons. b) overlapping of DNBs produced by three NO neurons.

**Fig. 6.** Detectors of direction of NO neurons. NO receptor neuron in the central position surrounded by 8 NO neurons.
mitters and weights respectively. In both cases it is necessary to establish exact connections between the neurons which are participating in the process. Some neural activities resulting in plasticity, learning or memory can be more easily explained if a retrograde cellular messenger, such as NO, can be considered. Interestingly, NO has been implicated in long-term potentiation (LTP) in the hippocampus [2] and [10]. Nitric oxide diffuses over large distances in cortical tissue. Because of this it may underlie a form of non-local, non-synapse-specific learning. This non-locality leads to issues that are commonly ignored in Hebbian learning rules [2].

The first explicit statement of the physiological learning rule for synaptic modification was given by Donald Hebb in 1949 [3] who tried to explain learning and memory processes. Hebb’s assumption, known as the Hebb synapse, at the moment called Hebb learning law, can be stated as follows: When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased [3].

There is not direct evidence of the cause of the postulated change of the efficiency between two connected cells, A and B. We propose as a probable biological support to the Hebb learning law the presence of an NO retrograde cellular messenger. This neuromessenger is produced during the depolarization process of postsynaptic neuron. It freely diffuses through membranes affecting all neighbouring cells, including the presynaptic one and the cell that released it. Because of this we must take into account a new conception of the meaning of the Hebb law; it is not a correlational postulate any more.

This new conception of Hebb’s rule implies a reformulation of its mathematical expression. The change in NO concentration has to be considered as a new variable with an important role in the learning process, and the incursion relation as the formal framework for expressing this learning process.

\[
w_{AB}^{t+1} = w_{AB}^t + \delta \left( g_1(x_A(t), \Delta n_A)g_2(S_B(t), \Delta n_B, w_{AB}^{t+1}) \right) . \tag{7}
\]

where \( t \) is discrete time, \( w_{AB} \) is the interconnection weights between A and B neurons, \( x_A \) is the information from the presynaptic neuron, \( s_B \) is the activation state of a postsynaptic neuron, \( \delta \) is the learning rate which is related to the gas diffusion process, \( \Delta n_A \) and \( \Delta n_B \) are the gradients of NO concentration around neurons A and B respectively. These gradients will be determined by means of transport phenomena using phenomenological laws, as diffusion equations. Finally, \( g_1 \) and \( g_2 \) can be linear or non-linear functions which indicate the specific effect of pre and postsynaptic information in the learning process.

We present the effect produced in a learning process due to volume transmission. We propose the simplest modification of Hebb’s learning law in a neural architecture where the modification of weights is based on the correlation.
3 Diffusion Associative Network (DAN)

In this section, we describe the behaviour of an artificial neural network, both in the learning mode and functioning mode, when the existence of the synaptic and volumetric transmission is considered. We will develop the diffusion version of a classic architecture, the Associative Network (AN) using the model proposed for volumetric transmission. The learning law of the proposed new neural architecture, known as Diffusion Associative Network (DAN), will be of the form given in the equation (9) and it will specifically reflect its most simple abstraction. The Associative Network is based on correlation matrix model. It has an input layer and an associators layer. The input layer, consists of two parts: the key field \{a_1,a_2,a_3,...,a_n\} used for the encoding of data and the data field \{u_1,u_2,u_3,...,u_m\}. All input elements are called receptors. The associators consists of a set of elements labelled by a pair (i,j) corresponding to the ith element of the key field and the jth of the data field to which the particular associator is connected.

The learning process to memorise data consists of increasing the value of every associator element \(M_{ji}\) by the amount directly proportional to \(a_i u_j\).

For a set of patterns \((a_1^1,u_1^1),(a_2^2,u_2^2),(a_3^3,u_3^3),..., (a_p^p,u_p^p)\),

\[ M_{ji} = \rho \sum_p a_i^p u_j^p. \]  

where \(\rho\) is the normalising constant.

The recall of a particular \(u_j^r\) associated with a key vector \(a_r^r\) is made by transformation

\[ \hat{u}_j^r = \sum_i M_{ji} a_i^r. \]  

The structure of the Diffusion Associative Network (DAN) is shown in the Fig. 7. The associator elements are organised in neural assemblies. The number of neural assemblies in a DAN is equal to the dimension of the key field, n. Each assembly is made up by a number of associator elements equal to the size of the data field of the DNA. Each associator belonging to an ith neural assembly receives two connections from the receptor field. One comes from the ith receptor of the key field, which is the one that defines the neural assembly, and the other connection is received from the receptor belonging to the data field, of the same order as the associator. An associator element of the DNA is defined by \(M_{ij}\), where the subscript \(i\) indicates the ith neural assembly, and the subscript \(j\) the jth associator element of the assembly. Thus, a jth associator element of the ith assembly receives the input of the \(u_j\) and \(a_i\) elements. Fig. 7.

This artificial neural network involves the influence of neural structures in each of its neurons, not only due to the synaptic transmission by means of the specified synaptic connections, but also to the volumetric transmission by means of the diffuse neighbourhoods (DNB). In this way, an assembly will exercise an influence on the other assemblies that are in the DBN. Therefore, in this monodimensional...
structure of neural assemblies, we have selected influences such as the \(i\)th assembly, which can be influenced by the neural assemblies \((i-1)\)th, \(i\)th and \((i+1)\)th, during the diffusion period. This involves the existence of neural information transmission between assemblies supported by the volumetric transmission and therefore volumetric plasticity and a volume learning\(^2\).

A generalisation of this model would involve considering that DNB intra assemblies and not only inter-assemblies exist. Our model includes the inter-assembly volumetric transmission. This is expressed in equations (10) and (11), where the DNA learning process is formalised, which allows the data submitted to it to be learnt, which is a set of \(p\) data in this case. This equation can be seen as a special particularization, discret in time, of the equation (7). Where \(\delta = \rho\) are represented by the parameter \(c\) obtained from our model in the discrete of the volumetric transmission, and \(g_1\) and \(g_2\) are chosen in such a way that represents the correlation between the input and output.

\[
M_{ji} = \rho \sum_p c_i^p a_i^p u_j^p. \quad (10)
\]

In this case the correlation is weighed by \(c_i^p\) which has a behaviour based in the diffusion\(^5\)

\[
c_i^p = (1 - 2D)f(a_i^p) + D(c_i^{p-1} - c_i^{p+1}). \quad (11)
\]

and where

\[
f(a_i^p) = \begin{cases} 
\eta & \text{if } a_i^p = \max(a_i^p) \\
q_i^{p-1} & \text{otherwise}
\end{cases} \quad (12)
\]

where \(\eta\) is the strength of the NO neurons.

### 3.1 Results

In this section, we will show that consideration of the existence of fast diffuse neural transmission and therefore volumetric transmission has implications for the learning process. Not only do the results submitted corroborate the reliability of the proposed model, but also that the synaptic and volumetric transmission are the signal transmission mechanisms responsible for learning. On the other hand, it backs the modification of Hebb’s law set out in equation (9), as well as the new formal framework for learning in ANN proposed by Suarez Araujo\(^2\) and \(\text{[12]}\).

We submitted the set of patterns shown in the Fig.\(\text{[8]}\), to both the AN and the DAN networks, so that they were memorised by them. We performed a data recovery process and the results obtained show the great capacity of the DNA compared to the AN for learning and therefore for recovering the data with greater precision. Furthermore, the reduction of the crosstalk phenomena in the reaction of the Diffusion Associative Network is shown. The Fig.\(\text{[9]}\) shows the
results of the two associative networks, AN and DAN, for a data recovery process for the patterns shown in Figs. (8)(a) and (8)(d).

4 Conclusions

In this paper we present a CAST approach in order to develop biologically plausible model of NO diffusion, moving towards a volume transmission model in biological and artificial scenarios. We have used problem-solving by models and have established a dynamic type model. This paper, using this mathematic-computational model, has explored the neurobiological and artificial implications of NO as a retrograde cellular messenger, proposed in the first paper belonging to NO and ANNs Series \[2\]. The dynamics of NO diffusion behaviour has been considered and we have shown the existence of a Fast Diffusion Neural Propagation (FDNP). We have also looked at the emergence of complex neural structures and self-organisation processes by the emerging the Diffuse Neighbourhood (DNB) and multiple temporal/spatial DNB overlapping. This paper has also studied the implications of NO effects on learning processing in general, and on the learning laws in particular. Specific consideration has been given regarding the reformulation of the mathematical expression of Hebb’s learning law and another one concerning a new formal framework for learning in ANNs. This framework will consist of two kinds of learning:
one based on the classical neurotransmission and a further one based on volume transmission.

All these considerations have been taken in an Associative Network (AN). The result has been a new neural architecture, the Diffusion Associative Network (DAN). Thus, we have demonstrated how adding volume transmission mechanisms to an artificial neural network leads to better solutions for real problems. The research in this paper also has repercussions for establishing hypothesis on the existence of underlying neural mechanisms, such as the capacity to detect NO-neuron direction. It also contribute to design and re-interpretation of new biological experiments on behaviour and effects of NO on the brain structure and function.

Finally, it can be said that the proposed model and the new artificial neural network, DAN, confirm the important fact that biological structures can supply new ideas in order to obtain more realistic and powerful artificial intelligent systems. In the same way, it can also be said that the computational models can supply new biological hypothesis which address the design and re-interpretation of new biological experiments.

References

Towards a Model of Volume Transmission in Neural Networks

Fig. 9. Results of the two associative networks, AN and DAN for a data recovery process for the patterns shown in: (a) Fig. 8(a). (b) Fig. 8(d).
An Online Information Retrieval Systems by Means of Artificial Neural Networks

Marta E. Zorrilla, José Luis Crespo, Eduardo Mora

Department of Applied Mathematics and Computer Sciences, University of Cantabria.
Avda. de los Castros s/n 39005 Santander. SPAIN
{zorrillm, crespoj, morae}@unican.es

Abstract. The aim of this paper is to present a new alternative to the existing Information Retrieval System (IRS) techniques, which are briefly summarized and classified. An IRS prototype has been developed with a technique based on Artificial Neural Networks which are different from those normally used for this type of applications, that is, the self-organising networks (SOM). Two types of network (radial response and multilayer perceptron) are analyzed and tested. It is concluded that, in the case of a limited number of documents and terms, the most suitable solution seems to be the Multilayer Perceptron network. The results obtained with this prototype have been positive, making the possibility of applying this technique in real size cases a cause for a certain degree of optimism.

1 Introduction

At present, thanks to the technological advances of telecommunications and computer technology, information is becoming more and more accessible to the user. Internet for example, is one of the greatest sources of information ever known. With sources of information on such diverse topics, introduced by users with such different search criteria, new requirements arise in the areas of storage, searching and visualisation of information.

Such needs, however, have not arisen now – they have been felt since the sixties, Gerard Salton [10],[11] and his disciples took their first steps in this area with a view to improving the management of library catalogues.

Traditionally, this information is stored in Relational Data Base Systems, in which a document is represented by means of a series of structured fields, such as author, title, ISBN... and the searches are carried out through Boolean techniques. At present, technology enables library catalogues to be amplified and to incorporate summaries and even complete electronic versions of articles, books, etc in their own data bases. However, relational data base systems cannot support search techniques which enable texts to be found by the words used in them – this is called full-text search.

These characteristics are supported by the so-called Information Retrieval Systems (IRS), the case under study in this paper. In the following section, the various
Information Retrieval Techniques will be briefly outlined; next, the neural network architectures for classification applications are presented; and finally, the characteristics of the proposed IRS, some results obtained and future lines of research will be specified.

2 Information Retrieval Techniques

Before entering into further detail, it should be borne in mind that, whatever technique is used, a preliminary stage is practically always required, consisting in obtaining what could be called the "pure text", independently of the original format of the document. This process is usually called "Filtering".

The techniques most widely used in Information Retrieval Systems are now classified and described. Figure 1 shows a basic classification.

![Fig. 1. Classification of Information Retrieval Systems](image)

The most important aspect for the classification of this type of techniques is related to the set of words with which they can work. There are then two main blocks, depending on whether they use any word (free dictionary) or only a certain set of words (pre-established dictionary).

In the case of the free dictionary, there are also two possibilities: to act on words from the language being used (by words), which implies that they maintain, to a certain extent, their meaning; or to act on blocks of characters of a fixed length n, (by n-grams), with the result that their meaning in the text is, in some way, lost. In the latter case, the number of indexes may be quite small and, moreover, this system is independent of the language.

In both cases, the indexing technique most widely implanted commercially is that of "Inverse indexes". With respect to the "by words" system, this technique consists in considering the texts to be divided into documents, paragraphs and sentences. As texts are entered, a DICTIONARY of indexed (usable in searches) words is generated dynamically. New words are incorporated in it, updating the number of documents in which each word is found and the total number of appearances. At the same time, a LIST OF WORDS is formed, in which, for each appearance of a word, its number of appearance in the order of the sentence, of the sentence in the paragraph, of the paragraph in the document and the document code is stored. The process is similar in the case of n-grams, the function of the words being substituted by the n-grams.
The other possible form of grouping consists in limiting the set of words to be used (pre-established dictionary) to a specific set of a relatively small volume. There are basically two alternatives using this approach: Document Clustering and Latent Semantic Indexing.

In both methods, each document is represented by a vector in which each component assumes a numerical value indicating the relevance of a specific word in it. Hence, the dimension of these vectors will be equal to the number of words in the dictionary of terms (key words), which must be defined before the process begins.

The document search is carried out using a query text and consists in encoding it as a vector of all those used to represent documents, making a subsequent comparison between them.

In the case of Document Clustering, the next step consists in obtaining the types of documents closest to this vector and, of these, the documents most alike the query text. The agility of this method is based on comparing the search vector with types of documents rather than doing this with all the documents individually. This system requires a previous classification of documents. There are basically two alternatives for making this classification:

1. Clustering through statistics [4]: This is based on the use of statistics for evaluating the degree of similarity between vectors; the most widely used are Euclidian Distance and the cosine similarity function. The calculation of the similarity between all of the vector pairs leads to the similarity matrix. The required classification is obtained from this. Traditionally the most efficient algorithms in cluster generation, hierarchic agglomerative methods, are the least efficient from a computational point of view, since they need to calculate the matrix of similarity between all the documents in the collection. The main advantage of these methods is that they are clearly defined and there is ample literature available about them both from a theoretical and the practical viewpoint.

2. Clustering through self-organising neural networks: One alternative to clustering by means of statistics is the use of artificial competitive learning neural networks, as outlined below.

Latent Semantic Indexing [1] is a method which tries to solve the problem of lexical pairing between the terms of a search and the available documents, using conceptual indexes rather than individual words. The system is based on the reduction of the dimensions of the problem, by means of changes of base obtained using a truncated Singular Value Decomposition of the document-word frequency matrix.

3 **Artificial Neural Networks for Classification**

There are a large number of reference works which offer the results obtained using artificial self-organising neural networks applied to IR systems. In [12], Scholtes assesses the performance of the self-organising network method according to Kohonen’s algorithm (SOM) applied to a small collection of documents. In [15], the use of the Kohonen method is compared with Fritzke’s Growing Cell Structures method and in [6] Teuvo Kohonen, describes the WEBSOM system, developed by
him and his colleagues, based on the use of his method and applied to large collections of documents (over one million).

Neural networks are computational structures which process information in a distributed and parallel way, using a number of units called processors or neurons. These communicate with each other by sending signals through the connections joining them. Depending on their topology, learning mechanism, the type of association between the input and output information and the way in which this information is represented, the neural network will have one application or the other.

There are several proposed architectures oriented to classification tasks such as multilayer perceptrons, competitive networks (self-organising, for instance) and radial basis function networks.

In the competitive networks, each processor in the output layer corresponds to a class (in this case, a document). For each input point, there is only one processor in the output layer that has a non-null response, which indicates the class to which it belongs.

The radial response networks, unlike the competitive networks, offer a continuous output. All the processors may have a response, some higher than others. In order to find out the classification of each point, the categories are assigned according to the responses of the corresponding processors. For each input, the most probable category is that of the processor with the highest response.

These networks are quite similar to the perceptrons; the main difference lies in the activation function (radial basis function) and in the operations made at the connections.

A radial basis network with local activation function may require more neurons than a feedforward multilayer perceptron network with a tagsig or logsig activation function. This is because sigmoid neurons can have a non-zero output in a more extensive region of the input space, while radial neurals only respond to a small region.

4 Proposed Example

Most indexing techniques related to neural networks are based on the use of a pre-established dictionary (between 100 and 300 terms) and on the representation of documents as a vector of terms whose components respond, to a certain degree, to the importance of this term in the document and with respect to the collection to be indexed, obtaining a classification of these.

The indexing technique proposed here consists of an artificial neural network with suitable classification characteristics. So the aim is not to create clusters of documents but rather to identify each output neuron with a document from the collection. Along these lines, some prototypes have been developed using general purpose tools such as MATLAB, NODELIB and PDP++ [8],[9],[13] and the results are discussed and analysed in the context of the usual techniques.

In short, the model proposed consists in a neural network which has as its input layer the word from the dictionary in binary format, and at its output has as many
processors as the collection has documents (each of these processors represents one document).

From among the different types of neural networks proposed for classification, our tests focus on the use of neural networks with a radial basis functions (RBFs) and on multilayer perceptrons (MLPs).

4.1 Radial Basis Networks

Radial response networks are normally networks with one single layer, although it is possible to build arbitrary networks. The radial function is of the Gaussian type. These networks share a certain likeness with perceptrons, the main difference being in the activation function (radial base function) and in the operations performed at the connections (distance instead of scalar product).

4.2 Multilayer Perceptrons

Basically, in this type of network, processors are grouped together in layers and these are generally ordered in such a way that the processors of each layer take the signal from the previous one (forward feeding). Each processor calculates a function from the scalar product of its weights and its inputs. This function is normally the logistic or the hyperbolic tangent function.

The training in these network models is supervised. The adjustment is an important point in the implementation of the neural networks. Originally, a gradient descent in the error function was proposed as the adjustment technique.

The learning methods must be computationally efficient, and must converge, that is, approach a minimum reliably, and this must be the overall minimum, avoiding any local minimum which the objective function may have. Algorithms based on derivatives are generally used as they are the fastest. These include the gradient descent (based on the fact that the variation in each step taken to reduce the error is more effective if it is in the direction of the gradient); the conjugate gradient descent, which attempts to solve the limited efficiency of the gradient descent method by advancing in orthogonal directions; and, the Newton method, based on an approximation up to the second derivative of the error function [14]. Given that the calculation of second derivatives is more difficult to obtain, there are variants of this algorithm in which an approximation of the Hessian matrix is performed. These are: the Gauss-Newton algorithm, the Levenberg-Marquardt algorithm and the Quasi-Newton algorithm, by means of the approximation of the inverse matrix to the Hessian matrix. In these methods, a one-dimensional minimization is required at each step all along the direction to be taken. Several algorithms can be applied for this minimization, such as those of Newton, Brent or Charalambous [2],[14].
4.3 Data Used

The collection of documents used for this work corresponds to the articles of Spanish Civil Law and, in particular, to the articles under Section Headings I, II and III. For the sake of agility, and to solve the problem of the search and storage of documents, each article was considered a document in ASCII format. In total, 140 documents were considered.

The dictionary of the system is made up of a subset of words, taken from the COES\textsuperscript{1} dictionary, verifying that between 60-70\% were words found in the collection of documents. Given that there is no preference for the coding, each item of vocabulary is encoded as the binary number corresponding to its alphabetical position in the dictionary.

4.4 Discussion of Results

Firstly, the results obtained in the tests performed with RBF type networks will be outlined and next the results achieved with MLP type networks will be presented.

Results with Radial Basis Function Networks

The solution offered by hidden radial layer and linear output layer networks \cite{5} is ineffective, since it creates as many neurons in the hidden layer as there are different words in the dictionary. Therefore the number of parameters required with respect to the inverse index technique is higher.

In the case of a multilayer perceptron with a hidden layer and radial response, it was observed that the geometry of the error function (the entropy and square functions were used) tends too readily towards degenerate minimums with the data handled.

Results with Multilayer Perceptron

The results presented here correspond to an example in which a dictionary of 14 words and a collection of 10 documents were used. The tests employed an architecture of 10 input neurons, 5 neurons in the hidden layer and 10 neurons in the output layer; the square and entropy error functions and various learning methods were used such as stochastic gradient with sample permutation, scaled conjugate gradient, Quasi-Newton with one-dimensional linear minimization using the Brent algorithm and with fixed rate minimization.

The most significant outcomes of the tests performed are as follows:

- A network with 5 processors in the hidden layer is capable of learning the sample without error.
- The method used in the optimization may be of crucial importance.

\textsuperscript{1} COES. Spanish dictionary over 53,000 terms developed by Santiago Rodríguez and Jesús Carretero who work in the Architecture and Computer Systems Department of the Polytechnique University of Madrid.
• The same method of learning used in different programs \[8\],[9],[13] does not offer the same results.
• The error function does not seem to be of great importance.
  In order to access and act directly on the parameters of the optimization process, the multilayer perceptron was programmed with the architecture shown in Figure 2. Two learning methods were used, the conjugate gradient method and the Quasi-Newton method with one-dimensional linear minimization using the Golden method (minimum by intervals) and the Brent method (parabolic interpolation). It was tested with two error functions, the square error and the entropy functions.

![Diagram of a network](image)

**Fig. 2.** Graph Representation of the programmed network

From the tests performed, it can be concluded that:
• The final results may often be considered as local minima.
• In a 10 –5 –10 architecture network, the mean error is still high (> 0.2) although it approximates perfectly several of the patterns.
• The higher the number of hidden processors, the more satisfactory the results, optimal results being obtained with 10 processors (see Graph 1).
• The adjustment is most efficient with the Quasi-Newton method and one-dimensional minimization using the parabolic interpolation method.
• Learning using the conjugate gradient method is slower and offers worse results (mean error 0.5)
• The use of the entropy error function does not offer better results than the mean square error function.
• The results offered by the network are not sensitive to variations in the coding of words (order by frequency rather than by alphabetical order).

Figure 3 presents in graph form the evolution of the mean square error with respect to the number of neurons present in the hidden layer.
5 Conclusions

This paper presents a classification of the main indexing techniques in use today in the field of IRS and describes them briefly. With a view to incorporating a new option into this classification, an IRS prototype has been developed with a technique which the authors consider to be unused at present. It is based on non-self-organising Artificial Neural Networks.

Various architectures have been analyzed for use, concluding that the most suitable one would seem to be the MLP. In this area, the results obtained show that the solution proposed is valid in practical cases of limited dimensions; that is, few documents as opposed to few terms. It has also been observed that the learning method used may be vitally important to the successful operating of the network.

Finally, the tests presented here lead to some optimism as to the possible use of the prototype in real size cases.

References


8. NODELIB. Gary Willian Flakes


13. SNNS, version 4.1, A. Zell et al., University of Stuttgart


Abstract. The paper, after some theoretical hints on the “morphogenetic neuron” proposes the use of this new technique to solve one of the most important themes in robotics, the manipulator kinematics structure representation and the following solution of the inverse kinematics problem. Even if the application has been completed and fully tested with success only on a two degrees of freedom SCARA robot, the first results here reported obtained on a more complex manipulator (spherical) seem to confirm the effectiveness of the approach.

1 Introduction

Research on neural architectures, neurofuzzy, wavelets, indicates the need for the introduction of a more general concept than that of the neural unit, or node, introduced in the pioneering work by McCulloch and Pitts [9]. The neural unit that is widely used today in artificial neural networks can be considered as a non-linear filter. From these basic filter units so-called "integrated" neural architectures may be built, in which many different neural networks cooperate. In order to do research on such neural architectures a description language is needed in which an artificial neural network can be considered as a single, dedicated entity. On the basis of these considerations we use in the present paper a generalization of the concept of neural unit, which will be denoted as morphogenetic neuron. The name “neuron” was adopted because the activation function of such a device is characterised, in the same way as in classical neural units, by a bias potential and by a weighted sum of suitable, in general non-linear basic functions or population code. The attribute “morphogenetic” was chosen because the data generate the weights of linear superposition of functions with special shapes or morphology. The operations in the morphogenetic neuron are divided in two steps. In the first step the neuron learns the structure of the n-dimensional functional space or context. In the second step it learns how to implement invariant forms or rules inside the context. These two-steps appear very useful to study the kinematics synthesis and analysis in robotics.
2 Morphogenetic Neuron

We know that the classical neuron net model is given by expression

\[ u_i = f(\sum_j w_{i,j} s_j) \]  

(1)

Where \( s_j \) are the input, \( w_{i,j} \) are the weights and \( u_i \) are the output.

Fig. 1 shows a classical network scheme.

![Fig. 1. Sample of classical network](image)

Any morphogenetic neuronal net is very similar to the classical neuronal net, with the difference that input is made up by non-linear functions denoted \( \text{basic functions} \)

\[ u_i = \sum_j w_{i,j} \psi_j (s_1, \ldots, s_n) \]  

(2)

![Fig. 2. Sample of morphogenetic neuron](image)
3 Context Determination by Morphogenetic Neuron

In the classical neuronal net as Percetron, Hopfield net or back propagation net and others, the main goal of the net is to generate wanted functions by suitable weights.

In the morphogenetic neuron we have two main goals. The first is to create the context; the second is to calculate the weights for wanted functions. The context is composed by a set of basic functions $\psi_j$ in a Hilbert vector space. We remember that in the Hilbert space the co-ordinates are the basic functions.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{hilbert_space.png}
\caption{Example of Hilbert space with three basic functions}
\end{figure}

4 Horthogonality of Basic Functions and Scalar Product

The scalar product of the basic functions $\psi_j$ and the fundamental tensor are:

$$g_{i,j} = \psi_i \cdot \psi_j = \int_{\Omega} \psi_i(s)\psi_j(s) ds$$  \hspace{1cm} (3)

When $s$ belongs to a finite set of values, the scalar product can be written in the ordinary way:

$$g_{i,j} = \psi_i \cdot \psi_j = \sum_k \psi_i(s_k)\psi_j(s_k)$$  \hspace{1cm} (4)
4.1 Scalar Product and Context

Given the three basic functions $\psi_1, \psi_2, \psi_3$, the fundamental tensor is

$$
\begin{bmatrix}
\psi_1 \cdot \psi_1 & \psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_3 \\
\psi_2 \cdot \psi_1 & \psi_2 \cdot \psi_2 & \psi_2 \cdot \psi_3 \\
\psi_3 \cdot \psi_1 & \psi_3 \cdot \psi_2 & \psi_3 \cdot \psi_3
\end{bmatrix}
$$

(5)

When the three vectors or functions $\psi_1, \psi_2, \psi_3$ are orthogonal one another we have

$$
\begin{bmatrix}
\psi_1 \cdot \psi_1 & \psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_3 \\
\psi_2 \cdot \psi_1 & \psi_2 \cdot \psi_2 & \psi_2 \cdot \psi_3 \\
\psi_3 \cdot \psi_1 & \psi_3 \cdot \psi_2 & \psi_3 \cdot \psi_3
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

and the context is the Euclidean context.

4.2 Example

For the set of functions $\psi_1 = \cos(\alpha) / \sqrt{\pi}$, $\psi_2 = \sin(\alpha) / \sqrt{\pi}$ we have:

$$
\psi_1 \cdot \psi_1 = \frac{1}{\pi} \int_0^{2\pi} \sin^2(\alpha) d\alpha = g_{11} = 1, \quad \psi_2 \cdot \psi_2 = \frac{1}{\pi} \int_0^{2\pi} \cos^2(\alpha) d\alpha = g_{22} = 1
$$

$$
\psi_1 \cdot \psi_2 = \psi_2 \cdot \psi_1 = \frac{1}{\pi} \int_0^{2\pi} \sin(\alpha) \cos(\alpha) d\alpha = g_{12} = g_{21} = 0
$$

So the functions $\psi_1$ and $\psi_2$ are orthogonal.

When the functions are orthogonal, any vectors in the space can be written as the superposition of the basic functions. For the three previous basic functions we can write

$$
\psi = a \psi_1 + b \psi_2 + c \psi_3
$$

(6)

where

$$
\psi \cdot \psi_1 = (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_1 = a \psi_1 \cdot \psi_1 + b \psi_2 \cdot \psi_1 + c \psi_3 \cdot \psi_1 = a
$$

$$
\psi \cdot \psi_2 = (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_2 = a \psi_1 \cdot \psi_2 + b \psi_2 \cdot \psi_2 + c \psi_3 \cdot \psi_2 = b
$$

$$
\psi \cdot \psi_3 = (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_3 = a \psi_1 \cdot \psi_3 + b \psi_2 \cdot \psi_3 + c \psi_3 \cdot \psi_3 = c
$$
In function $\psi$ the three functions $\psi_1$, $\psi_2$, $\psi_3$ are mixed. With the scalar product we can separate (filter operation) function $\psi$ in the three basic functions.

### 4.3 Numerical Example

Given the complex function

$$\psi = 2\psi_1 + \psi_2 + 3\psi_3$$

For the above described properties we have:

$$\psi \cdot \psi_1 = 2\psi_1 \cdot \psi_1 + \psi_2 \cdot \psi_1 + 3\psi_3 \cdot \psi_1 = 2 + 0 + 0 = 2$$

$$\psi \cdot \psi_2 = 2\psi_1 \cdot \psi_2 + \psi_2 \cdot \psi_2 + 3\psi_3 \cdot \psi_2 = 0 + 1 + 0 = 1$$

$$\psi \cdot \psi_3 = 2\psi_1 \cdot \psi_3 + \psi_2 \cdot \psi_3 + 3\psi_3 \cdot \psi_3 = 0 + 0 + 3 = 3$$

We can separate the complex function $\psi$ in its components (filter operation). As an example, we have the Fourier transformation where the basic functions are $\psi_n = \sin(n\omega t)$. The complex function is:

$$\psi = 2\sin(\omega t) + \sin(2\omega t) + 3\sin(3\omega t)$$

that can be shown in Fig. 4

In conclusion, when the basic functions are orthogonal the context is completely defined and any complex functions can be located in the space as vector superposition of the elementary functions as co-ordinates.

![Fig. 4. Vector representation of complex functions](image-url)
When the functions are not orthogonal (non Euclidean space) for the three basic functions \( \psi_1, \psi_2, \psi_3 \) we have:

\[
g_{i,j} = \begin{bmatrix}
\psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2 \\
\psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2 \\
\psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2 & \psi_1 \cdot \psi_2
\end{bmatrix} = \begin{bmatrix}
g_{11} & g_{12} & g_{13} \\
g_{21} & g_{22} & g_{23} \\
g_{31} & g_{32} & g_{33}
\end{bmatrix}
\]

Given the mixed function \( \psi = a \psi_1 + b \psi_2 + c \psi_3 \), we cannot separate the mixed function in its components by the scalar product. In fact we have

\[
\begin{align*}
\psi \cdot \psi_1 &= (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_1 \\
&= a \psi_1 \cdot \psi_1 + b \psi_2 \cdot \psi_1 + c \psi_3 \cdot \psi_1 = a g_{11} + b g_{21} + c g_{31} \\
\psi \cdot \psi_2 &= (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_2 \\
&= a \psi_1 \cdot \psi_2 + b \psi_2 \cdot \psi_2 + c \psi_3 \cdot \psi_2 = a g_{12} + b g_{22} + c g_{32} \\
\psi \cdot \psi_3 &= (a \psi_1 + b \psi_2 + c \psi_3) \cdot \psi_3 \\
&= a \psi_1 \cdot \psi_3 + b \psi_2 \cdot \psi_3 + c \psi_3 \cdot \psi_3 = a g_{13} + b g_{23} + c g_{33}
\end{align*}
\]

With the scalar product we cannot obtain directly the components \( a, b, c \), which, however, can be obtained by the system:

\[
\begin{bmatrix}
g_{11} & g_{12} & g_{13} \\
g_{21} & g_{22} & g_{23} \\
g_{31} & g_{32} & g_{33}
\end{bmatrix} \begin{bmatrix}
a \\
b \\
c
\end{bmatrix} = \begin{bmatrix}
\psi \cdot \psi_1 \\
\psi \cdot \psi_2 \\
\psi \cdot \psi_3
\end{bmatrix}
\]

the solution of which is:

\[
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix} = \begin{bmatrix}
g_{11} & g_{12} & g_{13} \\
g_{21} & g_{22} & g_{23} \\
g_{31} & g_{32} & g_{33}
\end{bmatrix}^{-1} \begin{bmatrix}
\psi \cdot \psi_1 \\
\psi \cdot \psi_2 \\
\psi \cdot \psi_3
\end{bmatrix} = g^{ij} \psi \cdot \psi_j
\]

In conclusion, only when we know the type of space or context, given by the tensor \( g_{i,j} \) we can obtain the components \( a, b, c \) of the vector space. Given the co-ordinates, or basic functions, or states \( \psi_j \) we must calculate the tensor

\[
\frac{1}{g_{i,j}} = g^{i,j}
\]

which can be used to obtain the components, in the Hilbert space, of the function \( \psi \). We remark that, when we know the tensor \( g_{i,j} \) and its inverse, any function can be decomposed (filter process) in its components. The generation of the fundamental
tensor g is the first step (context determination), being the second step to filter a given mixed function $\psi$.

**Example:** Given the complex function

$$\psi = 2\psi_1 + \psi_2 + 3\psi_3$$

we have

$$\psi \bullet \psi_1 = 2\psi_1 \bullet \psi_1 + \psi_2 \bullet \psi_1 + 3\psi_3 \bullet \psi_1 = 2 + 0 + 0 = 2$$

$$\psi \bullet \psi_2 = 2\psi_1 \bullet \psi_2 + \psi_2 \bullet \psi_2 + 3\psi_3 \bullet \psi_2 = 0 + 1 + 0 = 1$$

$$\psi \bullet \psi_3 = 2\psi_1 \bullet \psi_3 + \psi_2 \bullet \psi_3 + 3\psi_3 \bullet \psi_3 = 0 + 0 + 3 = 3$$

where

$$\begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix}^{-1} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{bmatrix}$$

With the inversion of matrix $(g^{-1})$ we can calculate the contravariant vectors $\psi^j$ that we use as a separator or filter in non-orthogonal set of basic functions. Given the mixed state $\psi = a\psi_1 + b\psi_2 + c\psi_3$, we show in Fig. 5 the filter process.

![Fig. 5. Filter process](image)

### 5 Invariants and Basic Tensor

Given the quadratic form

$$S = \sum_{i,j} a_{i,j} \psi_i(s_1,...,s_n) \psi_j(s_1,...,s_n)$$
we have that

\[ \int_{\Omega} S d\omega = \sum_{i,j} a_{i,j} \psi_i(s_1, \ldots, s_n) \psi_j(s_1, \ldots, s_n) ds = \sum_{i,j} a_{i,j} \int_{\Omega} \psi_i \psi_j ds \]

and

\[ \int_{\Omega} S d\omega = \sum_{i,j} a_{i,j} \psi_i \cdot \psi_j = \sum_{i,j} a_{i,j} g_{i,j} \]

by which we can calculate the parameters \( a_{i,j} \). By derivative process we can come back to the invariant \( S \). We prove that the linear combination of the fundamental tensor \( g \) gives the possibility to found invariant in more explicit way as regards to traditional methods. In fact, given the basic functions

\[ \psi_1 = \cos(p), \quad \psi_2 = \sin(p) \]

we have that \( g_{11}, g_{22}, g_{12} \) are

\[ \int_0^q \cos(p)^2 dp = \frac{1}{2} \cos(q) \sin(q) + \frac{1}{2} q, \quad \int_0^q \sin(p)^2 dp = -\frac{1}{2} \cos(q) \sin(q) + \frac{1}{2} q \]

We remark that \( g_{11} + g_{22} = q \) as

\[ \int_0^q \psi_1^2 dq + \int_0^q \psi_2^2 dq = 1 \]

and, with the derivative operator, we obtain the well known invariant

\[ \sin(\alpha)^2 + \cos(\alpha)^2 = \psi_1^2 + \psi_2^2 = 1 \]

Remark. When we know the functions \( \psi_1 = \cos(p), \psi_2 = \sin(p) \) we cannot directly know the relation between the two functions. With the use of the fundamental tensor \( g_{i,j} \), it becomes possible to see directly the relation between the two functions.

5.1 Example

Given the functions

\[ \psi_1 = 2\sin(\alpha) - \cos(\alpha), \quad \psi_2 = \sin(\alpha) + 4 \cos(\alpha) \]

connected one with the others by the ellipse given in Fig. 6.
With the scalar product we can calculate the coefficients in the form:

$$a_{11} \psi_1 \cdot \psi_2 + 2a_{12} \psi_1 \cdot \psi_2 + a_{22} \psi_2 \cdot \psi_2 = a_0 \alpha$$  \hspace{1cm} (7)

**Fig. 6.** Relation between the functions $\psi_1$ and $\psi_2$

In fact

$$\int_0^q (2 \sin(p) - \cos(p))^2 \, dp = \frac{-3}{2} \cos(q) \sin(q) + \frac{5}{2} q + 2 \cos(q)^2 - 2$$

$$\int_0^q (\sin(p) + 4 \cos(p))^2 \, dp = \frac{15}{2} \cos(q) \sin(q) + \frac{17}{2} q - 4 \cos(q)^2 + 4$$

$$\int_0^q (2 \sin(p) - \cos(p))(\sin(p) + 4 \cos(p)) \, dp = -3 \cos(q) \sin(q) - q - \frac{7}{2} \cos(q)^2 + \frac{7}{2}$$

For (7) we have

$$\begin{cases} 
-a_{11} + \frac{5}{2} a_{12} + \frac{17}{2} a_{22} + a_0 = 0 \\
\frac{15}{2} a_{22} - \frac{3}{2} a_{12} - 3a_{11} = 0 \\
-4a_{22} + 2a_{12} - \frac{7}{2} a_{11} = 0 \\
4a_{22} + 2a_{12} + \frac{7}{2} a_{11} = 0
\end{cases}$$
which leads to:

\[ a_{11} = 0.049, a_{11} = 0.209, a_{22} = 0.06, a_0 = 1 \]

So we have the invariant form

\[ a_{11}\psi_1\psi_1 + 2a_{12}\psi_1\psi_2 + a_{22}\psi_2\psi_2 = a_0 \]

that is the ellipse given in Fig. 6. We can generate invariant forms also with the filter process. In fact, with the set of functions \( \psi_1(x), \ldots, \psi_n(x) \), we can create a new set of functions \( G_1(\psi_1(x), \ldots, \psi_n(x)), G_2(\psi_1(x), \ldots, \psi_n(x)) \) if \( G_j = a_1 G_1 + a_2 G_2 + \ldots + a_{i-1} G_{i-1} + a_i G_i + \ldots + a_n G_n \) functions \( G_j \) can be written as a superposition of the basic functions \( G_1, G_2, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n \). When we know function \( G_j \), the computation of the coefficients \( a_i \) will be obtained by the inverse of the fundamental tensor \( g \) as we have seen above.

6 Learning Process and Fundamental Tensor \( g_{i,k} \) Construction

In many cases we cannot know the formal expressions of the basic functions \( \psi_1(x), \psi_2(x), \ldots, \psi_n(x) \). In general, for given sets of the independent variables \( x_1, \ldots, x_n \) in \( n \)-dimensional space we know \( p \) samples of the basic functions as we can see in Table 1.

<table>
<thead>
<tr>
<th>Samples</th>
<th>( \psi_1(x) )</th>
<th>( \psi_2(x) )</th>
<th>\ldots</th>
<th>( \psi_n(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>First sample</td>
<td>( x_1 )</td>
<td>( \psi_1(x_1) )</td>
<td>( \psi_2(x_1) )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>Second sample</td>
<td>( x_2 )</td>
<td>( \psi_1(x_2) )</td>
<td>( \psi_2(x_2) )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>Third sample</td>
<td>( x_3 )</td>
<td>( \psi_1(x_3) )</td>
<td>( \psi_2(x_3) )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>Fourth sample</td>
<td>( x_4 )</td>
<td>( \psi_1(x_4) )</td>
<td>( \psi_2(x_4) )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>( p )-th sample</td>
<td>( x_p )</td>
<td>( \psi_1(x_p) )</td>
<td>( \psi_2(x_p) )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

With the samples of the basic functions it is possible to create an approximation of the fundamental tensor.

\[ g_{i,j} \approx \psi_1(x_1)\psi_j(x_1) + \psi_1(x_2)\psi_j(x_2) + \ldots + \psi_1(x_p)\psi_j(x_p) \]
With this approximate tensor, with a “generalisation process” we can have the filter process and we can create invariants.

7 Biological Realisation of the Scalar Product and of the Filter Process

In Fig. 7 we show the single –neuron firing rates in monkey MT see David J.Heeger [22] and others.

![Diagram of neuron response](image)

**Fig. 7.** Example of scalar product in the neuron response

In Fig. 7 the neuron uses the scalar product to generate the response. The neuron sensible to arrows with direction up replies in the case of 100% of coherence.

8 Biological Analogy of the Basic Function Approach

8.1 Population Code and Basic Functions

In Fig. 8 we show the control of arms by basic functions see A.Pouget [23]

The positions of the tiger are the basic functions. All the positions or population codes (worlds) are elementary states. The superposition of all the positions gives a mixed state of the TIGER concept. In paper [24] single neurons in the macaque posterior inferotemporal cortex may be tuned to a DICTIONARY (population code) of thousands of complex shapes or points of view or basic functions (words).
9 Kinematics Synthesis and Analysis in Robotics

Inside an authors’ research theme whose aim is to apply new mathematical approaches in robotics field [3] [4] [5], in the present paper we propose the use of the morphogenetic neuron to find a polynomial representation of an industrial manipulator.

\[
\text{join angles } J \text{ of the arms} \\
J = \Sigma w_i \psi_i (V, P)
\]

In Fig. 8 the scheme representing the basic functions for different points of view (worlds) for a tiger is suggested.

One of the greatest problems in robotics is the mathematical representation of the kinematics structure of a manipulator. Some authors propose to describe the robot links as matrices, so that, at the end of the process, the robot is nothing but a product of a series of matrices, each one representing the mutual relative position of the rigid bodies constituting the manipulator. Another approach proposed by literature is to consider the robot as an open chain of vectors. Others, in order to solve the robot kinematics, utilise the graph theory: every branch of the graph is a transformation linking the relative co-ordinates used to describe the rigid bodies position. For simple robots some authors propose a graphical solution and solve the robot kinematics only by geometrical considerations. (For all these approaches we suggest the reading of papers [1],[14],[15],[29] of the references which gives a good overview of the different methodologies).

Fig. 8. Gaussian basic functions to control joint angles in the arm

In Fig. 9 the scheme representing the basic functions for different points of view (worlds) for a tiger is suggested.

Variable V is the vector that encode an object’s location in eye – centre space In input
In particular, the method well described in [14], [15] consists in associating to the robot a polynomial form, whose solution allows to pass from the robot workspace (the position of the gripper) to the joint space (the rotation of the links). These authors analyse various methods for solving systems of polynomial equations and discuss the fields in which the various approaches can be used with success according to the number of variables, unknowns, kinematics structure, etc. The conclusion is that there is not an optimal method to build the polynomial form, nor to solve it, but the researcher has to choose one or another methodology according to his own experience and following some qualitative rules given in the paper.

In the following, after some hints on the kinematics problems in robotics, we will show how it is possible to write polynomial forms (invariants) associated to the robot by means of the methodology given above in a very quick and almost automatic way. After the building of a series of polynomial invariant equations, it is possible to solve them by applying one of the methods for the polynomial equations solutions.

10 Hints about the Kinematics Problem in Robotics

The pose (position and orientation) of the gripper of a robot can be described by two different co-ordinates sets: the first one (typically formed by the Cartesian co-ordinates and the Euler angles), defines the so called working space, and utilises the co-ordinates as regard to a laboratory, or external, frame; the second set, constitutes the joint space, and uses the relative position of the robot links, that is its joint (or internal) variables. As relation which specifies the external co-ordinates as functions of the internal ones is typically non linear, it can give origin to multiple solutions, it can contain singularities, its solution and/or inversion is very difficult, and this can be rightly classified as one of the very felt problem in the field of robotics.

Due to the importance of the subject (which is linked with two very important practical aspects of robotics: the path and task planning for a manipulator and the implementation of the solving equations into the manipulator controller to effectively program and move the robot), research in this field has been continuing for many years.
years, with the classic approach being the use of mathematical algorithms which describe the kinematics relationships of the manipulator gripper pose with respect to its joint variables. To allow trajectory control in hand co-ordinates, the inverse kinematics equations are usually derived from the direct kinematics ones.

Hereafter the robot kinematics problem is related referring to the usual industrial manipulators which can be described as open chains of rigid bodies, and from this starting point, it can be explained, in general terms, by means of the following notation:

\( \mathbf{u} \): this is the vector of the external co-ordinates or end-effector co-ordinates (displacement and orientation of the robot gripper in its working space);

\( \mathbf{q} \): this is the vector of the inner co-ordinates or joint variables;

Relation

\[ \mathbf{u} = f(\mathbf{q}) \]

(named “position equation”) describes the link between the inner co-ordinates of the manipulator and the external ones. Its structure depends on the geometrical architecture of the robot, in particular on some dimensions of the links and on the kind of coupling pairs, and allows to find \( \mathbf{u} \) when \( \mathbf{q} \) is known (direct kinematics problem).

The inverse transformation can be formally expressed by means of the following relation:

\[ \mathbf{q} = f^{-1}(\mathbf{u}) \]

Many approaches can be followed to solve the above equations (see references). However, one of the most interesting consists in determine a polynomial form which has the same solution of \( \mathbf{u} = f(\mathbf{q}) \). The methodology is well documented in\(^{15}\) where, the authors show that with a proper choice of variables, it is always possible (even if not simple) to obtain a solving polynomial form.

Once the polynomial form has been recognised, the inversion (solution) can be gained by classical (analytical or numerical) methods.

### 11 Application to a SCARA Robot

![Fig. 10. Scheme of SCARA robot](image-url)
It is easy to write the equations linking the angular position of the links \( q = (\alpha, \beta) \) with the position of the gripper in the cartesian space \( u = (X, Y) \):

\[
X = L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) \\
Y = L_1 \sin(\alpha) + L_2 \sin(\alpha + \beta)
\]

With the basic functions

\[
\psi_1 = L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta), \quad \psi_2 = L_1 \sin(\alpha) + L_2 \sin(\alpha + \beta)
\]

For \( L_1 = 1 \) and \( L_2 = 1 \), we have the fundamental tensor for the variation of the angle \( \alpha \):

\[
g_{11} = \int_0^p \psi_1 \psi_1 = \left[ \cos(\alpha) + \cos(\alpha + \beta) \right]^2 d\alpha
\]

\[
g_{22} = \psi_2 \cdot \psi_2 = \int_0^p \left[ \sin(\alpha) + \sin(\alpha + \beta) \right]^2 d\alpha
\]

evaluating the fundamental tensor and by simple computations we have:

\[
g_{11} + g_{22} = 2\alpha + 2\alpha \cos \beta
\]  

(8)

By derivation respect to \( \alpha \) in the right part of (8) we obtain the invariant

\[
X^2 + Y^2 = 2 \left( 1 + \cos(\beta) \right)
\]

For \( L_1 \) and \( L_2 \) as free parameters we have:

\[
X^2 + Y^2 = L_1 + L_2 + 2 L_1 L_2 \cos(\beta)
\]

By varying \( \beta \) angle we have other fundamental tensors and the invariant

\[
X^2 + Y^2 = L_2^2 - L_1^2 + 2 L_1 \left[ X \cos(\alpha) + Y \sin(\alpha) \right]
\]

By the two above invariants, when we know \( X \) and \( Y \), \( L_1 \) and \( L_2 \), we can calculate angles \( \alpha \) and \( \beta \).

12 Application to a Spherical Robot

Kinematics equations:

\[
X = \left[ L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) \right] \cos(\gamma)
\]

\[
Y = \left[ L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) \right] \sin(\gamma)
\]

\[
Z = \left[ L_1 \sin(\alpha) + L_2 \sin(\alpha + \beta) \right]
\]
With the basic functions

\[ \psi_1 = [ L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) ] \cos(\gamma), \]
\[ \psi_2 = [ L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) ] \sin(\gamma) \]
\[ \psi_3 = [ L_1 \sin(\alpha) + L_2 \sin(\alpha + \beta) ] \]

For the variation of \( \gamma \) we have

\[ g_{11} = \int_{0}^{\gamma} \psi_1(\alpha, \beta, \gamma)^2 d\gamma, \quad g_{22} = \int_{0}^{\gamma} \psi_2(\alpha, \beta, \gamma)^2 d\gamma \]

obtaining the invariant

\[ X^2 + Y^2 = [ L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) ]^2 \]

which is a partial invariant.

For \( B = \pm \sqrt{X^2 + Y^2} \), we reduce the spherical robot to the SCARA robot with the equations:

\[ B = L_1 \cos(\alpha) + L_2 \cos(\alpha + \beta) \]
\[ Z = L_1 \sin(\alpha) + L_2 \sin(\alpha + \beta) \]

That we can solve.

References


Analysis and Verification Queries over Object-Oriented Petri Nets

Milan Češka, Vladimír Janoušek, and Tomáš Vojnar

Department of Computer Science and Engineering, Brno University of Technology
Božetěchova 2, CZ–612 66 Brno, Czech Republic
{ceska, janousek, vojnar}@dcse.fee.vutbr.cz

Abstract. In literature there have been proposed many different ways of asking formal analysis or verification questions to be answered over state spaces of various kinds of models. In this paper, we discuss how these methods can be accommodated for the case when the systems being examined are modelled by object-oriented Petri nets (OOPNs). We deal with the OOPNs associated with the PNtalk language and tool.

1 Introduction

The PNtalk language and tool based on a certain kind of object-oriented Petri nets (OOPNs) have been developed at the Brno University of Technology in order to support modelling, investigating, and prototyping concurrent and distributed object-oriented systems. PNtalk supports intuitive modelling of all the key features of such systems, including object-orientatedness, message sending, parallelism, and synchronisation. There has already been implemented a tool for simulating systems described by OOPNs. Currently we are working both on an implementation of a tool which should allow us to run OOPN-based prototypes in a truly distributed way as well as on state spaces-based methods for formally analysing and verifying properties of systems modelled by OOPNs.

Formal analysis and verification can be considered complementary to simulation because although we are not always able to fully verify or analyse the behaviour of a system, even partial analysis or verification can reveal some errors which tend to be different from the ones found by simulation. Among the different approaches to performing formal analysis or verification, generating and exploring suitably represented state spaces appears to be the most straightforward way for the case of OOPNs.

In [2,10], we have carefully examined some special issues that turn out to be related to generating state spaces of OOPNs (as it is also briefly mentioned below). This paper then provides a more detailed discussion of using state spaces of OOPNs. Particularly, we examine how the various existing ways of asking analysis and verification questions can be accommodated for the case when they are to be answered over state spaces of OOPN-based models.

In the following, we firstly introduce the main concepts of OOPNs and PNtalk. Next, we present a brief note about state spaces of OOPNs. Subsequently, we proceed to a discussion of asking analysis and verification questions...
to be answered over these state spaces. We finish the paper by some concluding remarks. Due to space limitations, the paper is written in an informal way—a more formal presentation can be found in [10].

2 Key Concepts of OOPNs

The OOPN formalism is characterized by a Smalltalk-based object-orientation enriched with concurrency and polymorphic transition execution, which allows message sending, waiting for and accepting responses, creating new objects, and performing primitive computations [6].

In the following, we explain the main principles of the structure and behaviour of OOPNs. A deeper introduction to the OOPN formalism can be found in [1] and the formal definition of OOPNs in [10].

As an example illustrating the notation of OOPNs, we present a model of the system of distributed philosophers from [6]. The class DPhil describing distributed philosophers is shown in Fig. 1. Distributed philosophers differ from the classical ones in not having a shared table which could be used for exchanging forks. Instead, they have to negotiate about forks via message sending using the methods giveLFork and giveRFork. A run of the system of distributed philosophers is to be initiated by creating an object of the class DDinner (Fig. 2). This object recursively creates (by means of the method makePh:from:to:) a ring of philosophers interconnected by references established via the methods leftNb: and rightNb:.

2.1 The Structure of OOPNs

An object-oriented Petri net is defined on a collection of elements comprising constants, variables, net elements (i.e. places and transitions), class elements (i.e. object nets, method nets, synchronous ports, and message selectors), classes, object identifiers, and method net instance identifiers. An OOPN has its initial class and initial object identifier, as well. The so-called universe of an OOPN contains (nested) tuples of constants, classes, and object identifiers.

An OOPN class (as e.g. DPhil) is given by its object net, its sets of method nets and synchronous ports, and a set of message selectors corresponding to its methods and ports.

Object nets consist of places and transitions. Each place has some (possibly empty) initial marking. Each transition has conditions and preconditions (i.e. inscribed testing and input arcs), a guard, an action, and postconditions (i.e. inscribed output arcs). Object nets describe what data particular objects encapsulate (as e.g. references to neighbours stored in leftNb and rightNb of the object net of DPhil) and what activities the objects may exhibit on their own (such as the possibility to get hungry, to start and stop eating, or to get a left or a right fork).

Method nets resemble object nets but, additionally, each of them has a set of parameter places and a return place. Method nets can access places of the appropriate object nets, which allows running methods to modify the states of the
objects which they are running in. Method nets (as e.g. leftNb: or giveLFork) specify how objects asynchronously respond to received messages.

Synchronous ports are special transitions that cannot fire alone but only dynamically fused to some regular transitions. These transitions (possibly indirectly) activate the ports via message sending from their guards. Each port has a set of conditions, preconditions, and postconditions over places of the appropriate object net, a guard, and a set of parameters. Parameters of an activated port $s$ can be bound to constants or unified with variables from the level of the transition or port that activated $s$. Synchronous ports (not used in our example) allow us to remotely test and change states of objects in an atomic way.

2.2 The Dynamic Behaviour of OOPNs

A state of an OOPN can be encoded as a marking, which can be structured into a system of objects. Thus the dynamic behaviour of OOPNs corresponds to an evolution of a system of objects. An object of a class $c$ is a system of net instances that contains exactly one instance of the object net of $c$ and a set of currently running instances of method nets of $c$. Each net instance entails its identifier and a marking of its places and transitions. A marking of a place is a multiset.
of tokens coloured by some elements of the universe. A marking of a transition \( t \) is a set of records about methods invoked from \( t \) and not yet terminated.

For a given OOPN, its initial marking represents a single, initially marked object net instance from the initial class. A change of a marking of an OOPN is the result of an occurrence of some event. Such an OOPN event is given by (1) its type, (2) the identifier of the net instance it takes place in, (3) the transition it is statically represented by, and (4) the binding tree containing the bindings of the variables used on the level of the involved transition as well as within all the synchronous ports (possibly indirectly) activated from that transition. There are four kinds of events according to the way of evaluating the action of the appropriate transition: A – an atomic action involving trivial computations only, N – a new object instantiation via the message new, F – an instantiation of a Petri-net described method, and J – a method net instance termination.

Firing an A event means removing and/or adding some tokens from/to the marking of certain places according to the arcs of the involved transition and synchronous ports and according to the appropriate binding. An N event differs from an A event by additionally creating and initializing a new object. An F event starts a new method, initializes it, and puts arguments into its parameter places. The firing transition is marked by a reference to the new method net instance and its output arcs are ignored. A J event retrieves a result token from the return place of a method net instance, deletes the instance and the transition marking element referencing it, and performs the output part of the appropriate transition. Garbage collection is a part of every event.

3 The Notion of State Spaces of OOPNs

In [2,10], we have examined some special issues related to generating state spaces of OOPNs. We have especially discussed the problem of state space explosion.
due to generating many (semantically redundant) states differing only in the names of the involved net instances, where the names allow us to separate the instances and to express references among them, but they cannot significantly influence the future behaviour of the system being explored. We have examined sources, consequences, as well as possible solutions of this problem, which we call the naming problem. Although the naming problem is not exclusively specific to OOPNs, we have argued that it manifests itself in a quite severe way in the area of OOPNs (and similar formalisms), and so solving it appears to be a necessity here rather than an option.

Inspired by the methods listed in [2,10], we have proposed and compared two methods for dealing with the naming problem in the context of OOPNs, namely sophisticated naming rules and name abstraction. Sophisticated rules for assigning names to newly arising instances attempt to decrease the degree of nondeterminism potentially present in the management of identifiers of dynamically arising and disappearing instances and thus to decrease the number of reachable states. Name abstraction is a fully transparent application of the concept of symmetrically reduced state spaces to solving the naming problem taking into account some issues specific to OOPNs.

When using name abstraction, we replace full state spaces of OOPNs by (complete) name abstracted state spaces whose states represent classes of OOPN markings that are equal up to renaming. Similarly, transitions of such state spaces represent classes of renaming equivalent OOPN events.

4 Using State Spaces of OOPNs

In literature there have already been introduced many different kinds of query or specification languages that can be used for stating analysis or verification questions within state spaces-based formal analysis or verification of concurrent systems (see e.g. [7,8,4,5,9,3]). The different ways of asking analysis or verification questions have different pros and cons and may be advantageous in different situations. In the following, we will discuss how they can be adapted (without affecting their common advantages, of course) for the case when the systems under investigation are modelled by OOPNs. Namely, we will consider: (1) different kinds of state space statistics, (2) versatile state space query languages for user-controlled traversing through state spaces, (3) instrumenting models and using property directives, labels, or automata, and (4) using high level specification languages (such as temporal logics).

Below, we will take into account that we can be interested not only in global properties of modelled systems, but also in local properties related to the particular instances involved. We will also consider the fact that it is desirable (as we will see) to be able to avoid referring to the concrete names of instances when specifying properties to be validated.

In all the considerations, we will bear in mind that it should be possible to answer questions expressed in any of the described ways by means of examining (possibly name-abstracted) state spaces of the appropriate OOPNs. We will take
care so that standard algorithms for checking different kinds of properties could be used in the new setting with as few as possible changes.

Still before we concentrate on the particular ways of asking analysis and verification questions over OOPN-based models and their state spaces, we will briefly discuss how we understand net instances to behave in time. This is necessary for a correct understanding of the spirit of the various instance-oriented properties which we will introduce in the following.

4.1 Instances and Their Behaviour in Time

From the point of view of examining temporal properties of systems, it is suitable to understand instances as entities with a unique identity and with a certain behaviour in time. In the following, the identity of an instance will not be considered the same thing as a net instance identifier that can be used for (temporarily) referring to this instance in a model. We will suppose every net to always have infinitely many instances. However, at a given point in time most of these instances will be lifeless and inaccessible. At the beginning of a run of an OOPN every net instance will be in the same special local state, which we can call a “prenatal” state. Creating a new net instance in some OOPN will then be viewed as selecting an arbitrary instance in the prenatal state, assigning a currently free net instance identifier to it (we consider the true identity of an instance to be invisible), and setting the local state of the instance to the appropriate initial marking. Similarly, deleting an instance will be interpreted as releasing the corresponding identifier (enabling its future reuse) and changing the local state of the deleted instance to a special “postmortem” state, which the instance cannot leave any more and which is the same for all deleted instances.

Formally, if we want to examine a behaviour of some (generally understood) instance in time, we deal with a sequence of equally identified OOPN net instances in a sequence of markings within a state space path such that an instance with the given identifier is present in every marking of the given sequence [10].

4.2 Statistics of State Spaces of OOPNs

In the context of OOPNs, we can use probably all the general state space statistics (such as numbers of states, numbers of strongly connected components, sets of home states, etc.) without any significant changes. On the other hand, Petri net-based statistics (as e.g. bounds of places or liveness of transitions) have to be accommodated for the OOPN-specific style of structuring models. We can also add some new statistics primarily connected to dealing with instances and objects in OOPNs.

Statistics Primarily Based on Instances. The basic statistics that can be related to dealing with instances and objects in OOPNs include maximum numbers of concurrently existing instances of various nets, maximum numbers of instances of particular method nets running within a single object, and so on.
Another interesting measure can be whether there arise some persistent, transient, and/or perspective instances of a certain net, or—more precisely—how many of such instances can be found in a single reachable state. We call an instance involved in some marking a \textit{persistent instance} if it cannot be deleted in any possible future of the marking. On the other hand, we call it a \textit{transient instance} if it is eventually deleted in every possible future. Finally, we call it a \textit{perspective instance} if it can, but need not, be deleted. Since the active life of an instance always begins with its creation, we can also introduce the notion of an event creating a persistent, transient, or perspective instance, which allows us to relate persistence, transientness, and perspectivness to the whole active life of instances arising in a running OOPN.

To present some examples, we can say that there can be created only persistent instances of the object net of the class of distributed philosophers in the model from Figures 1 and 2. Moreover, if we restrict ourselves to (at least) weakly fair behaviours, there can arise only transient instances of the method nets \texttt{makePh:from:to:}, \texttt{leftNb:}, and \texttt{rightNb:} that initialize the system of distributed philosophers. This means that, under the weak fairness assumption, the initialization of the system of distributed philosophers always terminates.

Computing maximum numbers of concurrently existing instances of particular nets or maximum numbers of method net instances running within a single object is relatively easy. It suffers from the same problem as most other comprehensive state space statistics, only. Namely, it gathers information from all over a state space, which complicates the possibility of reducing the state space.

Checking instances to be persistent, perspective, or transient is generally more complicated. This is because we have to examine the individual life of particular instances. When doing this, we have to take into account that the same identifier can be used to refer to different instances in different states. Therefore we cannot simply interrelate two local states of instances referred to via the same identifier in two different states without examining what state space paths there are between the two states. Moreover, in the case of using name-abstracted state spaces, we have to cope with the fact that when going from one state into another, the instance which we are interested in can be referred to by another identifier in the target state than in the source state.

For tracking the individual life of particular instances, we can—more or less explicitly—exploit the approach used within model checking the temporal logic called Indexed CTL* (ICTL*) over symmetrically reduced state spaces. (We will mention ICTL* in more detail later on.) The principle of this approach is that we separately track the particular instances through the so-called “indexed state spaces”. States of such state spaces have the form of tuples \((s, id)\) consisting of a state \(s\) of the appropriate base state space and of an identifier \(id\) that can be used to access the local state of some instance existing in \(s\). Transitions of indexed state spaces not based on name-abstracted base state spaces interconnect \((s_1, id)\) with \((s_2, id)\) if there is a transition from \(s_1\) to \(s_2\) and the given instance is not deleted within this transition. When using name abstraction, we have to further take into account that \(id\) can change to some \(id'\) in \(s_2\) as it is illustrated in Fig. 3.
(for more details see [3,10]). Obviously, it is then easy to examine the behaviour of any instance once we know how it is identified in some state.

a CNA state space fragment:

![CNA state space fragment diagram]

an indexed state space fragment:

![Indexed state space fragment diagram]

**Fig. 3.** The principle of indexing complete name-abstracted (CNA) state spaces

To check whether some identifier *id* refers to a persistent, perspective, or transient instance within some marking *M*, to check whether there can arise some persistent, perspective, or transient instances, or to check whether there can arise only perspective or transient instances of some net, we can use the later mentioned modification of ICTL* (and its model checking algorithm) or certain specialized methods based on series of depth first searches over the appropriate indexed state spaces (or their graphs of strongly connected components) as proposed in [10]. The time complexity of such techniques is usually linear in the maximum number of concurrently existing instances of the net being examined and in the size of the appropriate state space (with unfolded transition labels and self-symmetries in the case of complete name-abstracted state spaces). The situation complicates even more when we intend to apply some fairness assumptions, which is again close to the problem of model checking ICTL* formulae. On the other hand, let us add that checking whether all instances of some net arising in some OOPN are persistent is easier than the other cases—it is enough to check that no instance of the appropriate net can be deleted.

**Bounds of Places in OOPNs.** Due to the object-oriented style of structuring OOPN-based models, we can distinguish *instance place bounds*, *object place bounds*, and *overall place bounds*. In the case of instance place bounds, we examine markings of particular places separately for each instance of the appropriate net. In the case of object place bounds, we summarize markings of the same places of all the instances of the appropriate net running within a single object. Finally, in the case of overall place bounds, we entirely ignore the structuring of running models into objects and instances.

Furthermore, in OOPN marking element and multiset place bounds [7], we should avoid using instance identifiers because their concrete values are nor-

---

1 A marking element place bound can inform us about how many tokens with certain colours can appear in certain places. An upper/lower multiset set place bound is the
mally not important and hard (or impossible) to predict. Dealing with concrete instance names would especially be harmful in the context of name-abstracted state spaces where it would force us to (at least partially) unfold the appropriate renaming equivalence classes. To avoid dealing with the concrete identifiers of instances, we should define marking element and multiset place bounds w.r.t. some (default or user-defined) function that could transform the instance identifiers used within some marking to some other renaming-insensitive values. For defining such a function, the users could use, for example, the set of specialised functions for querying OOPN states and events which we will describe later on.

Let us now present several examples. The best upper integer instance bound of the place \texttt{leftNb} from the object net of the class \texttt{DPhil} from Fig. 1 is 1, which shows that each philosopher knows about at most one left neighbour at any time. If we represent instances by their types, the best upper multiset instance bound of the place \texttt{philosophers} from the object net of the class \texttt{DDinner} is \(5 \cdot \texttt{DPhil}\), which corresponds to the number of philosophers intended to be created in our system of distributed philosophers.

**OOPN Transition-Oriented Properties.** We will now have a look at the notions of dead and live OOPN transitions, which represent two out of the most commonly used Petri net transition-oriented properties. Due to the object-oriented style of structuring OOPN-based models, we can distinguish transitions to be overall, instance, or object dead or live.

The notion of overall dead and live OOPN transitions can be defined in a relatively straightforward way ignoring the object-oriented structuring. We say that a transition is \textit{overall dead} in some net if there is no reachable marking from which the transition could be fired in an arbitrary way within an arbitrary instance of the given net. Similarly, we consider a transition to be \textit{overall live} in some net if from every reachable marking there can be reached another marking in which there is enabled some event based on the given transition within an instance of the given net.

A more advanced notion of transitions to be dead or live can be obtained when we define transitions to be dead or live within particular net instances. Namely, we say that a transition \(t\) from a net \(n\) is \textit{instance dead} if there can be created an instance of \(n\) such that \(t\) can never be fired within this instance. Next, we consider a transition \(t\) from a net \(n\) to be \textit{weakly instance live} if there cannot be created an instance of \(n\) which could get (without being deleted) into a state from which it would be impossible to eventually fire \(t\) in an arbitrary way within this instance. Finally, we say that a transition \(t\) from a net \(n\) is \textit{strongly instance live} if instances of \(n\) cannot be deleted once they are created and it is always possible to eventually fire \(t\) within any of them.

To illustrate the above introduced notions, we can say that the transitions \texttt{start_eating} and \texttt{stop_eating} from the object net of the class \texttt{DPhil} from Fig. 1 are strongly instance live within the appropriate system. The transition \texttt{init} smallest/largest multiset \(M\) such that every reachable marking of the given place is a sub-multiset/super-multiset of \(M\).
from the object net of the class \texttt{DDinner} from Fig. 2 is not weakly instance live nor instance dead.

We can add that we can use the standard mechanisms for checking transitions of coloured Petri nets to be dead or live \cite{7} for checking OOPN transitions to be overall dead or live within certain nets. Checking transitions to be instance dead or live is much more difficult because we have to track the individual life of particular instances throughout the appropriate state spaces. For this reason, we can again (more or less explicitly) exploit the notion of indexed state spaces and/or their graphs of strongly connected components \cite{10}. The computational complexity is then multiplied by (a multiple of) the maximum number of concurrently existing instances of the appropriate net and it is also increased by unfolding self-renaming symmetries and transition labels of complete name-abstracted state spaces.

4.3 Querying States Spaces of OOPNs

A universal state space query language for user-controlled traversing through state spaces of OOPNs can be inspired by any of the already proposed languages of this kind, such as the one associated with the tool Design/CPN \cite{7,9}. However, a universal state space query language for the context of OOPNs has to provide some specialized tools for querying states and events of OOPNs respecting their highly specific structuring. Note that such tools can then also be used for describing terms embedded in temporal logic formulae specifying properties of systems to be verified by means of their OOPN-based models, for expressing system invariants to be checked, or for a fine-grained specification of legal termination states or progress states.

The main problem to be solved when querying states and events of OOPNs stems from the dynamism of OOPNs. We have to prepare tools for exploring properties of sets of states and events in a way which respects the fact that sets of the just existing instances, their names and relations can be different in every encountered state and cannot be normally predicted.

In a prototype of an OOPN state space tool, we have introduced two groups of functions to solve the above problem. Firstly, we use the so-called \textit{instance querying functions} that allow us to initially obtain sets of the just existing instances of certain nets (or to start with the initial object) and to further recursively derive sets of the net instances or constants which are directly or transitively referenced from the already known instances. There also exists a function for deriving sets of the method net instances that are just running over some given objects. All these functions avoid an exact knowledge of the just existing instances by working with not in advance restricted sets of them. In order to obtain the required characteristics of states, instance querying functions are intended to be combined with the so-called \textit{set iterating functions} working over sets of instances and constants. We can, for example, take all the just existing instances of some net, select the ones which contain some constant in some place, and then proceed by exploring some other instances referenced from the selected ones.
Let us present an example of examining states of OOPNs. Below, we define a predicate `eating_neighbours` for deriving the set of eating philosophers having an eating neighbour from a given state of the model of distributed philosophers. We use a Prolog notation of instance querying and set iterating functions as within our prototype tool for working with state spaces of OOPNs. The current state is considered to be implicit. The query `inst(Cs,Ns,Is)` returns the set `Is` of the just existing instances belonging to the nets from the set `Ns` and running within objects belonging to the classes from `Cs`. The query `token(Is,Ps,Cs,Ms)` returns the set `Ms` of tokens belonging to the classes from `Cs` and stored in the places from `Ps` within the instances from `Is`. The query `select(S1,X,P,S2)` selects all the elements `X` from `S1` which fulfil the predicate `P` over `X`.

```
eating_neighbours(EN) :-
    inst([dPhil],[[dPhil,object]], I),
    select(I,P,(token([P],[eating],all,E1),empty(E1,false),
        token([P],[leftNb,rightNb],all,LR),
        token(LR,[eating],all,E2),empty(E2,false)), EN).
```

The predicate `eating_neighbours` can be used to check the correctness of the proposed system that should not allow two neighbours to eat at the same time. This can be checked by a suitable state space query which evaluates the predicate `eating_neighbours` over every reachable state and not succeeds in finding a state where it would hold for a non-empty set. A more abstract approach would be checking the validity of the CTL formula $\forall G eating_neighbours(\emptyset)$.

### 4.4 Instrumenting OOPNs

As in the case of models described by other modelling languages, OOPN-based models can be extended by some constructions that will make it possible or easier to analyze or verify certain properties of the modelled systems. Such constructions can be created by means of the normal constructs of OOPNs, possibly extended or supplemented with some additional labels, directives, etc. In OOPNs, we can use all the common kinds of property labels and directives as well as property automata. We will now have a look at how they can be applied.

Let us start with assertions and invariants, which are probably the simplest (yet quite useful) kind of property directives. In the context of OOPNs, assertions can have a form of special guards which do not restrict the firability of transitions or ports, but once a transition or a port with an associated assertion is being fired the assertion must hold (otherwise an error message is generated). Invariants of OOPN-based models can be expressed in the common way as conditions over states that are supposed to hold in every reachable state (if this is not the case, an error is announced). Particularly, OOPN state invariants can be specified using the above mentioned sets of functions for querying states of OOPNs.

To allow illegal termination states of OOPNs to be identified and announced within generating a state space, we can introduce a special kind of labels for declaring the so-called end-state places. A terminal state will then be considered
legal if and only if all the instances existing in that state have some tokens in their end-state places. Additionally, we may require that such a state must not contain instances of the nets that do not involve any end-state places. When end-state place labels are not sufficient to distinguish legal and illegal termination states, we may apply general, arbitrarily complex end-state conditions. Such conditions can be described via the functions for querying states of OOPNs mentioned in the previous subsection. An end-state condition can mark a state to be a legal termination state if it evaluates to true over such a state.

In the context of OOPNs, it further seems to be useful to introduce some instance deletion place labels (or their generalization in the form of instance deletion conditions). By specifying instance deletion places in a certain net, we can say that instances of this net can be legally deleted only if the specified places are marked (otherwise an error will be announced). On the other hand, we can introduce labels indicating that all instances of certain nets are supposed to be persistent, i.e. that instances of these nets should never be deleted.

As a counterpart to the end-state place labels, we can introduce yet another kind of labels of places that will allow us to specify the so-called progress places. A loop encountered in a state space will then be considered legal if and only if it goes through a "progress" state which is characterized by the fact that it contains an instance whose progress places are marked. We can also introduce progress conditions as an analogy with the above mentioned end-state conditions.

For illustration, let us consider joining the progress label with the place eating from the object net of the distributed philosophers from Fig. 1. Using this progress place, we will find out that the system of distributed philosophers exhibits undesirable non-progress loops even under the strong fairness assumption. This is because the philosophers may keep exchanging forks via the transitions getLFork and getRFork and the methods giveLFork and giveRFork without enabling at least one philosopher to start eating.

Unfortunately, the above approach is not sufficient when we want to verify that each out of a set of concurrently existing instances of some net exhibits some kind of progress, but the particular instances can reach their local progress states in different states of a state space loop. To allow a specification of this kind of progress, we may introduce instance progress places. A loop in a (full) state space will then be considered legal if and only if it contains an instance progress state for each instance that exists throughout the entire loop. However, when verifying this kind of progress, we have to individually track the particular instances, and the associated computational complexity increases in a similar way as in the case of the later mentioned model checking of ICTL* formulae.

The common principles of using property automata need not be modified in the context of OOPNs. It is only necessary to be able to somehow describe them and to allow them to monitor the appropriate aspects of markings and/or events of the OOPNs being examined. For this reason, we can use the normal modelling elements of OOPNs, eventually combined with some property labels or directives. Nevertheless, due to the very complex structure and behaviour
of OOPNs, it seems to be advantageous to propose some simple extension of the PNtalk language for describing property automata, which holds especially for the case when the automata should be generated automatically from some high-level property specifications.

### 4.5 High-Level Specification Languages and OOPNs

Over state spaces of OOPN-based models, common temporal logics (such as LTL or CTL) can be applied without modifying their temporal operators (and the associated model checking algorithms). There only arises a problem how to specify atoms of formulae of such temporal logics. Nevertheless, atoms of temporal logic formulae refer to properties of particular states or events and thus we can describe them, for example, via the functions for querying states and events which we introduced within the discussion of a state space query language for the context of OOPNs.

Let us, however, note that common temporal logics do not provide sufficient tools for tracing particular instances along state space paths in a renaming-insensitive way. For this reason, we can use the already mentioned, specialized temporal logic called Indexed CTL\(^*\) whose application over symmetrically reduced state spaces of systems consisting of many processes was described e.g. in [3]. Among all, ICTL\(^*\) introduces two new quantifiers, namely \(\wedge_i\) and \(\vee_i\), which act as some kind of universal and existential quantifiers over processes. Note that using ICTL\(^*\) is not at variance with using name abstraction, because ICTL\(^*\) formulae with correct atoms only say that something is supposed to hold for some process or for all processes without any respect to their concrete identification.

When we apply ICTL\(^*\) over OOPN-based models, it is obvious that \(\wedge_i\) and \(\vee_i\) should range over net instances and/or objects. However, the original work on ICTL\(^*\) does not take into account the possibility of dynamically creating and destroying instances. To capture this phenomenon, we should define the process quantifiers in the context of OOPNs to range just over the instances which exist in the state in which we evaluate the appropriate (sub)formula. Moreover, we have to take into account that if we are examining some property of some instance over some state space path, it can happen that the instance is deleted in some moment. Since our intuition is that instances cannot be reincarnated (just only their identifiers can be reused), we should evaluate the given formula for the given instance as though the appropriate instance deletion state was a terminal state. Consequently, if \(EF P_i\) should hold for some instance \(i\) in some marking \(M\), \(P_i\) must get fulfilled on some path leading from \(M\) before \(i\) is destroyed, and if \(AG P_i\) should hold for some instance \(i\) in some marking \(M\), \(P_i\) must hold on all paths leading from \(M\) until \(i\) is destroyed.

As an example of an ICTL\(^*\) formula related to an OOPN, we present the following specification: \(AG \wedge_i (\text{hungry}(i) \Rightarrow AF \text{eating}(i))\). For our system of distributed philosophers from Figures [1] and [2], this formula asserts that if an arbitrary philosopher existing in an arbitrary reachable global state is hungry, then on all paths leading from that state, the philosopher will eventually start eating. However, a verification will tell us that the formula does not hold even under
the strong fairness assumption because of the already mentioned non-progress behaviour of the considered system. Note that the predicates \texttt{hungry} and \texttt{eating} can easily be defined via the functions for querying states of OOPNs.

When model checking ICTL* formulae in the context of OOPNs, we can use the original procedure from [3]. This procedure should only be slightly modified to index states with exactly the instances that are really present in them and to correctly deal with instance deletion states as described above. Compared to a non-indexed temporal logic, the time complexity of model checking ICTL* is increased by the necessity to track the individual life of particular instances [3]. The complexity becomes additionally polynomial (w.r.t. the chosen fairness assumption) in the maximum number of concurrently existing instances, and we have to take into account that some CNA state space labels have to be unfolded.

5 Conclusions

In the paper, we have shown how the various existing ways of asking analysis or verification queries can be applied in the area of models based on the PNtalk OOPNs. Particularly, we have presented some useful OOPN state space statistics, we have described what specific requirements a universal query language over state spaces of OOPNs should meet, we have discussed possible instrumentation of OOPN-based models, and we have also mentioned some issues related to applying temporal logics in connection to OOPNs. We have pointed out that analysis and verification queries to be answered over state spaces of OOPNs can either ignore or respect the structuring of running OOPNs into instances and we have mentioned that dealing with instance-oriented properties usually leads to a higher computational complexity. We have prepared all the ways of asking analysis and verification queries in the context of OOPNs such that we could avoid undesirable references to concrete instance identifiers. The considered specification and query techniques have been applied such that we could use as standard as possible algorithms for performing the actual analysis or verification. However, we have shown that sometimes we have to use special methods to deal with dynamically appearing and disappearing net instances and objects.

In our prototype state space tool, we have made some practical experiments with some of the described methods. Particularly, we have implemented a processor of a simple state space query language over name-abstracted state spaces of OOPNs. This processor can also be used for evaluating some of the mentioned OOPN state space statistics, for dealing with assertions or invariants, or for working with simpler temporal logic formulae if they are manually transformed onto a suitable state space query. A better tool support of the specification and query techniques that we have presented here remains a subject of our further work. It is especially necessary to implement some specialized support for dealing with all the various OOPN state space statistics, for checking properties specified by all the different property labels, and for verifying properties expressed by formulae of some (indexed) temporal logic. The employed algorithms should enable (if possible) both off-line as well as on-the-fly analysis or verification.
In the future the current prototype of the tool for formal analysis and verification over OOPN-based models should be completely reimplemented, made more efficient, and interconnected with the rest of the PNtalk tool suite. Apart from a better support of the here presented specification and query techniques, we should try to further optimize the name abstraction method used in the tool (as sketched in [10]) and to also implement some other state space reduction techniques, including especially some kind of partial order reduction. An interesting possibility how to implement formal analysis and verification over OOPNs is to try to exploit some existing non-OOPN-based formal analysis or verification tool that would allow us (may be after some extension) to cope with the problems associated with state spaces of OOPNs.

Let us further note that working with OOPN-based models of systems is likely to be complemented by exploiting various other modelling languages or notations typically used within object-oriented analysis and design. The different views on modelled systems expressed by means of these languages (object diagrams, communication diagrams, etc.) can be considered an important source of properties of the modelled systems, which could be validated by means of formal analysis or verification over the appropriate OOPNs. Currently we suppose modellers to manually select the key properties and to validate them via the answers which they can obtain to suitable analytical or verification questions specified via some of the ways described above. In the future it would be interesting to examine the possibility of some automated support for such a task.

Acknowledgment. This work is a part of the research project No. CEZ:J22/98: 262200012 - “Research in Information and Control Systems” and it was also supported by the Czech Grant Agency under the contract 102/00/1017 “Modelling, Verifying, and Prototyping Distributed Applications Using Petri Nets”.

References


Integrated Design and Implementation of Digital Controllers

Pedro Albertos and Alfons Crespo

Instituto de Automática e Informática Industrial
Univ. Politécnica de Valencia
{pedro, alfons}@aai.upv.es

Abstract. Similar to the integrated design of industrial control systems, including the process and the controller at once, the integration of the design has to simultaneously consider the controller design and its implementation. The resultant effect in the time constraints of the control requirements should be considered and balanced with the availability of real-time computational resources. The adequate design of the real-time system may reduce these undesirable effects. The analysis of this interaction as well as some techniques to develop a joint design of the control system and its real-time implementation are analyzed. Some examples point out the effectiveness of this methodology.

1 Introduction

The integrated design of industrial control systems, including the process and the controller, is one of the challenges to improve the performances of the controlled system. If a process is not appropriately designed may lead to a challenging control design, and viceversa, a minor modification in the process may result in a great simplification of the controller design. In many cases this is an application-dependent problem that should be analyzed on the basis of a knowledge of the full process.

Another approach of integrated design is to simultaneously consider the controller design and its implementation. Generally these are two sequential operations. First, a controller is designed based on some goals, process and constraints, but these constraints mainly refer to the signals magnitude or the control structure and interconnection. It is assumed that the control law can be precisely implemented, without resources limitations. Then, the control law is physically implemented, trying to match the theoretical design.

Nowadays, almost any controller is digitally implemented and included in a digital subsystem composed of the data acquisition, the communication and the control computing blocks. At the controller design stage, an almost ideal behavior of the digital system implementing the control is assumed. In fig. 1, a basic digital control loop is represented. It is assumed that the control algorithm is executed without any interference with other activities, with a perfectly defined timing.

But, in highly performing controlled systems, the duration of the sampling periods matters and the delays introduced by the digital control due to resources limitations, should be considered.

© Springer-Verlag Berlin Heidelberg 2001
This is mainly the case when the same CPU is used to run a number of control algorithms or the same communication channel is shared for different purposes. In this multitasking environment, the scheduling must consider the effect of these delays on the different control loops. The sampling period cannot be reduced as much as desired due to the time sharing among the different tasks.

The implementation on a unique CPU of multiple control loops, with different sampling periods, and communication among different devices implies the definition of a set of tasks running under timing parameters and constraints, as shown in fig. 2. The task scheduling is a fundamental issue in real-time control algorithm implementation. Over the last two decades a significant work has been made in order to make this effort suitable for real time applications and many real time task models have been proposed to study the scheduling implications of the multitasking environment. The main issue was to reduce the delay in delivering the result of any task.

On the other hand, there have been many proposals and methodologies to consider the unavoidable time delays in the design of the controllers, [1]. These techniques assume that most delays are given independently of the control system or they are due to the execution of the control algorithm. In any case, the time delay is either a data or a constraint and the major interest is to counteract the effect of this delay.

For a given task, its priority is different from the control viewpoint than from the real-time viewpoint. In general, most control loops have the same priority from the control point of view, although larger sampling periods could imply less stringent
timing. But the larger the control effort the more sensitive to delays the control loop is, [3]. On the other hand, from the real-time viewpoint, the priority is in general related to the task period or deadline. Liu and Layland [7] proved that the rate monotonic algorithm is the optimal fixed priority scheduling algorithm.

However there are not too many works in the direction of integrating all activities to get the best control performances and the interaction among these issues, and the treatment of the specific control problems has not been considered until very recently.

Following the idea of the integration, our purpose in this paper is to analyze the techniques to simultaneously design the controller algorithm and its scheduling in a multitasking digital control environment. This design can be done either off-line, that is, before the actual control system is operating, or introducing some sort of adaptation for on-line tuning of the controlled system optimization [4]. This paper is organized as follows. First, a review of the relevant control aspects to show the effects of the sampling period and the time delays over the system behavior is outlined. Then, the real-time scheduling effects on the delays as well as a strategy to reduce them are presented. Finally the integration methodology is analyzed. Some examples are included to illustrate the main results. The synergy between control design and real time task scheduling is summarized in the conclusion section.

2 Delay Effects on the Control

The effect of the delay is not the same in any control task. It is clear that open-loop control system performances are even not very sensitive to time delays. On the other hand, in feedback control systems, the controlled system behavior can be highly degraded if time delays are not included in the design approach. But the main problem is that in controlled systems that seem to be closed-loop well damped and stable may present stability problems if significant time delays happen.

The delay influence in the response can be easily shown in the following example. Let be the process given by the expression:

\[ G(s) = \frac{1.5}{(s+0.5)(s+1.5)} \]

and design a continuous time PID controller. A suitable set of parameters is \( K_p = 8 \), \( T_i = 0.2 \) sec, and \( T_d = 3.2 \) sec. For a discrete time implementation, the controller coefficients can be expressed as:

\[ q_0 = K_p + \frac{K_d}{T} \quad q_1 = -K_p - \frac{2K_d}{T} + K_iT \quad q_2 = \frac{K_d}{T} \]

With sampling periods \( T = 0.04, 0.08, \ldots, 0.24 \) sec, the closed-loop step response is degrading as \( T \) increases, as shown in Figure 3a. Moreover, if a sampling period, \( T = 0.1 \) sec is selected, a delay in the control action delivering of 80% of the period duplicates the step response overshoot, as shown in Figure 3b.
But the delay effect is not the same in any control loop. It depends on the "control effort". This concept illustrates the amount of actuation required to modify the open loop system behavior to achieve the desired closed loop one. In [1], this concept is related to the poles shifting in the real axis, from open to closed loop positions. In a more usual way, the delay effect is related to the system phase margin of stability.

For instance, given the system (A,B), with eigenvalues \{0, -0.5 \pm 1.323j\}, by the state feedback \( K = \begin{bmatrix} 3.1623 & 4.503 & 3.4728 \end{bmatrix} \), the closed loop poles are placed at \{-1.2033 \pm 0.2873i, -2.0662\}.

If instead, the initial system has the poles at the origin (a triple integrator), the feedback gain to place the poles at the same location is \( K = \begin{bmatrix} 3.1623 & 6.5030 & 4.4728 \end{bmatrix} \). A larger control effort is needed.

If there is an unexpected time delay of \( \Delta = 0.35 \) units, the first system remains stable but the second one oscillates. If the delay is known, there are many techniques [1] to take it into account at the controller design stage. Predictors and observers can reduce its effect for undisturbed systems. If it is unknown or time variable, it will impose some constraints in the achievable performances.

**Fig. 3.** a) Negative effect of the sampling period, b) negative effect of the delay
3 Control Implementation Issues

The implementation of the control algorithms consists in the definition of several tasks, one for each control loop, which will be periodically executed. Each control loop and, as consequence, each task, has to be executed within its period and before the deadline. The activities involving a control loop task can be structured in the following parts:

1. Data acquisition: data from external sensor is acquired.
2. Computation of the control action: it corresponds to the regulator computation and should be executed as soon as possible. At the end of this part, the basic control action is ready to be sent to the actuators.
3. Output the control action: must be done either as soon as possible or within a fixed time interval.

When several periodic control tasks are executed, the task activation can suffer delays due to the scheduling policy and the task priorities. Depending on the computation times and periods, different tasks can have higher or lower delays in the starting or finishing instants of each activation, [5]. This situation can add additional delays to the input jitter due to the data acquisition. A four task example will help to show the delays effects. The task set is defined in table 1, and their execution is shown in figure 4.

<table>
<thead>
<tr>
<th></th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>22</td>
<td>70</td>
</tr>
<tr>
<td>T2</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>T3</td>
<td>17</td>
<td>110</td>
</tr>
<tr>
<td>T4</td>
<td>19</td>
<td>110</td>
</tr>
</tbody>
</table>

Table 1. Task set example. Each task has the same deadline than the period and WCET is the worst execution time. The task priorities are assigned using a deadline monotonic (DM) criteria, so task 1 has the highest priority and T4 has the lowest one.

![Fig. 4. Task set chronogram. A pre-emptive priority-based scheduling is used to execute the task set.](image)

In the execution snapshot it can be seen that task T1 always starts at the beginning of the period and finishes after its execution time. However T4, the lowest priority
task, is preempted by the rest of the task and, as consequence, delayed during several clock ticks. These preemptions introduce a delay that is variable at each task activation (period). The fixed and variable delays attached to each task during a hyperperiod are shown in Figure 5.

![Figure 5. Fixed and variable delay of each task during a hyperperiod.](image)

The fixed delay corresponds to the time required to execute the control algorithm. Meanwhile the variable delay is associated to the preemptions suffered by a task. If the control action is delivered just after it is computed a variable delay, the so called Control Action Interval (CAI), is introduced.

In the previous section the effects of the variable delay in the control behavior have been described. To reduce the variable delay of the control activities, the task splitting approach, as described in [6], is implemented. Given a schedulable set of tasks $T$ using the DM theory, [8], each task being selected to minimize the variable variance, is split into three parts: data acquisition, algorithm evaluation and action delivery. Each part of a task will be considered as a separate task. Each new task redefines a new worst case execution time and holds the other task attributes. From task $T_k$ three tasks $T_{ki}$, $T_{km}$, and $T_{kf}$ are defined. The initial task of $T_k$ is defined as:

$$T_{ki} = (C_{ki}, D_{ki}, P_{ki}, F_{ki})$$

(4)

Where $C_{ki}$ is the worst case execution time required by the initial part; $D_{ki}$ is the deadline, $P_{ki}$ is the period, $F_{ki}$ is the offset (initially zero for all parts). In the same way $C_{km}$, $C_{kf}$ are the worst case execution time required by the mandatory and final parts, respectively. All the tasks resulting on the $T_k$ partition hold the same deadline and period ($D_{ki} = D_{km} = D_{kf}$; $P_{ki} = P_{km} = P_{kf}$). The proposed algorithm will modify the offset value of each task.

The priority of each new task is represented as a function of the task. Thus, $\text{Prio}(T_{kj})$ will denote the priority of the final part. The priority assignment is as follows:

- The priorities are grouped into three priority bands.
- The final parts ($T_{kj}$) are allocated in the highest priority band and inside the band, according to the DM policy.
The initial parts ($T_{ki}$) are assigned to the second priority band and also assigned according to the DM.

The mandatory parts are placed in the lowest priority band and also ordered by the DM.

The task decomposition for task $T_i$ in three tasks (initial $T_{ji}$, main $T_{jm}$ and final $T_{jf}$) is detailed in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>WCET</th>
<th>Period</th>
<th>Delays</th>
<th>Original</th>
<th>Task decomposed Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed</td>
<td>Variable</td>
<td>Priority Initial Main Final</td>
</tr>
<tr>
<td>$T_1$</td>
<td>22</td>
<td>70</td>
<td>22</td>
<td>0</td>
<td>1 5 9 1</td>
</tr>
<tr>
<td>$T_2$</td>
<td>15</td>
<td>100</td>
<td>15</td>
<td>23</td>
<td>2 6 10 2</td>
</tr>
<tr>
<td>$T_3$</td>
<td>17</td>
<td>110</td>
<td>17</td>
<td>31</td>
<td>3 7 11 3</td>
</tr>
<tr>
<td>$T_4$</td>
<td>19</td>
<td>110</td>
<td>19</td>
<td>59</td>
<td>4 8 12 4</td>
</tr>
</tbody>
</table>

Table 2. Task decomposition. Each task is decomposed in three task with the priorities shown in the last columns.

This task formulation ensures an almost fixed delay of the control action and, also, a small variable delay range for each task is accomplished. This approach drastically reduces the CAI and allows a better algorithm behavior. The fixed and variable delays obtained for each task in the set are shown in Figure 6.

4 Integrated Design

The off-line integrated design method considers the following steps:

1. Design a set of ideal continuous time controllers and discretized them according to the best available sampling period (or directly design the digital controllers)
2. Schedule all the tasks in the system, according to a DM, with the proposed task splitting
3. Compute the fixed and variable delay of all the control tasks
4. Redesign the controllers, if so required, considering the fixed delays
5. Reassign the priorities to raise that of the tasks involving stronger control actions
6. Go back to step 2, if required.

The on-line integrated design involves the updating of the sampling periods assigned to each control loop, based on the CPU load. If the computing load is reduced, the sampling periods can be shortened to improve the controlled plant performances. The interest of this updating, taking into account the extra tasks required to evaluate the system workload and the controller parameters updating, clearly depends on the benefits obtained by such a change in the periods. Some recent works present results in the so-called flexible schedulers [4], and it is a matter of current research.

5 Conclusions

An evaluation of the delays introduced by the control implementation and their effects on the control behavior has been pointed out in this paper. To reduce their effects, other than redesigning the controller, a task decomposition approach has been proposed. Thus, the variable delay is highly reduced. It allows considering an integrated design of the controllers incorporating the control implementation information in the controller design.

A procedure involving the controller redesign to take into account the fixed and average delays has been proposed. Also, some new ideas about the on-line re-scheduling of a RT control set of tasks has been pointed out, paving the way to integrate the design and implementation of digital controllers.

References

Symbolic Computation Tools for Dynamical Nonlinear Control Systems

Jesús Rodríguez-Millán
Universidad de Los Andes - Facultad de Ingeniería - Dept. Sistemas de Control
Apartado 11 - La Hechicera, Mérida 5101-A, VENEZUELA, Fax: 58-274-2402872
jrmillan@ing.ula.ve

Abstract. In this work we extend our previous two-blocks decomposition approach for the design of nonlinear extension of linear series compensators, to the design of a nonlinear extension of a typical combination of linear series and parallel compensators for nth order nonlinear dynamical control systems. This particular extension shows how the two-block decomposition approach would allow to design nonlinear extensions of virtually any combination of compensators. We illustrate the results by applying them to a model of the centrifugal pendulum in Watt's governor.

Keywords: Nonlinear control, Jacobian linearization, extended linearization, two-block decomposition approach, series-parallel compensation, centrifugal pendulum, Mathematica®.

1 Introduction

The teaching and practice of engineering sciences, and of systems engineering sciences in particular, have been dominated until very recently by two fundamental paradigms: the omnipresent use of linear models to approach the study of actual physical systems, and the numerical calculus as universal computation and simulation tool. Notwithstanding the incontestable success theses two paradigms have had, they are not well fitted to study intrinsically nonlinear phenomena, because, amongst other reasons, linear approximations can only describe local dynamics around hyperbolic equilibrium points, and numerical methods are not reliable enough in the neighbourhoods of bifurcation points. These limitations led engineers to explore different ways and approaches. As a consequence, during last two decades different kinds of nonlinear analytic, algebraic and geometric mathematical techniques have permeated different branches of engineering sciences, simultaneously posing very interesting computation challenges, because many of the new mathematical techniques rely on symbolic calculations. Within this context, symbolic computation is emerging as a promising analysis and design alternative in nonlinear dynamics and control, because it allows implementing complex, tedious, time consuming and error prone symbolic algorithms symbolically and reliably.

Even though Matlab is, and for years to come will surely keep on being, the standard computation tool of the control engineering community, several symbolic computation tools have been developed for the analysis and synthesis of control systems, both linear and nonlinear. Just to mention a few, Birk and Zeitz's group in Stuttgart developed MACNON, first in Macsyma and then in Mathematica, to design controllers by state-feedback [1], Luenberger observers by indirect extended
linearization [2], [3], perform Lyapunov stability studies [4], and nonlinear tracking [5]. Ogunye et al. in USA developed a group of tools to perform observability, controllability and balance realization studies [6], [7], and another for the analysis of time-invariant linear dynamical control systems [8]. Bram de Jager et al. used Maple to develop first ZeroDyn [9], for the study of the zero dynamics in affine nonlinear control systems, and later NonLinCon [10] to solve exact linearization problems. Oñate and Sira [11] also developed a symbolic computation tool to work out the symbolic calculations supporting the exact linearization of affine NLDCS, but using Mathematica as computation platform. The present status of the work of Blankenship et al. on the development of SCT for modeling multibody dynamical systems and designing nonlinear and adaptive control laws based on dynamic inversion methods has been recently summarized in [12]. Forsman and other members of the Swedish school of control have applied Gröbner bases to the construction of Lyapunov functions [13], nonlinear robustness theory [14], the symbolical solution of the algebraic Riccati equation [15], the analysis of polynomial or rational control systems [16], etc.

In previous works we reported the development of a set of symbolic graphic computation tools for the analysis of biparametric second order nonlinear dynamical systems [17], [18], and the design of controllers and observers by Jacobian and extended linearization for nth order nonlinear dynamical control system [19]. More precisely, we proved that nonlinear extensions of arbitrary linear filters admit to be decomposed as a cascade of an extended input PD-like controller, followed by an extended output state-feedback-like controller [20]. This two-blocks decomposition approach reduces the symbolic algorithmic design of nonlinear extensions of arbitrary linear filters to an appropriate assembly of the two subalgorithms for designing extended state-feedback controllers [17] and extended PID controllers [21].

In the present work, we will first consider the application of the two-block decomposition approach to a more general closed-loop compensation topology, namely, the series-parallel compensation scheme. These results show, that the two-block decomposition approach may be applied to arbitrary multi-compensation topologies, and that all the required nonlinearly extended compensators can be designed according to a finite constructive symbolic algorithm. The design algorithms are afterwards applied to the PID-PID series-parallel compensation of the centrifugal pendulum in Watt's regulator. This not only shows how to work out the calculations in a particular case, but also happens to be a simple, yet illustrative, example of a mathematical classification problem. Such problems could hardly be solved using numerical computation methods, but might be naturally formulated and eventually solved, at least in simple cases, within a symbolic computation environment.

2 The Two-Blocks Decomposition Approach

In the sequel we will consider nth order nonlinear dynamical control systems (f, h) (NLDCS (f, h)):

$$\dot{x} = f(x, u), \quad y = h(x), \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}, \quad y \in \mathbb{R}.$$  (1)
where the vector field $f$ and the output function $h$ are supposed to be as smooth as needed. The Jacobian linearization $(A(U), B(U), C(U))$ of the NLDCS $(f, h)$ around the $U$-parametrized operating point $(X(U), U, Y(U))$ will be denoted by

$$
\dot{\xi} = A(U)\xi + B(U)u, \quad \psi = C(U)\xi
$$

where $x = x + X(U)$, $u = u + U$, and $y = y + Y(U)$. A $U$-parametrized linear compensator $(U-LC)$ is a linear filter

$$
G_{lc}(s) = \frac{U(s)}{E(s)} = \frac{K_{c0}(U)s^l + K_{c1}(U)s^{l-1} + \ldots + K_{cl-1}(U)s + K_{cl}(U)}{s^m + a_{c1}(U)s^{m-1} + \ldots + a_{cm-1}(U)s + a_{cm}(U)}.
$$

Alternatively, the U-LC above may be described through its two-block decomposition

$$
\begin{align*}
\dot{z}_i &= e \\
\dot{e} &= K_{cl}(U)z_i + K_{cl-1}(U)e + \ldots + K_{c1}(U)e^{(l-1)} + K_{c0}(U)e^{(l)} \\
\dot{z}_o &= -\text{CompanionMatrix}[a_{cm}(U) a_{cm-1}(U) \ldots a_{c1}(U)]z_o + (0 \ldots 0 \hat{e})^T \\
u &= z_{o1}
\end{align*}
$$

where the first two equations describe the generalized input PD-like controller, and the second two the output state-feedback-like controller [20].

Nonlinear extensions of a particular linear device may be manifold. In next theorem, we propose a natural nonlinear extension of the U-LC above, based on a sufficient condition assuring the commutativity of the linearization and compensation operations [20].

**Theorem 1.** The nonlinear dynamical control system

$$
\begin{align*}
\dot{z}_i &= e \\
\dot{e} &= k_{cl}(z_i)z_i + k_{cl-1}(z_i)e + \ldots + k_{c1}(z_i)e^{(l-1)} + k_{c0}(z_i)e^{(l)} \\
\dot{z}_o &= -a(z_o) + (0 0 \ldots 0 \hat{e})^T \\
u &= z_{o1}
\end{align*}
$$

where functions $k_{cj}$, $0 \leq j \leq l$, and $a$ are the solutions of the following systems of PDE

$$
\begin{align*}
k_{cj}(z_i) &= [K_{cj}(U)]_{U=z_i} \quad (6-a)
\end{align*}
$$
\[
[D_{z_0^a}] (U; 0; \ldots; 0) = \begin{pmatrix}
0 & 1 & \ldots & 0 \\
0 & 0 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \ddots & 1 \\
a_{cm}(U) & a_{cm-1}(U) & \ldots & a_{c0}(U)
\end{pmatrix} \quad a(U, 0, \ldots, 0) = 0 \quad (6-b)
\]

qualifies as a nonlinear extension of the U-parametrized linear compensator \( G_c \).

**Proof.** For the proof, see [20].

It is worth to remark that the nonlinear extension proposed above satisfies the three requirements an ideal nonlinear extension should fulfil, namely: i) to preserve the internal topological structure of the linear system it is extending, ii) to introduce nonlinearities depending on the compensator’s state-vector, and iii) to implement the extension through physically meaningful parameters or characteristics.

### 3 Series-Parallel Compensation

In this section, we search for a constructive symbolic algorithm leading to calculate nonlinear extensions of the series-parallel compensation scheme shown in Fig. 1.

\[\begin{align*}
G_{sc}(s) &= \frac{U_{sc}(s)}{E(s)} = \frac{K_{s0}(U)s^k + K_{s1}(U)s^{k-1} + \ldots + K_{sk-1}(U)s + K_{sk}(U)}{a_{s0} + a_{s1}(U)s + \ldots + a_{sm-1}(U)s + a_{sm}(U)}, \quad (7-a) \\
G_{pc}(s) &= \frac{U_{pc}(s)}{Y(s)} = \frac{K_{p0}(U)s^j + K_{p1}(U)s^{j-1} + \ldots + K_{pj-1}(U)s + K_{pj}(U)}{a_{p0} + a_{p1}(U)s + \ldots + a_{pl-1}(U)s + a_{pl}(U)}, \quad (7-b)
\end{align*}\]

where \( k \leq m \), and \( j \leq l \), respectively. Alternatively, these compensators may also be described by their two-block decompositions:
Symbolic Computation Tools for Dynamical Nonlinear Control Systems

\[ z_{si} = e \]
\[ \dot{z}_{si} = K_s k(U)z_{si} + \sum_{i=0}^{k-1} K_s i(U)e(k-i) \]
\[ \dot{z}_{so} = -\text{CompanionMatrix}[a_s U \ldots a_m U]z_{so} + (0 \ldots 0 \dot{e}_s)^T, z_{so} \in \mathbb{R}^m \]
\[ u_s = z_{so1} \]
\[ z_{pi} = y \]
\[ \dot{z}_{pi} = K_p j(U)z_{pi} + \sum_{i=0}^{j-1} K_p i(U)e(j-i) \]
\[ \dot{z}_{po} = -\text{CompanionMatrix}[a_p U \ldots a_l U]z_{po} + (0 \ldots 0 \dot{e}_p)^T, z_{po} \in \mathbb{R}^l \]
\[ u_p = z_{po1} \]

where \( z_s \) and \( z_p \) denote the state-vector variables of the series and parallel compensators, respectively. The steady-state values of the state-variables in the two systems above are: \( u_s(\infty) = 0; z_s(\infty) = (0; \ldots; 0) \)
\( u_p(\infty) = U; z_p(\infty) = (U, 0, \ldots, 0) \).

**Theorem 2.** The nonlinear dynamical control systems

\[ z_{si} = e \]
\[ \dot{z}_{si} = k_s k(U)z_{si} + k_s k-1(U)e(k-1) + \ldots + k_s 1(U)e(k) \]
\[ \dot{z}_{so} = -a_s(z_{so}) + (0 \ldots 0 \dot{e}_s)^T, z_{so} \in \mathbb{R}^m \]
\[ u_s = z_{so1} \]
\[ z_{pi} = y \]
\[ \dot{z}_{pi} = k_p j(U)z_{pi} + k_p j-1(U)e(j-1) + \ldots + k_p 1(U)e(j) \]
\[ \dot{z}_{po} = -a_p(z_{po}) + (0 \ldots 0 \dot{e}_p)^T, z_{po} \in \mathbb{R}^l \]
\[ u_p = z_{po1} \]

where the nonlinear functions \( a_s; k_s 0; \ldots; k_s k \) and \( a_p; k_p 0; \ldots; k_p j \) are the solutions of the following two sets of PDE:

\[ k_s q(z_{si}) = [K_s q(U)]_{U=z_{si}}, \quad 1 \leq q \leq k \]
\[ [Dz_{so} a_s(U,0,\ldots,0)] = -\text{CompanionMatrix}[a_s(U) \ldots a_m(U)], a_s(0,\ldots,0) = 0 \]
(10)
\[ k_p q(z_{pi}) = [K_p q(U)]_{U=z_{pi}}, \quad 1 \leq q \leq j \]
\[ [Dz_{po} a_p(U,0,\ldots,0)] = -\text{CompanionMatrix}[a_p(U) \ldots a_l(U)], a_p(U,\ldots,0) = 0 \]
qualify as nonlinear extensions of the series and parallel compensators \( G_{sc} \) and \( G_{pc} \), respectively.

**Proof.** The proof goes along the same lines than the proof of Theorem 1 in [20].

To see how the calculations should be worked out in an actual case, we consider in next section the centrifugal pendulum of Watt's regulator [22] as a case study. The control goal will be to asymptotically stabilize the centrifugal pendulum at a deviation angle \( X \) with respect to its vertical rotation axis, according to a prescribed asymptotic convergence pattern.

### 4 A Case Study: The Centrifugal Pendulum in Watt’s Governor

The heart of Watt's flywheel governor is a centrifugal pendulum (CP) that may be modelled [22] by the following system of equations

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -\left[\frac{2}{\omega_0^2} + u^2 \cos(x_1)\right] \sin(x_1) - \rho x_2 \\
y &= x_1
\end{align*}
\]  

(11)

where \( x_1 \) represents the angular deviation of the pendulum with respect to its vertical rotation axis, the control signal \( u \) is the angular speed of the pendulum around its vertical axis, \( \omega_0^2 = g/l \), where \( l \) is the length of the weightless rigid bar of the pendulum, and \( \rho \) is the friction coefficient at the pivot. In the sequel we will refer ourselves to the operating point as \( X \), for \( X \) denoting the desired angular deviation with respect to the vertical axis at which the pendulum should be driven by the control law to be designed.

**Operating Point:** \((x_1(\infty); x_2(\infty); u(\infty); y(\infty)) = (X; 0; \frac{\omega_0}{\sqrt{\cos(X)}}; X)\).

**Local Transfer-Function of the Centrifugal Pendulum Around the Operating Point:** After linearizing the CP around the operating point to get \( A(X) \), \( B(X) \), \( C(X) \) and working out \( G_p(s) = C(X)[sI - A(X)]^{-1}C(X) \), we get the following expression for the local transfer-function of the pendulum around \( X \):

\[
G_p(s) = \frac{2\omega_0\sqrt{\cos[X]}\sin[X]}{s^2 + \rho s + \omega_0^2\sin[X]\tan[X]}.
\]  

(12)

**PID Series and Parallel Controllers:** We will assume the series and parallel compensators are respectively characterized by the transfer-functions
\[ G_{sc}(s) = K_{ps}(X) + K_{ds}(X)s + \frac{K_{is}(X)}{s} \]
\[ G_{pc}(s) = K_{pp}(X) + K_{dp}(X)s + \frac{K_{ip}(X)}{s}. \]

Transfer-Function of the Closed-Loop System: Standard block-diagram algebra leads to the following expression for the transfer-function of the closed-loop system:

\[
H(s) = \frac{G_{p}(s)G_{sc}(s)}{1 + G_{p}(s) + G_{p}[G_{sc}(s) + G_{pc}(s)]}
\]
\[
= \frac{2\omega_0\sqrt{\cos[X]\sin[X]} (K_{is}(X) + K_{ps}(X)s + K_{ds}(X)s^2)}{s^3 + \alpha_1(X)s^2 + \alpha_2(X)s + \alpha_3(X)}
\]

where the coefficients of the denominator’s polynomial are given by:

\[
\alpha_1(X) = \rho + 2\omega_0\sqrt{\cos[x]\sin[x]}(K_{ds}(X) + K_{dp}(X))
\]
\[
\alpha_2(X) = \omega_0^2\sin[X]\tan[X] + 2\omega_0\sqrt{\cos[X]\sin[X]}(K_{ps}(X) + K_{pp}(X))
\]
\[
\alpha_3(X) = 2\omega_0\sqrt{\cos[x]\sin[x]}(K_{is}(X) + K_{ip}(X))
\]

Characteristic Polynomial of the Closed-Loop System

\[
p(s) = \text{Denominator}[H(s)] = s^3 + \alpha_1(X)s^2 + \alpha_2(X)s + \alpha_3(X)
\]

Necessary Condition for PID-PID Series-Parallel Asymptotic Stabilizability: From the structure of the characteristic polynomial, it transpires that not every PID-PID series-parallel compensation scheme is apt to asymptotically stabilize the CP. More, precisely, we have:

**Theorem 3.** \( K_{ps}(X) + K_{pp}(X) \neq 0 \) & \( K_{is}(X) + K_{ip}(X) \neq 0 \) & \( K_{ds}(X) + K_{dp}(X) \neq 0 \) is a necessary condition for the PID series-parallel compensation scheme to be able to asymptotically stabilize the centrifugal pendulum.

**Proof.** Suppose \( K_{ps}(X) + K_{pp}(X) = 0 \) or \( K_{is}(X) + K_{ip}(X) = 0 \) or \( K_{ds}(X) + K_{dp}(X) = 0 \). Then, at least one of coefficients of the characteristic polynomial of the closed-loop system cannot be modified at will through the PID gains, wherefore the eigenvalues of the closed-loop system cannot be arbitrarily relocated on the complex plane.

Then, if not all configurations fit, what kind of PID-PID series-parallel compensation schemes actually allow to stabilize the CP at will? Corollary 4 answers this question.

**Corollary 4.** The centrifugal pendulum may be asymptotically stabilized at will using either a single series PID compensator, a single PID parallel compensator, or any
series-parallel compensation scheme whose PID gains satisfy the relations prescribed in Theorem 3. In particular, compensation schemes lacking a proportional, an integral, or a derivative component are not admissible.

**Design of a PID-PID Series-Parallel Compensation Scheme by Pole Assignment.**

The following Theorem provides an explicit symbolic recipe to design PID-PID series-parallel compensation schemes, allowing to fulfill any prescribed pattern of asymptotic convergence of the CP to the desired operating point. It is worth to remark that the design problem has not a unique but infinitely many solutions. Moreover, the choice of the particular gains for the series PID controller depends on the choice of the corresponding gains for the parallel controller, and viceversa.

**Theorem 5.** Suppose \( p_d(s) = (s + s_1)(s + s_2)(s + s_3)(s + s_4) = s^3 + b_1s^2 + b_2s + b_3 \) is the desired characteristic polynomial of the PID series-parallel compensated centrifugal pendulum. If the PID gains are chosen to be

\[
K_{pp}(X) = - K_{ps}(X) + \frac{b_2 \text{Csc}[X]}{2\omega_o \text{Cos}[X]} - \frac{\omega_o \text{Sin}[X]}{2\text{Cos}[X]^{3/2}} \\
K_{ip}(X) = - K_{is}(X) + \frac{b_3 \text{Csc}[X]}{2\omega_o \text{Cos}[X]} \\
K_{dp}(X) = - K_{ds}(X) + \frac{(b_1 - \rho) \text{Csc}[X]}{2\omega_o \text{Cos}[X]}
\]

then, the closed-loop system converges to the desired operating point asymptotically, according to the pattern prescribed by the desired characteristic polynomial \( p_d(s) \).

**Proof.** Suppose the PID gains are chosen as prescribed in (17-a) to(17-b). Then, \( p(s) = s^3 + b_1s^2 + b_2s + b_3 \). In fact, the prescribed values for the gains are the solutions of the system of equations:

\[
b_1(X) = \rho + 2\omega_o \sqrt{\text{Cos}[X] \text{Sin}[X]}(K_{ds}(X) + K_{dp}(X)) \\
b_2(X) = \omega_o^2 \text{Sin}[X] \text{Tan}[X] + 2\omega_o \sqrt{\text{Cos}[X] \text{Sin}[X]}(K_{ps}(X) + K_{pp}(X)) \\
b_3(X) = 2\omega_o \sqrt{\text{Cos}[X] \text{Sin}[X]}(K_{is}(X) + K_{ip}(X)).
\]

which comes out after equating the actual and the desired characteristic closed-loop polynomials coefficientwisely.

**Nonlinear Extension of the PID-PID Series-Parallel Compensation Scheme.** To calculate a nonlinear extension we must first choose the particular linear compensation scheme to be extended. To illustrate the nonlinear extension procedure, let us suppose
Fig. 2. Self-adjusting of the PID gains induced by a change in the operating point X. Given that $u_s(\infty) = 0$, whatever the operating point, the series control signal only modifies the transient response, whereas the steady-state value of the parallel control signal actually fixes the operating position of the CP.

Then, after the recipe in Theorem 2, the nonlinear gains are:

\begin{align}
K_{ps} \text{ is constant. Then, } K_{pp} &= K_{ps}(X) \\
K_{is} \text{ is constant. Then, } K_{ip} &= K_{ps}(X) \\
K_{dp} &= 0. \text{ Then, } K_{ds} = K_{ds}(X).
\end{align}

Then, after the recipe in Theorem 2, the nonlinear gains are:

\begin{align}
k_{ps}(e) &= K_{ps}, \quad k_{is}(e) = K_{is}, \quad k_{ds}(e) = \frac{(b_1 - \rho) \text{Csc}[e]}{2 \omega_o \sqrt{\text{Cos}[e]}}, \\
k_{pp}(y) &= -K_{ps}(y) + \frac{b_2 \text{Csc}[y]}{2 \omega_o \sqrt{\text{Cos}[y]}}, \quad \frac{\omega_o \text{Sin}[y]}{2 \text{Cos}[y]^{3/2}} \\
k_{ip}(y) &= -K_{is}(y) + \frac{b_3 \text{Csc}[y]}{2 \omega_o \sqrt{\text{Cos}[y]}}, \quad k_{dp}(y) = 0.
\end{align}
As the set of formulae (17) show, the gains of the linear series and parallel PID controllers are X-parametrized, wherefore they have self-tuning capacities regarding modifications of the operating point. This X-parametrization of the linear PID gains induces a dependence of the nonlinear PID gains respect to e and y, the state-variable of the series and parallel PID controller, respectively, as indicated by the set equations (20). Fig. 2 shows the behavior of the nonlinearly compensated CP when at t = 90 sec the desired operating point suddenly jumps from $X = \pi/12$ to $X = X = \pi/6$.

5 Discussion

In this paper we have shown that the two-blocks decomposition approach, originally developed in [20] for arbitrary series compensators, admits to be straightforwardly applied to series-parallel compensations schemes also. In fact, this extension suggests the two-block decomposition approach might be systematically applied to calculate nonlinear extensions of arbitrary multi-compensators schemes. Yet, some important points must be kept in mind to go through the design procedure smoothly: i) in each case the nonlinear gain functions must depend on the corresponding state-vector variable. So, the internal state-vector variables of every single input and output block must be clearly defined. ii) The state-vector variables of each component of the compensation scheme ought to be functionally distinguished from the input and output signals of the same component, even if they could correspond to the same physical variable. iii) Special attention has to be paid to establishing what the steady-state values of the state-variables of each compensator are. This may not be trivial in some cases.

To illustrate the proposed methodology we applied it to a PID-PID series-parallel compensation scheme, just because in this case the compensators have no output state-feedback like controllers. Had they, we should have had to calculate the nonlinear extensions of the output state-feedback like controllers. Depending on the compensator, this would have required much more calculation effort. Yet, this calculations may be automated using our previously developed symbolic computation tool NLControl [19].

The design of nonlinear extensions of arbitrary linear series-parallel compensation schemes may ultimately be reduced to an appropriate assembly of the two already known algorithms for nonlinear extended PID controllers in [21], and for state-feedback controllers in [17]. Therefore, the actual limitation to synthesize a particular nonlinear extension is the present capacity of Mathematica to solve the integrals in the state-feedback algorithm.

The centrifugal pendulum above is a typical academic example, frequently used in classrooms to illustrate the application and compare the performance of different control techniques. It is also possible, of course, to numerically check different control techniques on the CP through computer simulations. Yet, numerical analysis is not the natural computation tool when we try to get answers to qualitative or structural questions, like the classification of the stabilizing PID-PID series-parallel compensation schemes considered above. Mathematical classification problems are good examples of problems that might be successfully solved using symbolic computation techniques, at least in simple cases. Thus, further spreading of symbolic computation at the Engineering Sciences Faculties in the years to come would
definitely contribute a lot to develop a qualitative way of thinking in engineering students.

Symbolic computation might support the automation of many a symbolic mathematical technique in nonlinear dynamics, as an important step forward to bridge the existing gap between the qualitative and quantitative approaches in control and other fields of engineering sciences: circuits, mechanics, selfoscillations, etc. Yet, symbolic computation involves a lot of non-standard algebra and analysis, which will hopefully become standard engineering mathematics in the near future.

It seems reasonable to expect that the synergetic coupling of the new symbolic computation techniques with the traditional numerical and graphical computation techniques should lead to a higher leaf on the spiral of understanding and realization in the engineering sciences. Given that the most reliable way to design the future is to modify education today, engineering schools, and very in particular systems engineering schools, should modify their structures and programs not only to assimilate this new reality but to speed up its evolution. An engineering practice uniting the present numerical simulation capacity with the qualitative comprehension symbolic graphic computation could provide would certainly be more robust, safe, green and reliable than presently existing engineering.

References


Acknowledgments

This work was developed during the sabbatical leave of the author at the Research Institute for Symbolic Computation (RISC) of the Johannes Kepler University Linz, Austria. It is with deep gratitude that he acknowledges the invitation of Prof. Bruno Buchberger to RISC. The financial support of the Venezuelan National Council for Science and Technology (CONICIT), and the University of Los Andes made this sabbatical leave possible.
Integrated Symbolic-Graphic-Numeric Analysis and Design in Nonlinear Control through Notebooks in Mathematica

Jesús Rodríguez-Millán

Universidad de Los Andes - Facultad de Ingeniería - Dept. Sistemas de Control
Apartado 11 - La Hechicera, Mérida 5101-A, VENEZUELA, Fax: 58-274-2402872
jrmillan@ing.ula.ve

Abstract. In the present work we will present two sets of notebooks developed in Mathematica®: a first one to analyze and classify local dynamics of second-order parametrized nonlinear dynamical systems, and the second one to synthesize linear and extended nonlinear state-feedback controllers, PID controllers, and Luenberger observers for second-order nonlinear control systems.

Keywords. Analysis, Design, Nonlinear control systems, Jacobian and extended linearization, integrated symbolic-graphic-numeric computation, notebooks, Mathematica®.

1 Introduction

On performing the analysis, design and simulation of nonlinear dynamical control systems control engineers usually resort to several specialized computation tools. A typical extended or exact linearization design, for instance, involves: firstly, calculating a symbolic nonlinear control law by hand or partially supported by an algebra computation package like Maple® or Mathematica®, secondly, building up a block diagram using Simulink® and, thirdly, simulating open and/or closed-loop performances in Matlab®. Thus, to complete a design exercise control engineers must not only master the required theoretical control techniques and several specific purpose computation tools. They also have either to design appropriate interfaces to transfer data between different control packages, or to act themselves as human interfaces by transferring data manually, what is a very untrustworthy practice.

Matching different control packages through software interfaces may, in principle, do the job efficiently and reliably, but it is too rigid, difficult and costly for nonlinear control. On the other hand, manual interfacing may seem cheaper, at least for low order systems, but it is a very unreliable practice, specially within the nonlinear control context where, whatever the order of the system, control and observation laws can be page long symbolic expressions. So, neither of these approaches seem to be the natural one.

To improve reliability and simplify the solution of both analysis and design problems, it would be highly desirable to count on a single integrated computation tool, capable of performing long, complex, error prone, programmed sequences of symbolic, graphic and numeric calculations, fast and confidently. Moreover, such a computation tool should allow the user to modify the parameters, the components, or the structure of the system under study, to perform comparative performance experiments efficiently.

© Springer-Verlag Berlin Heidelberg 2001
In the present paper we describe a first attempt to develop such an integrated computation tool, mainly for teaching applications by the time being, using the notebook facilities provided by Mathematica®. Mathematica®’s notebooks are interactive documents, integrating the traditional features of text and graphics processors with the symbolic, graphic and numeric calculation capabilities of Mathematica®. Thus, notebooks provided a nearly ideal single developing environment because they provide the required symbolic, graphic and numeric calculation facilities, eliminate the necessity of interfacing different computation packages, may be easily modified and run like they were computer programs, and allow to document the whole work by incorporating texts, draws, and even graphic animation and acoustic backgrounds.

More precisely, we will present two sets of notebooks: the first one allowing to analyze and classify local dynamics of second-order parametrized nonlinear dynamical systems, and the second one to synthesize linear and extended nonlinear state-feedback controllers, PID controllers, and Luenberger observers for second-order nonlinear control systems.

2 Integrated Analysis of Nonlinear Dynamical Systems

The local analysis of a triparametric second-order single-input single-output nonlinear dynamical control system (NLDSCS):

\[
\begin{aligned}
\dot{x} &= f(x; u; \eta) \\
x(0) &= x_0 \\
\eta &\in \mathbb{R}^3 \\
y &= h(x, u, \eta),
\end{aligned}
\]

where \( f \) and \( h \) are smooth enough, is primarily based upon calculating: (i) the equilibrium or operating points of the systems, i.e.:

\[
(X(U); U) \in \mathbb{R}^n \leftarrow \text{such that } f(X(U), U) = 0,
\]

(ii) their associated linearizations:

\[
\begin{aligned}
\dot{x} &= A(U; \eta)(x - X(U)) + B(U; \eta)(u - U) \\
y &= C(U, \eta)(x - X(U)) + D(U, \eta)(u - U)
\end{aligned}
\]

where

\[
\begin{aligned}
A(U; \eta) &= D_x f(X(U); U; \eta) \\
B(U; \eta) &= D_u f(X(U), U, \eta) \\
C(U, \eta) &= D_x h(X(U), U, \eta), \\
D(U, \eta) &= D_u h(X(U), U, \eta)
\end{aligned}
\]

and (iii) the controllability and observability matrices:

\[
[B(U, \eta); A(U, \eta)B(U, \eta)] \text{ and } [C(U, \eta); C(U, \eta)A(U, \eta)]
\]

respectively.
On the linearizations basis, we profit of Mathematica® plotting capacities to construct topological classification maps for the equilibria, to provide an exhaustive description of local dynamics on the parameters space. Topological classification maps are obtained by plotting the zero and infinity level hypersurfaces of the trace, the determinant, and the discriminant of the system matrix $A(U, \eta)$ on the parameters-space. This set of hypersurfaces compartmentalizes the parameters-space into disjoints subsets corresponding to linearly non-equivalent local dynamics [1]. Among other applications, topological classification maps maybe used to support Lyapunov stability and robustness studies, design robust controllers, or detect self-oscillatory patterns of behaviour.

2.1 The Structure of the Integrated Analysis Notebook

The basic notebook allowing to perform integrated symbolic-graphic-numeric analysis of $\eta$-parametric NLDCS like (1) has the following standard structure:

1. **Introduction.** Consist of a group of text cells where generalities and the goals of the system analysis problem are described.
2. **Modelling the System.** Typically consists of a group of text cells covering the mathematical modelling of the systems to be analyzed. It may also contain input and output cells used to perform calculations supporting the modelling process.
3. **Declaring the System.** In this section, we use input cells to declare the parametrized NLDCS (1).
4. **Equilibrium and Operating Points.** In this section we input the vector field $f(x, u, \eta)$ and solve the equations $f(x, u, \eta) = 0$ or $f(x, U, \eta) = 0$ to calculate the equilibrium points. Results appear in output cells as a list of the equilibrium or operating points of the system.
5. **Linearizing the System.** In this section, matrices $A$, $B$, $C$ and $D$ in (4) are calculated and the linearized system (3) is constructed.
6. **Classification of Local Dynamics.** After Hartman-Grobman theorem [2], the local dynamics of the NLDCS (1) around its equilibrium point $(X(U), U)$ is generically determined by the triplet of signs $(\text{Sgn}[\text{Determinant}[A(U, \eta)]], \text{Trace}[A(U, \eta)], \text{Discriminant}[A(U, \eta)])$, for $A(U, \eta)$ in (4). To determine the triplet of signs we calculate the zero and infinity level hypersurfaces of Determinant$[A(U, \eta)]$, Trace$[A(U, \eta)]$, and Discriminant$[A(U, \eta)]$, and plot them on the parameters space. This plotting compartmentalizes the parameters space into disjoint subsets, each one of which corresponds to a local dynamics. The topological classification criterion for the local dynamics is summarized in Table 1.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>Degenerate equilibrium point</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Saddle point</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>+</td>
<td>Stable focus (spiral)</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Stable Node</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Unstable focus (spiral)</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Unstable Node</td>
</tr>
</tbody>
</table>

Table 1. Topological classification of the equilibrium point $(X(U), U)$. 
7. Additional Studies. Constructing a topological classification map usually only represents the starting point for further studies. In next section, we use them to detect the outburst of selfoscillations in a tunnel diode circuit. Yet, there are plenty of other applications: robust controller design, Lyapunov stability analysis, etc. Frequency domain analysis techniques, based upon transfer-functions, Bode plots, etc., can also be easily incorporated [3].

2.2 Case Study 1. A Tunnel Diode Nonlinear Oscillator

In this section, we present a sketch of an actual notebook in Mathematica with the basic structure described in previous section.

1. Introduction. We consider as a case-study the tunnel diode nonlinear circuit of Fig. 1, a system simple enough for illustrative purposes, yet interesting.

![Fig. 1. A second order tunnel diode nonlinear circuit. Parameters r, L, C are positive, and the nonlinear component is modelled by \( g(v) = -a v + b v^3 \), \( a > 0 \).](image)

2. Modelling the System. After applying Kirchhoff’s current law, choosing the state-variables as \( v = \sqrt{b} x_1 \), \( i = \sqrt{b} x_2 \), and rescaling time according to \( t = C s \), we obtain the following mathematical model for the tunnel diode circuit:

\[
\begin{align*}
\dot{x}_1 &= ax_1 - x_2 - x_1^3, \\
\dot{x}_2 &= d(x_1 - r x_2), \quad d = C/L.
\end{align*}
\]

(6)

where \( r > 0 \), \( C > 0 \), \( L > 0 \), \( a > 0 \).

3. Declaring Equations

\( \text{Clear}[f1, f2, f3, f4, fd1, fd2, x1, x2, X1, X2, A, AX, TrA, DETA, p, DiscA, a, d, r] \)

\( \text{f1}[x1_, x2_] := a x1 - x2 - x1^3 \)

\( \text{f2}[x1_, x2_] := d (x1 - r x2) \)

4. Equilibrium Points

\( \text{EquilPoint} = \{\text{f1}[x1, x2], \text{f2}[x1, x2]\} \)

\( \text{EquilPoint} = \text{Solve}[\text{EquilPoint} == 0, \{x1, x2\}] \)

\( \{\{x2 -> 0, x1 -> 0\}, \{x2 -> \frac{-\sqrt{r a - 1}}{r^{3/2}}, x1 -> \frac{-\sqrt{r}}{r}\}, \{x2 -> \frac{\sqrt{r a - 1}}{r^{3/2}}, x1 -> \frac{\sqrt{r}}{r}\}\} \)
5. Linearization

In[]: \[ A[x1_, x2_] := \text{Simplify}\{\{D[f1[x1, x2], x1], D[f1[x1, x2], x2]\}, \{D[f2[x1, x2], x1], D[f2[x1, x2], x2]\}\} \]

In[]: \[ \text{MatrixForm}[A[x1, x2]] \]

Out[]: \[
\begin{pmatrix}
-3 x1^2 + a & -1 \\
d & -rd
\end{pmatrix}
\]

Choosing the Equilibrium Point

In[]: \[ \text{ActOpPoint} = \text{EquilPoint}[[1]] \]

In[]: \[ X1 = x1 /. \text{ActOpPoint} \]

In[]: \[ X2 = x2 /. \text{ActOpPoint} \]

Out[]: \{x2 \to 0, x1 \to 0\}

Linearization Around the Equilibrium Point

In[]: \[ AX = A[x1, x2] /. \{x1 \to X1, x2 \to X2\}; \]

In[]: \[ \text{Simplify[MatrixForm[A[x1, x2]]]} /. \{x1 \to X1, x2 \to X2\} \]

Out[]: \[
\begin{pmatrix}
a & -1 \\
d & -rd
\end{pmatrix}
\]

Obtaining the Zero Level Hypersurface of Tr[AX], Det[AX] and Disc[AX]

In[]: \[ \text{TrO} = \text{Solve}[\text{Tr}[AX] == 0, a] \]

In[]: \[ \text{DO} = \text{Solve}[\text{Det}[AX] == 0, a] \]

In[]: \[ \text{DiO} = \text{Solve}[\text{Dis}[AX] == 0, a] \]

Out[]: \{\{a \to r d\}\}

Out[]: \{\{a \to \frac{1}{r}\}\}

Out[]: \{\{a \to -2 \sqrt{d - rd}\}, \{a \to 2 \sqrt{d - rd}\}\}

Plotting the Zero Level Hypersurface of Tr[AX], Det[AX] and Disc[AX]

In[]: \[ g1 = \text{Plot3D}[\{a /. \text{TrO[[1]]}, \text{Hue}[0.7, 0.9, 0.9]\}, \{d, 0, 2\}, \{r, 0.01, 2\}, \text{AxesLabel} \to \{"d", "r", "a"\}] ; \]

In[]: \[ g2 = \text{Plot3D}[\{a /. \text{DO[[1]]}, \text{Hue}[0.7, 1, 0.8]\}, \{d, 0, 2\}, \{r, 0.01, 2\}, \text{AxesLabel} \to \{"d", "r", "a"\}] ; \]

In[]: \[ g3 = \text{Plot3D}[\{a /. \text{DiO[[1]]}, \text{Hue}[0.4, 1, 0.8]\}, \{d, 0, 2\}, \{r, 0.01, 2\}, \text{AxesLabel} \to \{"d", "r", "a"\}] ; \]

In[]: \[ g4 = \text{Plot3D}[\{a /. \text{DiO[[2]]}, \text{Hue}[0.4, 1, 0.8]\}, \{d, 0, 2\}, \{r, 0.01, 2\}, \text{AxesLabel} \to \{"d", "r", "a"\}] ; \]

Out[]: The actual graphics of g1, g2, and g3, g4 are shown on Fig. 2-a, 2-b, and 2-c, respectively.

Constructing the Topological Classification Map

In[]: \[ g5 = \text{Show}[g1, g2, g3, g4] \]

Out[]: The actual graphics of g5 is shown on Fig. 2-d.
2.3 Does the Tunnel-Diode Circuit Self-oscillate?

We may roughly reformulate Poincaré-Andronov-Hopf (PAH) Theorem [4], [5] as a self-oscillations detection criterion as follows: a second order nonlinear dynamical system self-oscillates provided it has a spiral equilibrium point whose stability is transversally reversed when a bifurcation parameter crosses a PAH-bifurcation point. It is worth to remark, in any case, that self-oscillating could either means that on crossing a PAH-bifurcation point a local family of attracting nontrivial periodic orbits emerges (subcritical PAH-bifurcation) or a local family of repelling nontrivial periodic orbits disappear (supercritical). In this paper, we are only considering the detection of PAH-bifurcation.

![Fig. 2-a. Zero-level hypersurface of Trace[AX].](image)

Trace[AX] > 0 above the hypersurface. Thus, systems with an asymptotically stable origin are below the hypersurface.

![Fig. 2-b. Zero-level hypersurface of Det[AX].](image)

Det[AX] < 0 above the hypersurface. So, all the (d, r, a) associated systems have a saddle-point at the origin.

![Fig. 2-c. Zero-level hypersurface of Discriminant[AX].](image)

Disc[AX] < 0 between the two layers of the hypersurface, wherefore all associated systems have an spiral point at the origin.

![Fig. 2-d. Superposition of the zero-level hypersurfaces of Trace[AX], Det[AX], Disc[AX].](image)

This map is the starting point to construct a topological classification map for the origin.

Fig. 2. Constructing the Topological Classification Map for the Origin.
In order to apply PAH-Theorem as a nonlinear oscillations detection tool three conditions must then be checked to hold: (i) The existence of parameters dependent spiral equilibrium points. (ii) The existence of a set of PAH-bifurcation points, i.e., a (PAH) hypersurface on the parameters space separating attracting from repelling spiral equilibria, and (iii) The existence of transversal crossings of the PAH-hypersurface induced by changes in the PAH-bifurcation parameters. These conditions are shown to hold on Fig. 3. Fig. 4 illustrates the arising nontrivial periodic orbits associated to parameter a-controlled PAH-bifurcation.

2.4 Discussion

Every horizontal line (d, R, A) or (D, r, A), for A, D, R arbitrary but fixed, intersects the zero-level hypersurface of Trace[AX] transversally, wherefore d-controlled and r-controlled PAH bifurcations also exist. From a practical point of view these PAH bifurcations look more natural, in the sense that they would allow to control both the amplitude and the frequency of the oscillations by adjusting the values of the passive elements r, L and C in the tunnel diode circuit in Fig. 1. In fact, a-controlled bifurcations model the dependence of self-oscillations on the properties of the tunnel diode itself. Yet, physical realization considerations apart, the actual mathematical point about detecting PAH bifurcations through the topological classification maps is that for a-controlled PAH bifurcations to occur, all parallel lines to the a-axis must intersect the zero-level hypersurface of Trace[AX] transversally. And they do. Transversal intersection is a natural geometric (visual) condition, not needing, in general, further verification.


Even though the study of Kalman controllability and observability conditions [6] is a fundamental aspect of the analysis of a linear dynamical control system, for the sake of this paper we will include it in next section within the control system design problem.

3 Integrated Design of Nonlinear Dynamical Control Systems

In the sequel, we firstly approach the design of linear state-feedback controllers, PID controllers, and Luenberger observers, to asymptotically stabilize the NLDCS (1) according to the Jacobian linearization strategy. Later, we proceed to calculate the nonlinear extensions of these linear controllers and observers, to stabilize the NLDCS (1) using the nonlinear extension technique [7], [8]. Linear state-feedback controllers and Luenberger observers are designed using the standard pole-assignment technique [6]. To design linear PID controllers we also used a pole-assignment algorithm, because in most cases Ziegler-Nichols techniques do not apply to second-order plants [9]. Nonlinear extensions of previously designed linear controllers and observers are calculated using the symbolic algorithms developed in our previous works [10], [11], [12]. For the sake of this paper, we included constructing Kalman’s controllability and observability matrices and checking theirs rank condition, as an integral part of the integrated symbolic-graphic-numeric design notebooks. Yet, these sections of the
notebooks may be considered also, by their own right, as part of the analysis strategy of a linear control system.

**Fig. 3-a.** Existence of spiral equilibria.

Disc[AX] < 0 between the two layers of the 0-level hypersurface of Disc[AX], wherefore all associated systems have an spiral point at the origin.

**Fig. 3-b.** PAH-bifurcation points.

The 0-level hypersurface of Trace[AX] intersects the open (d, r, a)-subspace where Disc[AX] < 0. So, spiralling equilibria above (below) Trace[AX] = 0 are repelling (attracting). The set of PAH-bifurcation points is PAH-Set = {(d, r, a) | Trace[AX] = 0 & Disc[AX] < 0}.

**Fig. 3-c.** Transversal crossing at PAH-bifurcation Points.

Vertical lines intersect the PAH-Set transversally. Hence, after the PAH bifurcation Theorem, a-controlled PAH bifurcations do occur at the PAH-Set.

**Fig. 3-d.** Spiral vs. Saddle-Points.

The zero level hypersurfaces of Det[AX] and Disc[AX] do not intersect each other. So, all spirals actually spiral.

**Fig. 3.** The tunnel-diode circuit does oscillate.
Fig. 4. a-controlled oscillatory behaviour of the tunnel diode circuit. We study PAH bifurcation along the right most vertical line in Fig. 3-c, i.e., \((a, d, r) = (a, 1.75, 0.1)\). For these values of \(d\) and \(r\), the PAH-bifurcation point is located at \(a = rd = 0.175\).

As it will transpire, within an integrated symbolic-graphic-numeric computation environment like Mathematica®’s notebooks it is relatively easy to assemble, simulate and compare the performances of different open-loop or closed-loop control systems. All we have to do is gathering together the appropriate set of defining equations, fixing the desired initial conditions, parameter’s values, and simulation intervals, and running the notebooks like they were computer programs. So, on relaying on notebooks, we eliminate the necessity of using a symbolic package for designing linear and nonlinear controllers and observers, a graphic tool to assembly the closed-loop systems, a numeric package to simulate and plot performances, and a word processor to document the whole work. Moreover, and perhaps even more important, we simplify the whole procedure and increase its reliability, because we need neither to transfer data between different control applications nor creating interfaces between them.
3.1 The Structure of the Integrated Design Notebook

For practical and expository reasons, we will summarize our present work on integrated symbolic-graphic-numeric control systems design in two notebooks: one for nonlinear extended state-feedback control, and another one for nonlinear extended PID control. These design notebooks share as a common core with the analysis notebook described above all facilities allowing to calculate operating points, linearizations, coordinates translations, etc.

The basic notebook integrating symbolic, numeric and graphic calculations to design nonlinear extended state-feedback controllers has the standard structure described below. In the notebook for nonlinear extended PID control we omit the sections where controllability and observability studies are performed, and substitute them by a transfer-function section where open-loop and closed-loop transfer-functions, and ultimate frequencies and gains, can be calculated. Those sections where linear and nonlinear state-feedback control laws are calculated were consistently substituted by appropriate sections for the calculation of linear and non-linear PID control laws.

1. Introduction. Idem Section 2.1
2. Modelling. Idem Section 2.1
3. Declaring the Open-Loop Control System. Idem Section 2.1
4. Operating Points. Idem Section 2.1
5. Linearization. Idem Section 2.1
6. Linear and Nonlinear Open-Loop Dynamics. Through input cells we define and solve the ODE representing the open-loop linear and nonlinear plants. This allows us to plot and compare how linear and nonlinear open-loop dynamics approach the operating point.
7. Controllability. Using matrices A and B, we construct Kalman’s Controllability Matrix \([B; AB; \ldots; A^{n-1}B]\), and evaluate its rank.
8. Linear State-Feedback Control Law Design. Using input cells we construct the linear closed-loop system \(\dot{x} = (A – BK)x\), and calculate its characteristic polynomial. Then we input the design specifications, use the pole-assignment algorithm to calculate the state-feedback gain matrix K, and calculate the linear control law \(u(t) = -Kx(t)\).
9. Linearly Controlled Closed-Loop Dynamics. We symbolically construct and numerically solve the linearly controlled closed-loop \(\dot{x} = f(x, -Kx)\). We may then plot and compare open-loop dynamics vs. linearly controlled close-loop dynamics.
10. Nonlinear State-Feedback Control Law Design. Here we use symbolic algorithms in \([12]\) to calculate a nonlinear extended control law \(u = -k(x)\), such that \(\text{grad } k(X) = K\).
11. Nonlinearly Controlled Closed-Loop Dynamics. The nonlinearly controlled closed-loop system \(\dot{x} = f(x, -k(x))\) is symbolically constructed and numerically solved. At this point, we may compare the performances of the open-loop, the linearly controlled and the nonlinearly controlled systems, i.e., \(\dot{x} = f(x, -KX)\), \(\dot{x} = f(x, -Kx)\), and \(\dot{x} = f(x, -k(x))\), respectively.
12. Observability. Using matrices A and C, we construct Kalman’s Observability Matrix \([C; CA; \ldots; CA^{n-1}]\), and evaluate its rank.
13. **Linear Luenberger Observation Law Design.** The observation error equation \( \dot{e} = [A - GC]e \) and its characteristic polynomial are symbolically constructed. Design specifications are input, and the pole-assignment used to calculate the observation gain \( G \).

14. **Linearly Observed-Controlled Closed-Loop Dynamics.** Here we input the closed-loop system: \( \dot{x} = f(x, u), u = -Kz, \dot{z} = Az + Bu + G(y - w), w = Cz, y = Cx. \) Trajectories are calculated and plotted.

15. **Nonlinear Luenberger Observation Law Design.** The nonlinear Luenberger observer \( \dot{z} = f(z, u) + g(y) - g(w) \) is calculated, using algorithms is [11].

16. **Nonlinearly Observed-Controlled Closed-Loop Dynamics.** The nonlinear closed-loop system \( \dot{x} = f(x, u), u = -h(z), \dot{z} = f(z, u) + g(y) - g(w), y = h(x), w = h(z) \) is symbolically input and numerically solved. Trajectories may be plotted and compared with those of the linearly observed and controlled closed-loop system.

17. **Other Closed-Loop Topologies.** Were we interested on, we may define and study other nonlinear closed-loop systems. For instance, the nonlinearly controlled and linearly observed or the linearly controlled and nonlinearly observed closed-loop systems. We may also build up the linear closed-loop systems consisting of the linearized plant \( (A, B, C, D) \), the linear controller \( K \), and the linear Luenberger observer \( G \).

### 3.2 Case Study 2. A Mass-Spring-Damper van der Pol System

In what follows, we consider a mass-spring-damper realization of van der Pol equation as an example of a nonlinear plant, which we first stabilize by linear and then by nonlinear state-feedback control. Because of the lack of space, we will suppress nearly all output cells containing the symbolic results generated by Mathematica®. We will also leave aside the design of the Luenberger observers.

1. **Introduction.** The particular mass-spring-damper realization of van der Pol equation we are considering is shown in Fig. 5.

![Fig. 5. Mass-spring-damper realization of van der Pol equation. The force generated by the damper is \( b(x_1^2 - 1)x_2 \).](image)

2. **Modelling the System.** After applying Newton’s second law and choosing the position and velocity of the mass \( m \) as the states of the system, we obtain the following first order representation of the system:

\[
\begin{align*}
\dot{x}_1 &= x_2, \\
\dot{x}_2 &= \frac{1}{m} \left[ -kx_1 - b(x_1^2 - 1)x_2 + u \right], y = x_1. 
\end{align*}
\]  

3. **Declaring Equations**

\[
\text{In[]. Clear[x1, x2, X1, X2, A, AX, TrA, DETA, p, DiscA, a, d, r];} \\
\text{In[]. f1[x1_, x2_, u_] := x2;}
\]
4. Equilibrium Points

\[ f(x_1, x_2, u) := (k x_1 - b(x_1^2 - 1)x_2 + u)/m; \]

\[ h(x_1, x_2, u) := x_1; \]

5. Linearization

\[ A(x_1, x_2, u) := \text{Simplify}\left(\begin{bmatrix} D[f_1(x_1, x_2, u, x_1), x_1] & D[f_1(x_1, x_2, u, x_2)] \\ D[f_2(x_1, x_2, u, x_1)] & D[f_2(x_1, x_2, u, x_2)] \end{bmatrix}\right) \]

\[ B(x_1, x_2, u) := \text{Simplify}\left(\begin{bmatrix} D[f_1(x_1, x_2, u, u)] \\ D[f_2(x_1, x_2, u, u)] \end{bmatrix}\right) \]

6. Linear and Nonlinear Open-Loop Dynamics

\[ k = 0.2; b = 0.3; m = 5; Xp = 3; \]

\[ x_{10} = -2; x_{20} = 0.5; t_0 = 0; t_1 = 50; x_m = -5; x_M = 5; x_{10} = x_{110}; x_{20} = x_{210}; \]

\[ x = \{x_{11}(t) - X_1, x_{21}(t) - X_2\}; \]

\[ \text{Lin} = A(x_{11}(t), x_{21}(t), u(t)) \times B(x_{11}(t), x_{21}(t), u(t)) \times \left\{ \begin{array}{c} u(t) - U \end{array} \right\}; \]

\[ \text{Linp} = \text{Lin} / \left\{ k \rightarrow k_p, b \rightarrow b_p, m \rightarrow m_p, X \rightarrow Xp \right\}; \]

7. Controllability

\[ M(x_1, x_2, u) := \text{AppendRows}(B(x_1, x_2, u), A(x_1, x_2, u) \times B(x_1, x_2, u)); \]

8. Linear State-Feedback Control Law Design

\[ K := \{k_1, k_2\}; \]

\[ ABK := s \times \text{IdentityMatrix}[2] - A(x_1, x_2, u) + B(x_1, x_2, u) \times K; \]

Open-loop and closed-loop dynamics are shown in Fig. 6.
Integrated Symbolic-Graphic-Numeric Analysis and Design in Nonlinear Control

\[\text{In}[.\] \quad \text{KS} = \text{Solve}\{\text{CPK}[1] == \text{CPd}[1], \text{CPK}[2] == \text{CPd}[2]\}, \{k1, k2\};
\text{In}[.\] \quad K1 = k1 /. \text{KS}[1]; K2 = k2 /. \text{KS}[1];
\text{In}[.\] \quad K = \{K1, K2\}; \text{MatrixForm}[K];
\text{In}[.\] \quad Xd = \{\{x1d\}, \{x2d\}\}; \text{MatrixForm}[Xd];
\text{In}[.\] \quad \text{ud}[x1d_, x2d_] := -(K.Xd)[[1]]; \text{In}[.\] \quad u[x1_, x2_] := \text{Collect}[\text{Simplify}[U + \text{ud}[x1 - X, x2]], \{x1, x2\}];
\text{Out}[.\] \quad m \times x1 d^2 + x2 (b( -1 + X^2)) – 2 m \times x2 d \omega d) + x1 (k – m \times \omega d^2)

\text{Fig. 6-a.} Position dynamics of the nonlinear system.
\text{Fig. 6-b.} Position dynamics of the linearized system.

9. Linearly Controlled Closed-Loop Dynamics
\[\text{In}[] \quad \text{xclp}' = \{f1[x1cl[t], x2cl[t], u[x1cl[t], x2cl[t]]], f2[x1cl[t], x2cl[t], u[x1cl[t], x2cl[t]]]\} /\{k -> kp, b -> bp, m -> mp, X -> Xp, \xi d -> 0.7, \omega d -> 1\}
\text{In}[] \quad \text{SisNoLinCont} = \{x1c'[t] == \text{xclp}'[1], x2c'[t] == \text{xclp}'[2], x1c[0] == x10, x2c[0] == x20\}
\text{In}[] \quad \text{SolSisNoLinCont} = \text{NDSolve}[	ext{SisNoLinCont}, \{x1cl[t], x2cl[t]\}, \{t, t0, t1\}]
\text{In}[] \quad \text{Grafx1cl} = \text{Plot}[\{\text{Evaluate}[x1cl /. \text{SolSisNoLinCont}], Xp\}, \{t, t0, t1\}, \text{PlotRange} -> \{xm, xM\}, \text{PlotStyle} -> \{\text{RGBColor}[0.6, 0, 0], \text{RGBColor}[0, 0, 1]\}, \text{AxesLabel} -> \{t, \"x1cl Xp"\}]

The linearly controlled position dynamics, x1c11[t], is shown in Fig. 8-a.

10. Nonlinear State-Feedback Control Law Design
\[\text{In}[] \quad \kappa 1[s1_, s2_] := K1 /\{X1 -> s1, K2 -> s2\};
\kappa 2[s1_, s2_] := K2 /\{X1 -> s1, K2 -> s2\};
\text{In}[] \quad \text{I1}[x1_, x2_] := \text{Integrate}[\kappa 1[s1, s2], \{s1, X, x1\}];
\text{In}[] \quad \text{I2}[x1_, x2_] := \text{Integrate}[\kappa 2[s1, s2], \{s2, 0, x2\}];
\text{In}[] \quad \text{unl}[x1_, x2_] := -(\text{Collect}[U + \text{I1}[x1, x2] + \text{I2}[x1, x2], \{x1, x2\}]); \text{Out}[] \quad m \times x1 d^2 – x2 (b — b x1^2 + 2 m \times \xi d \omega d) – x1 (k – m \times \omega d^2)

11. Nonlinearly Controlled Closed-Loop Dynamics
\[\text{In}[] \quad \text{xcln}' = \{f1[x1cnl[t], x2cnl[t], unl[x1cnl[t], x2cnl[t]]], f2[x1cnl[t], x2cnl[t], unl[x1cnl[t], x2cnl[t]]}\];
The nonlinearly controlled position dynamics, \( x_{1\text{cnl}}[t] \), is shown in Fig. 7-b. Fig. 7 also contains the linear and the nonlinear state-feedback control signals.

**3.3 Discussion**

As Fig. 7-a and 7-b show, the nonlinear extended control law decreases the overshoot in the position dynamics. Moreover, this improved response is obtained using a smaller nonlinear control signal. For the lack of space we did not include in this paper, the observability study, the design of the linear and the extended Luenberger observers, the assembly of the different observed and controlled closed-loop topologies, and the corresponding simulations. Yet, this results, and the associated notebooks, are electronically available upon request.
4 Perspective and Future Work

By the time we are finishing the present paper some more general versions of the notebooks described above are available, allowing to design linear and nonlinear controllers and observers for higher order systems. At the present time, our work is evolving towards a library of specific packages that may be invoke anytime from a notebook under development. This facilities will, in principle, support nonlinear systems of arbitrary order. We have recently given a first step towards considering discrete-time linear systems. A first application of discretization, together with Mathematica® animated graphics facilities, has been the development of very basic mimic plants, upon which we could exhibit the effect of the designed control laws. The improvement of these mimics is one of our short term goals.

Integrated symbolic-graphic-numeric notebooks covering additional nonlinear control techniques and more general control topologies will also be available in the next half year. The development of a graphic facility allowing to transform sets of algebraic and differential equations into block diagrams would also be highly desirable.

References

4. Rodríguez-Millán, J., Constructing Parametrized Families of Bode Plots in Mathematica®, Cuadernos de Control, Volume 2001-1, Graduate Program in Control and Automation, University of Los Andes, Mérida, 2001.
9. Sira-Ramírez, H., Control por Linealización Extendida, Cuadernos de Control, Volume 94-2, Graduate Program in Control and Automation, University of Los Andes, Mérida, 1996.
Acknowledgments

I would like to express my deep feelings of gratitude to Prof. Dr. Bruno Buchberger for the inspiring influence that our many talks at RISC about mathematics, computing and teaching have had upon my work since then. Somehow, our present work on integrated computation for nonlinear control is also an offspring of Bruno’s philosophy supporting Theorema.
Towards a Computer-Aided Design of Reactive Systems*

Marc Frappier and Richard St-Denis

Département de mathématiques et d’informatique
Université de Sherbrooke
Sherbrooke, Québec, CANADA J1K 2R1
{Marc.Frappier, Richard.St-Denis}@dmi.usherbro.ca

Abstract. We consider a new approach to synthesize abstract machines for reactive programs that interact with processes in order to achieve some control requirements in the context of the Supervisory Control Theory. The motivation behind our synthesis approach is related to the problem of scalability. Generally, synthesis procedures are based on a comparison of two state spaces (fixpoint calculation-based approach) or an exploration of a state space (search-based approach). In neither case do the synthesis procedures scale to specifications of realistic size. To circumvent this problem, we propose: i) to combine two formal notations for the representation of reactive programs in addition to the one for specifying control requirements; and ii) to integrate a synthesis procedure in a framework in which various transformations are applied with the sole aim of solving a smaller control problem from an abstract model.

1 Introduction

There are a wide range and diversity of safety-critical systems and their importance in many areas stems from the need to supervise the operations of a process efficiently, while simultaneously satisfying several specific-domain constraints. In this context, a clear distinction is made between the reactive program (P) and the process (S) it controls. Typically, the process S to be controlled, also called a plant, is a physical system and the reactive program P, also called a controller or a supervisor, is a hardware or software implementation of a control algorithm. A reactive program must be built in such a way that its interaction with the process satisfies a certain specified property (p). To increase reliability of safety-critical systems, it is often recommended to use formal methods in their development [2]. These include mathematically-based specification and reasoning techniques to verify or synthesize a solution [17]. On one hand, formal verification refers to a variety of methods used to prove the satisfaction of a given property by a system to be checked. On the other hand, formal synthesis refers

* The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds pour la formation de chercheurs et l’aide à la recherche (FCAR).
to a category of methods that focus on the calculation of a system satisfying a given property. Verification and synthesis methods require first and foremost that a model $\mathcal{M}$ represents the process $\mathcal{S}$. Let $\mathcal{M} \parallel \mathcal{P}$ denote the interaction of $\mathcal{P}$ with $\mathcal{M}$, also called the feedback loop. The verification problem and the synthesis problem are formulated as follows.

Verification problem – Given $\mathcal{M}$, $p$, and $\mathcal{P}$, verify that $(\mathcal{M} \parallel \mathcal{P}) \models p$. (1)

Synthesis problem – Given $\mathcal{M}$ and $p$, compute $\mathcal{P}$ such that $(\mathcal{M} \parallel \mathcal{P}) \models p$. (2)

Verification and synthesis approaches are dual in the sense that if either problem (1) or (2) has a solution, so does the other. However, the reactive program $\mathcal{P}$, often constructed by empirical means in a verification approach, is not always optimal with respect to some criteria (e.g., minimally restrictive, lower cost) defined in a synthesis approach [14]. Since the synthesis problem is the dual of a verification problem, the method that we propose to solve it is based on emerging ideas in the verification domain [16]. Their concrete expression is, however, different because there is more information in the latter. The reactive program $\mathcal{P}$ missing in the former must be computed.

In contrast to verification techniques that have successfully been used by industry, notably for communication protocols and hardware devices, the effectiveness of synthesis techniques has not yet been demonstrated. A major issue with current synthesis techniques seems to be their insufficient scalability because of the state explosion problem. The synthesis community did not pay special attention to this issue and few techniques have been proposed to reduce the computational complexity. Either they reduce the complexity by a factor of $n$ when the process consists of $n$ similar components [13] or they have a polynomial complexity but to the detriment of a weaker level of reliability since errors are detected during system operations, which is unacceptable for a safety-critical system [4].

The method that we propose to tackle the scalability issue combines several mathematical techniques borrowed from different research domains. First, it is founded on a theory for discrete event systems, the Supervisory Control Theory [11] [14], that gives conditions for the solvability of synthesis problems and provides various synthesis procedures to generate correct solutions. Second, it encompasses three different formal notations: a temporal logic to specify the required system properties [10], an algebraic specification language to describe abstract data types modeling uncontrollable passive objects [6], and transition structures to represent the behavior of controllable active components [19]. In addition to their formal syntax and semantics, each of these notations has an underlying theory that allows automated reasoning about different aspects of the system. This contrasts with the dual language approach, very widespread in the verification domain, in which no distinction is made between passive and active components [8]. Finally, it relies on recent advances in the verification domain, particularly on the soundness property that must be preserved by transformations useful for downscaling the concrete model of a system [8] [15]. The sole aim of the integration of several mathematical techniques into a consistent framework
is to make feasible the design exploration of realistic reactive programs by means of a synthesis approach.

The paper is structured as follows. Section 2 presents an overview of the method and major related issues. Section 3 shows the weaknesses of conventional synthesis methods and roughly illustrates the main phases of the method with the aid of an example. Section 4 details the crucial phases of the method. It provides formal definitions of some transformation rules preserving correctness. It also indicates the level of human intervention required in each phase and the reasons why some of them cannot be fully mechanized. Finally, Section 5 ends the paper with a discussion on related work and some concluding remarks.

2 Overview of the Method and Major Issues

Figure 1 shows the main phases of the method. It is oriented towards the calculation of solutions from abstract models instead of a concrete model. The crucial task concerns the abstraction of irrelevant parts from the concrete model to obtain a reduced model that can be evaluated by human beings and handled by a synthesis procedure. Since it is utopian to determine appropriate abstractions in only one trial, they are identified and evaluated through repeated runs of a combined modeling-parameterization-reduction-synthesis-merging cycle. If the obtained solution is unsatisfactory, or if alternative designs are to be evaluated, the cycle is repeated until a satisfactory solution is reached.

Three major related issues must, however, be studied when synthesis problems are solved by following the schema in Fig. 1. The first issue pertains to the selection of an appropriate formalism to represent reactive programs with the sole aim of avoiding an exhaustive search of the whole state space. The second issue concerns the use of transformations and their implementation through theoretically-based techniques to support automated software production to the extent possible. Finally, the third issue is related to the formulation of conditions that maintain the correctness property guaranteed by conventional synthesis procedures in the sense that the answer to the question that appears in Fig. 1 must be positive. Each of these issues are detailed below.

2.1 Representation of Reactive Programs

A reactive program is represented by an attributed controller, a notion introduced in [18]. It comprises an attributed transition structure to monitor the sequence of actions executed by the process, a conditional control policy to restrain process behavior by disabling some controllable actions, and typed objects to represent passive components involved in the evolution of the process or in the formulation of the required properties. The major feature of this new formalism is the capacity to associate operations, which can be applied to typed objects, with actions. These associations are considered not only by the synthesis procedure, but also during feedback loop execution.
Symbols representing the identities of passive components are defined from algebraic specifications of abstract data types that support the use of hidden sorted algebra appropriate for the object paradigm \[\text{[7]}\]. The internal states of an object are ground terms derived from constructors of the corresponding abstract data type. At each evolution time of the feedback loop, the attributed controller observes the action executed by the process and applies the operations associated with it to some objects. This results in new terms that are rewritten in a simpler form by using equations defined in the corresponding abstract data types. The global effect is a change in the internal state of each object involved in the current step. Therefore, an attributed controller bases its decisions not only on the current state of its transition structure, as in the conventional approach, but especially on the current state of its objects through the evaluation of conditions, represented by Boolean terms, in accordance with Boolean operations and equations that appear in the specification of abstract data types.

Fig. 1. Overview of the method

There are two reasons for the adoption of this algebraic approach. First, in comparison with the conventional representation of reactive programs by transition structures and control policies, the addition of typed objects provides a means to memorize the current state of passive components and to express control requirements in a simple and concise manner that should otherwise be
represented by huge state-transition models. Second, it is the extensive algorithmic and theoretical support available for equational reasoning, particularly algorithms for term rewriting, used in the synthesis and merging phases.

2.2 Transformations

The method depicted in Fig. 1 supports a strategy in which two kinds of transformations are exploited: transformations for reduction and transformations for merging.

In the reduction phase, one must consider component reduction and data reduction. Component reduction is relevant when one may abstract from differences between active components because they have the same behavior and play the same role. They are considered as being partitioned into a small number of equivalence classes. It should be noted that it is not sufficient to retain only one representative active component per equivalence class, because of various connections (e.g., synchronization, physical constraints, direction of material flow) between passive and active components, but the number of active components should be kept as small as possible in accordance with conditions for maintaining the correctness property. Data reduction is useful when one may abstract from real internal dimensions of passive components and then consider only small ranges of values instead of all the possible values involved in the dynamics of the concrete model. Since the method has been primarily developed for discrete event systems, the case of continuous variables with infinite, but bounded, spaces of values is ignored in this paper.

Working on a single large and monolithic specification, which encompasses all components and the environment, is not feasible in practice. Some prerequisites for effective downscaling should be fulfilled before the reduction phase. In the modeling phase, one must consider the process to be controlled as composed of individual active components. Generally, the active components are not sufficient to specify control requirements and the system description needs to be augmented with additional passive components. In the parameterization phase, one must specify the system in a parametric form. Internal dimensions of passive components and the cardinalities of equivalence classes of similar components are replaced by symbolic parameters of unknown, fixed value. Scalar quantities are not assigned to these parameters until the reduction phase. Instead of calculating a controller with respect to specific values, the objective is to construct a controller regardless of parameter values or, if not possible, for the larger set of admissible values.

The automated procedure used in the synthesis phase to calculate an intermediate controller expands a finite labeled directed graph from timed transition graphs of active components, a formula $f$ that expresses the control requirements, and the application of operations on objects in relation to equations of abstract data types and associations between actions and operations. The labeled directed graph generated by the synthesis procedure contains the final transition structure of the intermediate controller. The expansion involves a verification of $f$ over trajectories of the process while taking uncontrollable actions.
into account [1]. From this expansion results a state space where each state is a \((m + n)\)-tuple, \(m\) and \(n\) denoting the number of active components and the number of passive components, respectively. The representation of a global state by a \((m + n)\)-tuple allows symbolic reasoning in terms of local states of each component, since we have many states of many components instead of a single state of a monolithic transition model. In addition to the finite labeled directed graph, the synthesis procedure calculates a control policy that maps each state \((\langle q, t \rangle)\) onto a set of controllable actions \(A_k\) that must be prohibited (called a control action) when the current state of the controller is \(\langle q, t \rangle\) during feedback loop evolution. An entry of the control policy has the form \(\langle q_1, \ldots, q_m, t_1, \ldots, t_n \rangle : A_k\).

Finally, in the merging phase, one must reduce the number of entries of the control policy. This can be realized by moving the decision logic embedded in a memoryless structure, the transition structure and control policy of the intermediate controller, to a more compact memory structure. Some local states, which are less and less relevant to discriminate between control actions, are removed as Boolean terms are assigned to control actions. In the merging phase, one must then consider term or state abstraction and flattening. Term abstraction consists in finding a Boolean term \(b_i\) equivalent to the ground term \(t_i\) \((1 \leq i \leq n)\), through the use of operations and equations of abstract data types, and assigning the conjunction \(c = b_1 \land \cdots \land b_n\) to the control action \(A_k\) in order to obtain a conditional control action. At this stage, the local states \(t_1, \ldots, t_n\) and condition \(c\) can be considered as interchangeable. Flattening consists in syntactically merging conditional control actions and removing ground terms in the corresponding global states. Consider the following two entries:

\[
\langle q_1, \ldots, q_m, t_1, \ldots, t_n \rangle : \langle A_1 : c \rangle \text{ and } \langle q_1, \ldots, q_m, t'_1, \ldots, t'_n \rangle : \langle A_2 : c' \rangle.
\]

If condition \(c\) is false when it is evaluated from terms \(t'_1, \ldots, t'_n\) and condition \(c'\) is false when it is evaluated from \(t_1, \ldots, t_n\), then the previous two entries could be merged as follows: \(\langle q_1, \ldots, q_m \rangle : \langle A_1 : c \rangle \langle A_2 : c' \rangle\). The transition structure must then be rearranged by considering the states \(\langle q_1, \ldots, q_m, t_1, \ldots, t_n \rangle\) and \(\langle q_1, \ldots, q_m, t'_1, \ldots, t'_n \rangle\) as equivalent. A similar technique can be applied by considering the local states of active components. They cannot, however, be directly replaced by Boolean terms unless suitable objects are further added in the modeling phase. Nevertheless, one can take advantage of some situations in which all the conditional control actions are independent of the states of active components. State abstraction consists in checking the self-reliance of conditional control actions through the evolution of objects. In summary, term abstraction and state abstraction make it possible to obtain an attributed controller that works for a process whatever the dimension of passive components and the number of active components. This logically based symbolic reasoning technique must be done in a way that does not introduce inconsistencies.

### 2.3 Sound Abstraction

The notion of sound abstraction introduced in the domain of verification [8] can be adapted for the sequence of activities shown in Fig. [1]. The procedure used
in the synthesis phase generates an intermediate controller $C^I$ that is correct with respect to an abstract model $(\mathcal{M}_A, f_A)$, that is, $(\mathcal{M}_A \parallel C^I) \models f_A$. In the merging phase, the intermediate controller is transformed into an attributed controller $C^A$. The transformations used in the merging phase maintain correctness if $(\mathcal{M}_A \parallel C^A) \models f_A$. Since the attributed controller $C^A$ must be embodied in a feedback loop that includes the physical process (modeled by $\mathcal{M}$), it must perform in the same way for equivalent actions. The attributed controller modified to take into consideration equivalence classes of components identified in the modeling phase is denoted by $\overline{C^A}$.

**Definition 1.** Soundness – The abstract model $(\mathcal{M}_A, f_A)$ is a sound abstraction for the concrete model $(\mathcal{M}, f)$ if $(\mathcal{M}_A \parallel C^A) \models f_A$ implies $(\mathcal{M} \parallel \overline{C^A}) \models f$.

Downscaling carried out by using the four transformations introduced in Section 2.2 must preserve soundness as defined above. Because some of these transformations require human assistance, the effectiveness of the method does not only depend on techniques for reduction, but also on conditions specific to a family of control problems. Given a control problem, they must be checked to assert that reductions proposed by humans are admissible. This aspect is discussed further in Section 5. Finally, it should be noted that the notion of complete abstraction is not considered in this paper.

### 3 An Illustrative Example

To explain the motivation behind the method proposed in the preceding section and to illustrate it, let us consider $l$ independent users sharing a single resource [14]. Figure 2 shows the timed transition graph $G_i$ that represents the behavior of user number $i$ ($1 \leq i \leq l$). There are three states: $I_i$ (Idle), $R_i$ (Requesting), and $U_i$ (Using). For instance, the user moves from state $I_i$ to state $R_i$ on action $\alpha_i$ (request the resource), then from state $R_i$ to state $U_i$ on action $\beta_i$ (allocate the resource), and finally, from state $U_i$ to state $I_i$ on action $\gamma_i$ (release the resource). Every action has a duration of one time unit and actions $\beta_i$ are controllable. The timed transition graph modeling the free behavior of the process is obtained by taking the shuffle product of $G_1, \ldots, G_l$.

A controller must be derived in order to satisfy the following two constraints: only one user can own the resource at one time (mutual exclusion) and the resource is allocated according to first-come, first-served policy (fairness). These two constraints are expressed by the following metric temporal logic formula $f$ written in a compact form:

$$f = \square_{\geq 0}(\neg(U_i \land U_j) \land ((\neg R_i \land U_{\geq 0} R_j) \rightarrow (\neg U_i \land U_{\geq 0} U_j)))$$

(3)

for all $i$ and $j$ such that $1 \leq i \leq l$, $1 \leq j \leq l$, and $i \neq j$. 

Towards a Computer-Aided Design of Reactive Systems
3.1 Computational Complexity

It is easy to show that the number of states in the timed transition graph modeling the free behavior of the process is $3^l$. The following expression gives the number of states in the transition structure of the derived controller with respect to $l$ by using the synthesis procedure proposed by Barbeau et al. [1].

$$3^l - \sum_{k=0}^{l-2} \binom{l}{k} 2^k + \sum_{k=2}^{l} \binom{l}{k} (l-k+1)(k!-1)$$

Expression (4) includes three terms. The first term represents the total number of states in the timed transition graph modeling the free behavior. The second term gives the number of states in which there are at least two users sharing (using) the resource. These states are bad and must be removed to satisfy the mutual exclusion constraint. The last term corresponds to the fairness constraint and is more complex. The subterm $\sum_{k=2}^{l} \binom{l}{k} (l-k+1)(k!-1)$ gives the number of states in which there are at least two users requesting the resource, but without any information about the order of these requests. These states are useless and must be replaced by $\sum_{k=2}^{l} \binom{l}{k} (l-k+1)k!$ states. The last subterm gives the number of states that must be added to take into consideration the order in which the
resource is requested. For realistic values of \( l \), the state space is so huge that conventional synthesis approaches are unworkable.

### 3.2 Outline of a Solution

In this control problem, there is only one equivalence class of active components that includes \( l \) users. Usually, the value of \( l \) is fixed for a concrete problem but, in the parameterization phase, the variable \( l \) is a parameter that models the number of users. In the modeling phase, one must also identify passive components. Even though not explicitly mentioned in the problem description, a queue can be naturally introduced to take into consideration the fairness constraint and express it in a simpler form. A queue is defined using the following algebraic specification:

\[
\text{Queue}(\text{capacity} \in \text{nat}) :=
\]

import: bool, nat, element

hidden sorts: queue

operations:
- \( \text{New}: \to \text{queue} \)
- \( \text{Add}: \text{queue} \times \text{element} \to \text{queue} \)
- \( \text{Remove}: \text{queue} \to \text{queue} \)
- \( \text{Front}: \text{queue} \to \text{element} \)
- \( \text{IsEmpty}: \text{queue} \to \text{bool} \)
- \( \text{IsHead}: \text{queue} \times \text{element} \to \text{bool} \)

equations: \( q \in \text{queue} \), \( e, e' \in \text{element} \)

\[
\begin{align*}
\text{Remove}(\text{New}) &= \text{New} \\
\text{Remove}(\text{Add}(q, e)) &= \text{if IsEmpty}(q) \text{ then New else Add}(\text{Remove}(q), e) \\
\text{Front}(\text{New}) &= \text{ERROR} \\
\text{Front}(\text{Add}(q, e)) &= \text{if IsEmpty}(q) \text{ then } e \text{ else Front}(q) \\
\text{IsEmpty}(\text{New}) &= \text{TRUE} \\
\text{IsEmpty}(\text{Add}(q, e)) &= \text{FALSE} \\
\text{IsHead}(\text{New}, e) &= \text{FALSE} \\
\text{IsHead}(\text{Add}(q, e'), e) &= \text{if IsEmpty}(q) \text{ then } \text{if Eq}(e, e') \text{ then TRUE else FALSE } \text{ else IsHead}(q, e)
\end{align*}
\]

An object \( q \) of type \( \text{Queue} \) with capacity \( l \) is then declared. Its initial internal state is given by the ground term \( \text{New} \). In this step, operations must be associated to actions: \( \langle \alpha_i, q \rangle \mapsto \text{Add}(q, i) \) and \( \langle \gamma_i, q \rangle \mapsto \text{Remove}(q) \). In an object-oriented paradigm, the notation \( q.Add(i) \) is used to abbreviate the expression \( \text{Add}(q, i) \).

In the reduction phase, the value of the only parameter, \( l \), can be fixed to two (\( l = 2 \)) and formula (5) as well as an instance of formula (3), with \( l = 2 \), can be considered. Nevertheless, the former allows reasoning not only about states of active components but also in terms of internal states of passive components through the use of Boolean operations.
\[ f_A = \Box_{\geq 0}(\neg (U_1 \land U_2) \land (\neg U_2 U_{\geq 0} \neg Is_{\text{Head}}(q, 1)) \land (\neg U_1 U_{\geq 0} \neg Is_{\text{Head}}(q, 2))) \]  

The intermediate controller calculated in the synthesis phase from the transition graphs \( G_1, G_2 \), object \( q \), and formula \( f_A \) has nine states and the following control policy:

\[
\begin{align*}
\langle I_1 \ I_2 \ New \rangle : \ {} & \{ \} \quad \langle R_1 \ R_2 \ Add(Add(New, 1), 2) \rangle : \{ \beta_2 \} \\
\langle R_1 \ I_2 \ Add(New, 1) \rangle : \ {} & \{ \} \quad \langle R_1 \ R_2 \ Add(Add(New, 2), 1) \rangle : \{ \beta_1 \} \\
\langle I_1 \ R_2 \ Add(New, 2) \rangle : \ {} & \{ \} \quad \langle U_1 \ R_2 \ Add(Add(New, 1), 2) \rangle : \{ \beta_2 \} \\
\langle U_1 \ I_2 \ Add(New, 1) \rangle : \ {} & \{ \} \quad \langle R_1 \ U_2 \ Add(Add(New, 2), 1) \rangle : \{ \beta_1 \} \\
\langle I_1 \ U_2 \ Add(New, 2) \rangle : \ {} & \{ \}
\end{align*}
\]

Clearly, \( Is_{\text{Head}}(q, i) \) gives TRUE when this Boolean term is evaluated with the substitution \( [q \ Add(Add(New, i), j)] \ (i = 1, 2) \), yielding the following intermediate conditional control policy:

\[
\begin{align*}
\langle I_1 \ I_2 \ New \rangle : \ {} & \{ \} \\
\langle R_1 \ I_2 \ Add(New, 1) \rangle : \ {} & \{ \} \quad \langle I_1 \ R_2 \ Add(New, 2) \rangle : \{ \} \\
\langle U_1 \ I_2 \ Add(New, 1) \rangle : \ {} & \{ \} \quad \langle I_1 \ U_2 \ Add(New, 2) \rangle : \{ \} \\
\langle R_1 \ R_2 \ Add(New, 1), 2 \rangle : \{ \beta_2 : Is_{\text{Head}}(q, 1) \} \\
\langle R_1 \ R_2 \ Add(New, 2), 1 \rangle : \{ \beta_1 : Is_{\text{Head}}(q, 2) \} \\
\langle U_1 \ R_2 \ Add(New, 1), 2 \rangle : \{ \beta_2 : Is_{\text{Head}}(q, 1) \} \\
\langle R_1 \ U_2 \ Add(New, 2), 1 \rangle : \{ \beta_1 : Is_{\text{Head}}(q, 2) \}
\end{align*}
\]

Based on the facts that controllable actions that are physically impossible in the process from a given state can arbitrarily be enabled or disabled and that the conditional control action

\[ \{ \langle \beta_2 : Is_{\text{Head}}(q, 1) \rangle, \langle \beta_1 : Is_{\text{Head}}(q, 2) \rangle \}, \]

when applied to any state, has the same effect as the previous conditional control policy, then the nine entries of the conditional control policy can be merged into a single entry in the merging phase. The following conditional control action

\[ \{ \langle \beta_1 : \neg Is_{\text{Head}}(q, 1) \rangle, \langle \beta_2 : \neg Is_{\text{Head}}(q, 2) \rangle \} \]

is another possibility. The second solution is better than the first one because it works regardless of the value \( l \). The next section explores how such a solution can be systematically derived.

## 4 Design Exploration by Means of Synthesis Approach

### 4.1 Preliminary Definitions

An attributed controller is a structure \( \mathcal{S} := (Q_a, Q_o, A, \delta_a, \delta_o, q_{0_a}, q_{0_o}, \varphi) \), where \( Q_a \) and \( Q_o \) are finite set of states of the active components and objects, respectively; \( A \) is a finite set of actions; \( \delta_a : Q_a \times A \rightarrow Q_a \) and \( \delta_o : Q_o \times A \rightarrow Q_o \) are partial transition functions; \( q_{0_a} \in Q_a \) and \( q_{0_o} \in Q_o \) are the initial states of
the active components and objects, respectively; and $\varphi : Q_a \times Q_o \rightarrow \mathbb{P}(A)$ is the control policy that determines which actions are inhibited in a given state.

The object types are defined using algebraic specifications. The function $\delta_o$ is defined using the operations provided in the algebraic specifications. During the synthesis procedure, equations of the algebraic specifications are used to compute normal forms (through rewriting) for the controller’s object states.

The state of an attributed controller is given by a pair $\langle q_a, q_o \rangle$. The initial state is $\langle q_{0a}, q_{0o} \rangle$. When the process executes an action $a$, the controller moves to a new state $\langle \delta_a(q_a, a), \delta_o(q_o, a) \rangle$. In a given state $\langle q_a, q_o \rangle$, the controller inhibits actions $\varphi(\langle q_a, q_o \rangle)$.

4.2 Transformations Preserving Correctness

The synthesis procedure produces a control policy $\varphi$ given by an enumerated set of pairs $\{ \ldots, \langle q, t \rangle : A_k, \ldots \}$ where $q$ and $t$ denote a list of local states and a list of ground terms, respectively. The merging phase consists in finding a new representation of this control policy that is more efficient in terms of space and algorithmic complexity. This new representation is called a conditional control policy, noted $\varphi'$, and expressed as a set of pairs $\{ \ldots, \langle a_i : c_i(q_o) \rangle, \ldots \}$. This set of pairs has the following meaning: $\varphi'(t) = \{ a_i \mid c_i[q_o\setminus t] \}$. Note that the conditional control policy $\varphi'$ does not depend on $q$, the state of the active components.

The conditional control policy must be equivalent to the original control policy. Physically impossible actions (i.e., actions that cannot occur in a given state of the active components) are taken into account to determine if two control policies are equivalent. Formally, the set $\omega(q_a)$ or physically impossible actions in a state $q_a$ is defined as $\omega(q_a) = \{ a \mid \langle q_a, a \rangle \notin dom(\delta_a) \}$. Two control policies are said to be equivalent iff, for all pairs $\langle q, t \rangle : A_k$ of the original control policy, we have $A_k \cup \omega(q) = \varphi'(t) \cup \omega(q)$. In other words, when an action is inhibited or physically impossible in the original control policy, it must also be inhibited of physically impossible in the conditional control policy.

The use of algebraic specifications enables an automatic search through the Boolean terms (conditions) of the algebra to find terms to construct $\varphi'$. Heuristics are used to find such terms, given that the algebraic specification satisfies some hypotheses enabling semi-automatic theorem proving and term rewriting (see [9]). When no appropriate terms are found, designer intervention is required. Designer suggestions can be checked for correctness using a theorem prover.

The first step of the merging phase (term abstraction) consists in finding, for each pair $\langle q, t \rangle : A_k$ of the original control policy $\varphi$, a condition $c(q_o)$ such that $c[q_o\setminus t]$ holds. These conditions define an intermediate control policy $\varphi''$ given as an enumerated set of pairs $\{ \ldots, \langle q, t_i \rangle : \{ (a_i : c_i(q_o)) \}, \ldots \}$ for each $a_i \in A_k$. It is interpreted as follows: $\varphi''(q, t) = \{ a_i \mid c_i[q_o\setminus t] \land \langle q, t \rangle = \langle q, t_i \rangle \}$. When $A_k$ is empty, the pair $\langle q, t_i \rangle : \{ \}$ is used. By definition, $\varphi''$ is equivalent to $\varphi$. Note that the second component of a pair of $\varphi''$ is a set whose structure is the same as a control policy $\varphi'$; it is called a local conditional control policy. The conditional control policy is obtained by merging, in a step-wise, equivalence
preserving manner, the local conditional control policies until all states have the
same local control policy.

We propose some equivalence-preserving transformation rules to refine an
intermediate control policy into a conditional control policy.

Rule 1 (Flattening)
Two pairs \(<q_1, t_1>: \varphi_1'\) and \(<q_2, t_2>: \varphi_2'\) such that \((\varphi_1' \cup \varphi_2')(t_1) \cup \omega(q_1) = \varphi_1'(t_1) \cup \omega(q_1)\) and \((\varphi_1' \cup \varphi_2')(t_2) \cup \omega(q_2) = \varphi_2'(t_2) \cup \omega(q_2)\) can be replaced, respectively, by the pairs \(<q_1, t_1>: \varphi_1' \cup \varphi_2'\) and \(<q_2, t_2>: \varphi_1' \cup \varphi_2'\).

This rule is repeatedly applied until all control policies are merged. If it cannot
be applied, it is either because the conditions chosen for the local conditional
control policies are inadequate or because the objects chosen are inadequate. The
conditions are inadequate when they are too weak, thereby inhibiting an action
in a state \(t\) when it should not be. In that case, the conditions must be strength-
ened. The following rule states this fact.

Rule 2 (Strengthen local policy)
A pair \(<q, t>: \{ \cdots, \langle a : c_1(q_o)\rangle, \cdots \} \) can be replaced by a pair \(<q, t>: \{ \cdots, \langle a : c_2(q_o)\rangle, \cdots \} \) when \(c_2[q_o \setminus t] \rightarrow c_1(q_o)\).

When all local conditional control policies have been merged, the intermedi-
ate control policy can be replaced by a conditional control policy. The next rule
expresses this fact.

Rule 3 (State abstraction)
An intermediate control policy \(\{\langle q_1, t_1>: \varphi', \cdots, \langle q_k, t_k>: \varphi'\}\) can be replaced
by the conditional control policy \(\varphi'\).

In the example in Section 3, Rule 1 can be applied repeatedly until all local
control policies are merged, yielding the following conditional control policy:

\[\{\langle \beta_1: Is\_Head(q, 2)\rangle, \langle \beta_2: Is\_Head(q, 1)\rangle\}\].

This policy is valid for two active components. If the same exercise is conducted
for three components, the following conditional control policy is obtained:

\[
\{\langle \beta_1: Is\_Head(q, 2) \lor Is\_Head(q, 3)\rangle, \langle \beta_2: Is\_Head(q, 1) \lor Is\_Head(q, 3)\rangle,
\langle \beta_3: Is\_Head(q, 1) \lor Is\_Head(q, 2)\rangle\}.
\]

Clearly, the conditional control policies obtained for 2 and 3 components are not
adequate, since they are not solutions for an arbitrary number of active com-
ponents. Here again, there are two possible explanations: the conditions chosen
are inadequate or the objects chosen are insufficient. A human intervention is
required to solve this problem.

Analyzing the resulting control policies for 2 and 3 components, one may
observe that, for each action \(\beta_i\), the negation of condition \(c_i\) denotes states of the
forms \(New\) or \(Add^k(Add(New, i), \cdots)\), for \(1 \leq k < l\). This set of states is denoted
by the condition $Is\_Empty(q) \lor Is\_Head(q, i)$. The negation of this condition is equivalent to condition $c_i$, and is syntactically independent of the number of components. Hence, the conditional control policy $\{\langle \beta_i : \neg(Is\_Empty(q) \lor Is\_Head(q, i)) \rangle\}$ for $1 \leq i \leq l$ is valid. It also seems to be sound. We have no proof of its soundness, but it is not difficult to convince oneself that it is. Proving soundness is an open problem that we are currently investigating.

Other control policies could be found. It depends on the heuristics used to find conditions in the first step of the merging phase. For instance, the weaker and syntactically simpler condition $\neg Is\_Head(q, i)$ can also be used. Note that this condition already appears in formula (5). Hence, a good heuristic may be to try conditions appearing in the constraints applied to the process.

### 4.3 Summary of Transformation Characteristics

Table 2 summarizes the transformations that are used in the method and their characteristics in terms of automated support, soundness provability, and correctness provability.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Automated Support</th>
<th>Proving Soundness</th>
<th>Proving Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Reduction</td>
<td>N</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Component Reduction</td>
<td>N</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Term Abstraction</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>State Abstraction</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Flattening</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

Data reduction and component reduction are manual transformations that must be accomplished by the designer. Computer assistance is very difficult to provide, because these steps require a good knowledge of the problem domain. Proving the soundness of an abstract model is an open problem. Intuitively, some form of inductive argument over the number of components seems to be reasonable, but it turns out that the induction hypothesis is difficult to use. The attributed controller is generated by state exploration, not by symbolic reasoning. For each value of $l$, a state exploration is done. It seems difficult to lift up this state exploration and embody it in an inductive proof. Moreover, the metric temporal logic extended with universal and existential quantification of first-order logic should be considered.

Term abstraction, state abstraction, and flattening are amenable to automated support when algebraic specifications are used. Moreover, it is possible to show that correctness is preserved by these transformations. Simple rules that preserve correctness have been presented. The proof obligations could be conducted using symbolic reasoning and term rewriting. Another solution, less
general but completely automatable, is to use models for the algebraic structures (i.e., concrete implementations of objects) and to exhaustively check that a formula holds. These models are usually small, so this is quite feasible in practice.

5 Conclusion

The research reported in this paper was originally motivated by the issue of scalability arising from synthesis procedures. The gist of our approach is to decompose the process to be controlled into active and passive components and describe them in a parametric form. It is then possible to investigate the design of a reactive program from an abstract model by applying some reductions to the concrete model. The method relies on different specification formalisms and on the integration of several logical and equational reasoning techniques implemented in a symbolic manner. This new unified method is an adequate support for prototyping by means of a synthesis approach.

Our approach could be seen as a formalization of intuitive methods that several designers use. Most designers typically solve a problem by first solving a simplified version (an abstract model). For instance, in telephony, services are first explored for scenarios involving few users. The solution is then extended to cover an arbitrary number of users. There is no guarantee that the solution and its extended version are correct with respect to either the abstract model or the concrete model. In our approach, the solution to the abstract model is automatically derived using the Supervisory Control Theory. The solution is extended in a systematic manner, using correctness preserving transformations. Therefore, the extended solution is correct with respect to the simplified version of the problem. Proving that the solution is also correct with respect to the concrete problem remains an open question.

It appears that very few attempts to put into practice synthesis procedures have been carried out in the context of the Supervisory Control Theory. Many obstacles remain before they will be powerful enough to tackle a range of real-world applications. Some are technical, including the development and integration of specialized tools to support the whole method presented in this paper or similar paradigms. It is not expected that such paradigms can be fully automated because human assistance is still required. Others are theoretical, involving the study of conditions on the transformation of the process model under which it is possible to use a controller, easily calculable from a simplified model, for the original concrete model. There are some partial results in this direction but they do not explicitly refer to the issue of scalability (e.g., [3], [5], and [12]).

The aforementioned work is related to the problem of preserving correctness and soundness. This important problem requires further research. The aim of presenting this work at present was not to prove particular results but to show how various techniques can be integrated to obtain a unified framework suitable for the synthesis of reactive programs. The example presented in this paper has been selected to demonstrate the potential of our method. Nevertheless, we are aware of limits because program synthesis, in general, is not yet sufficiently
mature for usage in software practice. Much work remains to be done to achieve this goal. This paper is a step in that direction.

References


Contributions to the Control and Stabilization of the Pole-Cart System

Agustin Raya¹, Dario Maravall²

¹ Department of Applied Physics and Mechanics for Agroforestal Engineering
Universidad Politécnica de Madrid
28040 Madrid, Spain
araya@fis.etsia.upm.es

² Department of Artificial Intelligence
Universidad Politécnica de Madrid
28660 Boadilla del Monte, Madrid, Spain
dmaravall@fi.upm.es

Abstract. The organization of the paper is as follows. First, we obtain the dynamical equations of the pole-cart system by applying the Lagrange equations for the two variables of interest: the cart position and the deviation angle of the pendulum with respect to the vertical line. The corresponding system of two differential equations is highly nonlinear, with a rather restricted analytical treatment. After obtaining the approximate, linear equations of the pole-cart couple, we discuss its control and stabilization throughout an external force that depends on the deviation angle of the pendulum. The type of feedback control schemes that stabilize the pendulum on its unstable, vertical position are discussed in the paper. We also illustrate such stabilization with several simulated cases, for both the complete, nonlinear version and the approximate, linear version. Conventional P and PD control schemes are applied, with excellent results. Finally, we approach the evaluation of the computer-based implementation by means of computer simulations and obtain the critical design parameters for the control and the stabilization of the pole-cart system.

1 Introduction

The inverted pendulum is one of the best known and probably most studied nonlinear systems [1] – [3], mainly for three reasons: (a) its relative tractability from the mathematical point of view, (b) its importance as a basic phenomenon appearing in many mechanical and engineering systems, [4] – [6], and (c) the challenges that this apparently simple physical system poses to its control and stabilization, [7] and [8].

In this paper we approach the problem of the inverted pendulum on top of an electrical cart and we develop a complete analytical solution to the stabilization of the inverted pendulum, which for the first time in the technical literature, to the best of our knowledge, formally explains the well known practical experience that in order to
stabilize the pendulum in the vertical position on top of a cart, an oscillatory movement of the latter has to be produced. Almost everyone has experimented with such situation by trying to maintain an inverted broom in its vertical position by moving forwards and backwards the finger or the hand.

The organization of the paper is as follows. First, we obtain the dynamical equations of the pole-cart system by applying the Lagrange equations for the two variables of interest: the cart position and the deviation angle of the pendulum with respect to the vertical line. The corresponding system of two differential equations is highly nonlinear, with a rather restricted analytical treatment. On the other hand, the practical interest of the pole-cart pair is centered on its behavior in the proximity of its equilibrium point, i.e. the vertical line. In such region of interest the system can be analyzed in depth by means of its linearization about the equilibrium position.

After obtaining the approximate, linear equations of the pole-cart couple, we discuss its control and stabilization throughout an external force that depends on the deviation angle of the pendulum. The type of feedback control schemes that stabilize the pendulum on its unstable, vertical position are discussed in the paper. We also illustrate such stabilization with several simulated cases, for both the complete, nonlinear version and the approximate, linear version. Conventional P and PD control schemes are applied, with excellent results.

The most interesting conclusion of this part of the paper is that in order to stabilize the inverted pendulum, a necessary and sufficient condition is to apply an external force based on the feedback provided by the current deviation angle of the pendulum. This general result applies for both the linear and the nonlinear versions. Obviously, a physical sensor for the measurement of the pendulum’s deviation angle and its coupling to a computer system are needed for the practical implementation of the external force. As a consequence, the discrete-time version of the pole-cart dynamical equations has to be obtained. Once established the corresponding difference equations, we develop a similar analysis for its control and stabilization, which is obviously realized by means of an external force based on the deviation angle. As a theoretical curiosity, we analytically explain the oscillatory behavior of the electrical cart when it stabilizes the inverted pendulum. Representative simulations of the discretized pole-cart couple are finally expounded and the critical design parameters for the control and stabilization of the inverted pendulum are finally obtained.

Figure 1 shows an inverted pendulum on top of an electrical cart, also known as the pole-cart system. The pole-cart electromechanical system is composed of an inverted pendulum of punctual mass \( m \) and length \( l \) placed over a mobile cart of mass \( M \) which is subject to a lateral force \( F \). The mass of the pendulum’s bar can be considered negligible as compared with the punctual mass \( m \) and its motion, frictionless.

The cart position with respect to the origin \( O \) is given by the coordinate \( x \). The pendulum’s deviation from the vertical line is given by the angle \( \theta \). The objective of the force \( F \) is to maintain the pendulum in its equilibrium position \( \theta = 0 \), which is highly unstable. This force can be produced by an electrical motor on the cart.
2 Stabilization of the Pole-Cart System in the Continuous-Time Domain

In order to obtain the dynamical equations of the pole-cart system we will apply the Lagrange equations:

\[
\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = F_i, \quad i = 1,2
\]

\[ L = T - U \quad (1) \]

Where \( L \) is the lagrangian, \( T \) is the kinetic energy and \( V \) is the potential energy of the system. \( F_i \) are the generalized forces and \( q_i \) are the generalized coordinates, that for the pole-cart pair are \( x \) and \( \theta \). Obviously, \( F \) is the only generalized force.

The kinetic energy consists of two components: one due to the cart and the other to the pendulum:

\[ T = \frac{M}{2} x^2 + \frac{m}{2} v^2 \quad (2) \]

where \( v \) is the pendulum’s velocity and its coordinates are:

\[ x + l \cdot \text{sen} \theta; l \cdot \cos \theta \quad (3) \]

By differenciating these coordinates with respect to time it is immediate to obtain the pendulum’s kinetic energy:

\[
\frac{m}{2} \left[ \left( \frac{d}{dt} (x + l \cdot \text{sen} \theta) \right)^2 + \left( \frac{d}{dt} l \cdot \cos \theta \right)^2 \right] \quad (4)
\]
which substituted into (2) gives the total kinetic energy:

\[ T = \frac{M}{2} x'^2 + \frac{m}{2} [(x' + l\theta' \cos \theta)^2 + (l\theta' \cdot \text{sen} \theta)^2] \]  

(5)

The potential energy is:

\[ U = mgl \cdot \text{cos} \theta \]  

(6)

As F is an horizontal force, by applying equations (1) to the energies given by expressions (5) and (6), we obtain the Lagrange’s equations for \( q_1 = x \) and \( q_2 = \theta \):

\[ \frac{d}{dt} \left( \frac{\partial L}{\partial x'} \right) - \frac{\partial L}{\partial x} = F; \quad \frac{d}{dt} \left( \frac{\partial L}{\partial \theta'} \right) - \frac{\partial L}{\partial \theta} = 0 \]  

(7)

After some calculation the dynamical equations of the pole-cart system can be written as follows:

\[ (M + m)x'' + ml\theta'' \cos \theta - m(l\theta')^2 \text{sen} \theta = F \]
\[ x'' \cos \theta + l\theta'' - g \text{sen} \theta = 0 \]  

(8)

This is a highly non-linear system with a rather limited analytical treatment. At the same time, the practical interest of this system is centred on its behaviour in the proximity of its equilibrium point \( \theta = 0 \). In such region of interest the system can be analyzed in depth by means of its linearization about the equilibrium position.

To linearize the inverted pendulum we introduce the following change in the Lagrangian \( L \):

\[ \text{sen} \theta \rightarrow \theta ; \text{cos} \theta \rightarrow 1 - \frac{\theta^2}{2} \]  

(9)

that result from taking the first term in the Taylor series development of \( \sin \theta \) and the first and second terms of \( \cos \theta \). By substituting (9) into the equations (5) and (6) of the kinetic and potential energies we obtain:

\[ T = \frac{M}{2} x'^2 + \frac{m}{2} (x' + l\theta')^2; \quad U = mgl \left( 1 - \frac{\theta^2}{2} \right) \]  

(10)

The term \( (l\theta' \cdot \text{sen} \theta)^2 \) can be considered negligible in the proximity of \( \theta = 0 \). Furthermore, the constant term of the potential energy can be also neglected, so that we can write the Lagrangian as:

\[ L = \frac{M}{2} x'^2 + \frac{m}{2} (x' + l\theta')^2 + mg \frac{\theta^2}{2} \]  

(11)
The corresponding Lagrange equations are:

\[
(M + m)x'' + ml\theta^* = F \\
mlx'' + ml^2\theta^* - mgl\theta = 0
\]  
(12)

It is immediate to obtain the individual equations for each independent coordinate:

\[
x'' + \frac{mg}{M}\theta = \frac{F}{M} \\
\theta'' - \frac{(M + m)g}{Ml}\theta = -\frac{F}{Ml}
\]  
(13)

Now, we are going to analyze the stability and controlability of the pole-cart system when there is no force F present –i.e. the autonomous case-. In such situation, the second equation in (13) that gives the pendulum’s deviation from the vertical is:

\[
F = 0 \Rightarrow \theta'' - \frac{(M + m)g}{Ml}\theta = 0
\]  
(14)

The characteristic equation and its corresponding roots are:

\[
r^2 - \frac{M + m}{Ml}g = 0 \Rightarrow r_{1,2} = \pm \sqrt{\frac{M + m}{Ml}}g
\]  
(15)

The dynamics of the deviation angle \( \theta \) is governed by hyperbolic sines and cosines and therefore \( \theta \) grows limitless with \( t \) tending to infinite. This unstable behaviour can only be stabilized by the application of a force \( F \). The physical interpretation of the stabilization of the inverted pendulum by applying an external force is the decelerating effect of such force on the dynamics of the pendulum’s deviation from its equilibrium position.

In order to design an effective stabilization mechanism for the pole-cart system we will introduce a negative feedback, where the basic idea is to apply an external force \( F \) which is a function of the deviation angle \( \theta \) as shown in figure 2.

![Fig. 2. Stabilization of the pole-cart system by negative feedback](image-url)
The error variable is:
\[ e(t) = \theta_d - \theta(t) = 0 - \theta(t) = -\theta(t) \]  
(16)
as the objective is to maintain the inverted pendulum on its equilibrium point \( \theta = 0 \) in which case the error will be null.

The simplest stabilization scheme is the proportional control:
\[ F = -k_p e = k_p \theta \quad k_p > 0 \]  
(17)
By substituting (17) into the second equation of (13) we have:
\[ \theta^* + \frac{1}{Ml} \left( k_p - (M + m)g \right) \theta = 0 \]  
(18)
By choosing \( k_p > (M + m)g \) the solution of equation (18) is stable, as it is formed by damped sinusoids and therefore it holds:
\[ \lim_{t \to \infty} \theta(t) = 0 \]  
(19)
In most practical situations, however, the single proportional control is not efficient enough, so that a more powerful proportional+derivative PD control scheme should be attempted:
\[ F = -k_p e - k_d e' = k_p \theta + k_d \theta' \]  
(20)
that substituted into the second equation of (13) yields:
\[ \theta^* + \frac{k_d \theta'}{Ml} + \frac{1}{Ml} \left( k_p - (M + m)g \right) \theta = 0 \]  
(21)
This dynamical equation gives a solution based on the roots of the characteristic equation:
\[ r_{1,2} = -\frac{k_d}{2Ml} \pm \sqrt{\frac{k_d^2}{4M^2 l^2} - \frac{1}{Ml} \left( k_p - (M + m)g \right)} \]  
(22)
We have the two free parameters \( k_d \) and \( k_p \) to guarantee the stability of the deviation angle about its equilibrium position \( \theta = 0 \). Basically, \( k_d \) and \( k_p \) are chosen to place the roots on the left half-plane of the complex plane to simultaneously control the speed and the fiability of the dynamical trajectory of \( \theta(t) \) towards its unstable equilibrium position.

More complex control laws like the full proportional+integral+derivative control, i.e. the popular PID algorithm, can be applied for the stabilization of the inverted pendulum, although the use of additional free parameters can create some trouble in the overall pendulum’s stabilization process.

To finish this analytical treatment of the pole-cart system, we are now going to analyze the interesting particular case of a force \( F \) proportional to the deviation error, equation (17), with restriction (19) for the control parameters. If we write:
the general solution of the pendulum dynamics, equation (18), is:

$$\theta = \theta_0 \cos wt + \frac{\theta'_0}{w} \sin wt$$  \hspace{1cm} (24)$$

where \(\theta_0\) and \(\theta'_0\) are the initial position and velocity of the deviation angle, respectively. In this case the cart dynamics, first equation in (13), results to be:

$$x' = \frac{k_p - mg}{M} \theta$$  \hspace{1cm} (25)$$

By integrating (25) we obtain:

$$x' = x'_0 + \frac{k_p - mg}{Mw^2} \left[ \frac{\theta_0}{w} \sin wt + \frac{\theta'_0}{w} \left(1 - \cos wt\right)\right]$$  \hspace{1cm} (26)$$

as for \(t = 0\) \(x'\) is \(x'_0\). Equation (26) shows that the cart movement is composed of (1) a uniform movement at velocity:

$$x'_0 + \frac{k_p - mg}{Mw^2} \theta'_0$$  \hspace{1cm} (27)$$

and (2) a pure sinusoid oscillation. If we wished to cancel out the uniform movement it would be necessary to apply an initial velocity \(x'_0\) such that cancels expression (27):

$$x'_0 = \frac{mg - k_p}{Mw^2} \theta'_0$$  \hspace{1cm} (28)$$

in which case the velocity should be:

$$x' = \frac{mg - k_p}{Mw} \left( \frac{\theta'_0}{w} \cos wt - \theta_0 \sin wt \right)$$  \hspace{1cm} (29)$$

By integrating (26) we obtain for the cart position:

$$x = x_0 + x'_0t + \frac{mg - k_p}{Mw^2} \left[ \frac{\theta_0}{w} \left(\cos wt - 1\right) + \frac{\theta'_0}{w} \left(\sin wt - wt\right)\right]$$  \hspace{1cm} (30)$$

as for \(t = 0\), \(x\) is \(x_0\). If expression (28) holds, then there is no uniform movement and this equation simplifies to:

$$x = x_0 + \frac{mg - k_p}{Mw^2} \left[ \theta_0 \left(\cos wt - 1\right) + \frac{\theta'_0}{w} \sin wt \right]$$  \hspace{1cm} (31)$$
which is a remarkable result obtained through an analytical treatment and which means that the cart follows a curious sinusoidal movement. On the other hand, this analytical result confirms the well-known oscillatory movement necessary to stabilize a broom with a hand on its unstable equilibrium point: i.e. the vertical position. From a practical point of view, the physical restriction given by expression (28) is almost impossible to guarantee and in such case the real cart trajectory is composed of a linear movement at constant velocity \( x' \) and the oscillatory movement given by the last term in expressions (30) or (31).

3 Experimentation via Computer Simulations

Due to the highly non-linear nature of the pole-cart system and the corresponding lack of analytical and general results, [7] and [8], the experimental work is absolutely mandatory. The experimentation with a mechanical system via computer simulations presents significant advantages as compared with its physical experimentation. In the case of the inverted pendulum, where high nonlinearities are present and, above all, the system’s behaviour is extremely fast and very hard to measure, computer simulations are still more attractive [9]. In effect, we have shown that both the deviation angle and its derivative are used for the feedback force that stabilizes the system and we know that these variables are extremely hard to measure with enough accuracy. To make things worse, the physical experimentation needed for the selection of the control and stabilization algorithms depend not only on the range of values of the control parameters, but also on the specific values of the physical elements of the pole-cart system: the mass of the cart and the pendulum’s mass and length. However, in despite of the evident advantages of computer simulation as regard to flexibility, it must be reminded its sensitivity to the mathematical model’s accuracy [9]. In the sequel, a concise summary of the results obtained via computer simulations are presented.

3.1 Ideal Pole-Cart System Simulation.

The analytical results obtained in the preceding paragraphs are only valid in the proximity of the unstable equilibrium point. Therefore, it is very interesting to check whether or not these results extend beyond the linear approximation by a thorough simulation of the pole-cart non-linear model. To begin with, we summarize the main analytical results obtained for the linearized model:

1. The system is completely unstable without an external force.
2. Negative feedback provides an apparently robust mechanism for the stabilization of the pendulum.
3. Furthermore, feedback requires a visual sensor and a computer system for its implementation.

For the preliminary simulations, we have ignored the elements in point 3 above, and the corresponding “ideal” version of the system has been simulated. By ideal is meant that the sensors and the computer in charge of the control are not considered.
As a consequence, in this ideal pole-cart system the following parameters have not been included in the simulations: (1) sampling period, (2) delay time for the measurement of the angle and its derivative and (3) delay time of the control action. The concrete values of the pole-cart system’s physical elements used in the simulations reported below are the following: pendulum mass, 0.5 Kgs; pendulum length, 0.25 m; cart mass, 2 Kgs and force limit, 20 Nw. The basic design parameters under scrutiny have been the initial deviation angle $\theta_0$ and the control parameters $k_p$ and $k_d$. The graphics below display the initial deviation $\theta(0)$ and the system’s physical parameters.

### 3.1.1 Pure Proportional Control

These graphics confirm the analytical results; i.e. for a pure P control, the non-linear pole-cart model produces an oscillatory stabilization. It is interesting to remark that the oscillations amplitude is always equal to the initial deviation $\theta(0)$ and the frequency depends on $k_p$ and the system’s physical parameters.

### 3.1.2 Proportional + Derivative Control

With the PD control the analytical results have been also confirmed by the simulations and from figure 4 it can be observed that even for very high initial deviation angles the system is quickly stabilized.

The general conclusions can be summarized as follows.

− The continuous, “ideal” non-linear system provides an upper limit to the pole-cart’s behaviour, so that further simulations of the computer-based version are necessary.
− P and PD control algorithms guarantee suitable stabilization, even for high initial deviations.

The continuous linear and non-linear models present rather similar qualitative behaviour, in particular for small and medium initial deviations.
3.2 Simulation of the Computer-Based Pole-Cart System

We have shown that in order to stabilize the pole-cart system it is compulsory the application of an external force $F$, which is a function of the deviation angle of the pendulum and its derivative. Obviously, a physical sensor for the measurement of these variables and its coupling to a computer are needed for the practical implementation of such external force. As a consequence, the main performance design parameters to be considered and evaluated in the simulation of the computer-based system are: (1) the sampling period of the discretized model, $T_s$, (2) the delay time $\tau_{obs}$ of the observed deviation angle and its derivative and (3) the delay time of the control action, $\tau_{act}$.

3.2.1 Evaluation of the Sampling Period and the Global Delay Time

In a first evaluation, only the global delay time, $\tau = \tau_{obs} + \tau_{act}$, together with the sampling period, has been considered. The range of values used in the results reported below is the following: $T_s = 20$ ms, 100 ms, 500 ms; $\tau = 20$ ms, 70 ms; $\theta(0) = 1^\circ$, $5^\circ$, $25^\circ$.

The applied force has been limited to 20 Nw, which is equivalent to a maximum acceleration of $10 \ m/s^2$. This limit is an important practical design constraint, as the system can be stabilized for, in principle, any initial deviation angle if there is no limit in the magnitude of the applied force.

The main conclusion is that for the sampling period and for the combined delay time there are critical stabilization limits, approximately around 20 ms. Generally speaking, $T_s$ is less critical than both delay times; therefore, an extra analysis of the delay times influence on the stabilization is mandatory. Furthermore, as the sampling
Contributions to the Control and Stabilization of the Pole-Cart System

\[ \theta(0) = 5^\circ; Ts = 20ms; \tau = 20ms \]

\[ \theta(0) = 25^\circ; Ts = 20ms; \tau = 20ms \]

\[ \theta(0) = 1^\circ; Ts = 0.1s; \tau = 70ms \]

\[ \theta(0) = 5^\circ; Ts = 0.1s; \tau = 20ms \]

**Fig. 5.** Evaluation of the influence on stabilization of sampling period and global delay time.

period is a flexible design parameter, in the sense that it can be reduced to as much as the duration of the combined delay times, we have considered \( Ts = \tau_{obs} + \tau_{act} \).

3.2.2 Analysis of the Influence of the Observation and the Actuation Delay Times

The rationale in this analysis is to investigate the critical values of both delay times that make the pole-cart pair uncontrollable. To this aim, we have simulated for each initial deviation angle different observation and actuation delay times, until the system stops being stabilized. Apart from the usual delay times, we have also considered an initial delay time which is the time until the control force begins to act.

Figure 6 displays several results for initial deviations of 1º, 5º and 25º, as usual, and for different combinations of the delay times. The general conclusion is that for combined delay times less than 40 ms, the stabilization is always guaranteed and beyond that limit, the system begins to be poorly controlled and stabilized.

4 Conclusions

An analytical study of the control and stabilization of the pole-cart system by means of negative feedback has been presented. Afterwards, an upper limit of the system’s performance has been obtained by the computer simulation of the “ideal” non-linear system –i.e. the continuous, non-linear mathematical model-. The following step has been to evaluate the system’s computer-based implementation, also by means of computer simulations. The main design parameters for such evaluation have been the sampling period, the observation delay time and the actuation delay time. Throughout
extensive computer simulations –which allow a much more complete and profound analysis than the physical experimentation- it has been possible to find the critical design parameters for the control and the stabilization of the pole-cart system.

References


Abstract. It is known that the techniques bracketed under the topic of Soft Computing have a strong capability of learning and cognition joint to a good tolerance with uncertainty and imprecision. Due to these properties they can be applied successfully in Intelligent Vehicles Systems. In particular Fuzzy Logic is very adequate to build qualitative or linguistic models of many kinds of systems. The aim of this paper is to integrate in a qualitative model the vehicle operation and the driver behavior in such a way that an unmanned guiding system can be developed around it [2] [6].

1 Introduction

There is an increasing stream of applications that include the emerging techniques of Soft Computing -fuzzy logic, neural and evolutionary computing, machine learning and probabilistic reasoning- to cope with the great challenges of the information age. The expected performances of intelligent vehicle systems are among these challenges [3].

Due to their strong learning and cognitive ability and good tolerance with uncertainty and imprecision, soft computing techniques have found wide application in the field of intelligent vehicle systems. The present work is framed in this field but limited to on board driving engines based on fuzzy logic. For instance, adaptive cruise control, collision avoidance, lane tracking and driver behavior modeling are systems that can be approached with the qualitative model herein described.

The present work is aiming at the field of intelligent vehicles but limiting its scope to on board driving engines based on fuzzy logic systems. Due to their strong cognitive ability, good tolerance with uncertainty and imprecision and easiness to do linguistic and understandable models, fuzzy logic techniques have found wide application in the field of intelligent vehicle systems. In effect, the decisions made for a driver are based on approximate perceptions of the environmental and traffic conditions, the velocities of the surrounding vehicles, etc. By now the only approach to compute perceptions is fuzzy logic. For instance, adaptive cruise control, collision avoidance, lane tracking and driver behavior modeling are systems that can be approached with the qualitative model herein described.
2 Kinematic Foundations

At the most abstract level a vehicle and their movements can be modeled by a sequence of states in a vectorial space. The elements of this space are vectors which components define the current state of the vehicle, its position, velocity and acceleration respectively, according with the classic control theory.

\[ \mathbf{z} \]

\[ \mathbf{z}' \]

\[ \mathbf{z}'' \]

In this state space formal operators that change the values of the current parameters and pass the vehicle from a state to another can be defined, but what is relevant for the purpose of this work is that, in practice, these operators are equivalent to the actions of the guidance system, either a person or an automatic control system.
3 Qualitative Model

The main property of a qualitative model must be its closeness and understandability to the user. This consideration lead us to an alternative representation of the vehicle state, more convenient for a qualitative model, in which the explicit parameters are more intuitive for the user than the parameters of the classic control form (1). In this new representation the state of the vehicle is the expression:

\[ S = ((x, y), q, v, c) \]  \hspace{1cm} (2)

where \((x, y)\) stands for the current position of the vehicle, \(q\) stands for the current orientation, \(v\) stands for the module of the current velocity and \(c\) stands for the curvature of the current trajectory of the vehicle. It is clear that all these parameters represent intuitive concepts that all the drivers have acquired, what can not be said of the parameters of the expression (1).

In the expression (2) the parameters take integer values with the following meanings: A positive orientation means that the angle forming between the axe of the vehicle tangent to the trajectory is clockwise (state \(S_i\), Fig. 2) and a negative value means that this angle is counter clockwise (state \(S_k\), Fig. 2). Similarly a positive value for the curvature means that the current turn is clockwise (state \(S_i\), Fig. 2), by the contrary, a negative value means that the current turn is counter clockwise (state \(S_k\), Fig. 2). A positive velocity means that the course is forward and a negative velocity is that the course is backward.

![Graphical representation of the parameters of position, orientation, and radius of curvature of two states of the vehicle, according with the expression (2).](image)

The operators of the state space can be defined according to the mechanical behavior of the vehicle that, in a first cinematic approach, can be modeled by the intrinsic movement equations of a point rigidly linked to the vehicle, for instance the center of gravity, the center of the rear axe, or any other point in which a position sensor can be situated. These intrinsic equations are:

\[ S_i = ((x, y), q, v, c) \]

\[ S_j = ((x, y), q, v, c) \]

---

1 To simplify the height vehicle position is no considered
where $\vec{u}$ and $\vec{n}$ are unitary vectors tangent and normal to the trajectory respectively.

By other hand, the real control actions are performed over the steering wheel and the accelerator and brake pedals respectively. Thus, taking into account the intrinsic movement equations, the operators of the vehicle state space can be modeled by one array of actions

$$A_i = (\Delta q_i, \Delta v_i)$$

being $\Delta q_i$ the increment or decrement of the angle of the steering wheel and $\Delta v_i$ the increment or decrement of the velocity in each control cicle.

Finally, the abstract model explained before allows representing the route of a vehicle as a sequence of states, from one initial $S_i$ to one final $S_f$, being each state $S_j$ the state of the vehicle at the beginning of each control cycle. Similarly the movements can be represented by a sequence of actions $A_f \ldots A_i \ldots$ being each action $A_j$ the control action computed in the current control cycle. In this model is not relevant the nature of the control agent, can be a human being or an automatic system.

### 4 Control Model

As whichever control system, the procedure employed to guide an unmanned vehicle calculates the control actions in each control cycle in function of the differences, or signal errors, between to consecutive states. In classic controllers these differences are calculated from the intrinsic movement equations, even from the dynamical equations if it was necessary to take into account the stability of the vehicle. In this work the control procedure is a fuzzy controller, so these differences are modeled qualitatively as well as the behavior of a human driver by a set of linguistic “if … then …” rules. From a user point of view a great advantage of this qualitative model is that it can be formulated from the user experience with sentences close to the user natural language.

The fuzzy controller performs as an uncoupled system, so the direction and the velocity can be controlled separately. In fact there are two sets of rules, one set controls the acceleration, so the control outputs act over the accelerator and brake pedals, and a second set controls the direction, so the control output acts by turning the steering wheel.
As an example of building of the qualitative model of the guidance system the control of the direction is considered. Its objective is maintaining the desired trajectory. As it is shown in the Fig. 3 two linguistic variables, named here deviation and yaw, model the differences perceived by the driver between the current and the desired state of the vehicle. By other hand the qualitative model of the driver is formed by the set of rules [4] contained in the following knowledge base:

<table>
<thead>
<tr>
<th>Knowledge base for the direction control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong></td>
</tr>
<tr>
<td>yaw { null }</td>
</tr>
<tr>
<td>deviation { left, right }</td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
<tr>
<td>steering { left, right }</td>
</tr>
<tr>
<td><strong>Rules</strong></td>
</tr>
<tr>
<td>IF yaw MORE THAN null THEN steering right</td>
</tr>
<tr>
<td>IF yaw LESS THAN null THEN steering left</td>
</tr>
<tr>
<td>IF yaw null AND deviation left THEN steering right</td>
</tr>
<tr>
<td>IF yaw null AND deviation right THEN steering left</td>
</tr>
</tbody>
</table>

The qualitative model is translated into one internal representation based on the intrinsic equations (3) to (5). The deviation variable is controlled in function on the second term of the equation (5) and the control of the yaw variable is based on the equations (3) and (4) and in the first term of the equation (5).

The guidance system performs three steps in each control cycle: the current values of the deviation and yaw variables are measured by the sensors boarded in the vehicle, the fuzzy controller is executed, the outputs control are applied to the vehicle actuators.

Fig. 3. Graphical representations of the two linguistic variables involved in the qualitative model of the direction control system
5 Implementation

The automatic guiding system consists essentially in two fuzzy controllers dealing with the steering wheel and the throttle separately. From a functional point of view both controllers are implemented in a fuzzy coprocessor. In true, the implementation is accomplished in a conventional processor at two levels: the high level contains the qualitative model of the vehicle control problem, in other words, the knowledge base of the concrete control problem written with sentences close to the natural language. The low level contains the internal representation, in other words, the typical procedures to implement the conventional fuzzy operations (fuzzification, inference, defuzzification, etc.) and the intrinsic movement equations.

Fig. 4. To the left realistic map of the circulation zone. To the right a real route superposed to the route of reference

6 Experiments and Results

The performances of the automated driver are similar to the ones of the cruise control systems and lateral control systems included in some prototype of vehicles and some commercial vehicles [1] [5]. Real experiments has been done with two commercial Citroën Berlingo vans that have been equipped with a DGPS, an industrial computer, a communication system and the electronic actuators to move the steering wheel, the throttle and the brake. The vans move along an experimental circuit of about 1 Km. of total length. At the beginning of each experiment the route to be followed is fixed in a simbolic way, in true a sequence of street names, the system maps this simbolic route in a reference route formed by a chain of straitgh segments.

Speed control results. The results obtained show an error less than 0.5 km/h when a constant speed has to be maintained.
Direction control results. The system is able of tracking straight segments at speed up to 65 km/h, which is the maximum speed the vehicle can reach in the longest path (about 250 meters) of the experimental circuit. The greatest separation of the route of reference is 25 cm. Besides straight angle and closer curves (Fig. 4) are tracked at a maximum speed of 6 km/h.

References


Acknowledgment. This research has been founded by The Spanish Agency CICYT under the grant TAP98-0813.
Hyper-Automaton System Applied to Geometry Demonstration Environment

Júlio P. Machado\textsuperscript{1}, Márcia R. Notare\textsuperscript{1}, Simone A. Costa\textsuperscript{2}, Tiarajú A. Diverio\textsuperscript{1}, and P. Blauth Menezes\textsuperscript{1}

\textsuperscript{1} Instituto de Informática, UFRGS
Caixa Postal 15064, CEP 91501-970, Porto Alegre - RS - Brasil
\{jhapm, notare, diverio, blauth\}@inf.ufrgs.br
\textsuperscript{2} Centro de Ciências Exatas e Tecnológicas, Unisinos
Av Unisinos 950, CEP 93022-000, São Leopoldo - RS - Brasil
sac@exatas.unisinos.br

Abstract. This paper describes the conception and implementation of a learning system on Euclidean Geometry demonstrations and its knowledge base. We use the formalism of finite automata with output to represent and ordain the statements that constitute a geometric demonstration in the knowledge base. The system is built on the MOSCA learning protocol, based on learning with the assistance of examples and interaction among five agents (Mestre, Oráculo, Sonda, Cliente and Aprendiz) involved in the learning process. We briefly revise the Hyper-Automaton concept as a structural model for hypertext and its use as the basis for the central core of the agents in a learning system is analyzed.

1 Introduction

This paper describes the conception and implementation of the knowledge base of a learning system being applied in the Euclidean Geometry context, more specifically to the geometric demonstrations developed by Euclid \cite{5}, using the formalism of finite automata with output \cite{4} to represent and ordain the statements that constitute a demonstration. The knowledge base is composed of several automata, where the nodes represent the stages of the demonstration and the arcs represent definitions, axioms and postulates that allow the transitions. For the development of this proposal we chose the first of the thirteen books written by Euclid in Elements. This book approaches the Fundamentals of Plain Geometry and is composed by 23 definitions, 5 postulates, 5 axioms and 48 propositions (34 theorems and 14 problems)\cite{5}.

An automaton can be represented by a graph structure, which may be manipulated in several ways \cite{3}. Formal semantic graphs may be used to provide a programming interface for the control of hypermedia materials \cite{2}, and have been used to good effect in computer-aided instruction and other fields where control over the traversal of a graph is important.

In this sense, this work extends the use of the Hyper-Automaton system, which was designed to manage hyperdocuments in the Web \cite{6,7,9,10}, to a
model for the knowledge base of an agent-based learning system in Geometry demonstrations.

Initially, the concept of the Hyper-Automaton for hyperdocuments structured as automata is presented and we briefly discuss its use in hypermedia systems. Next, we introduce the multi-agent learning environment on Euclidean Geometry. Then, we define the protocol for the agents communication and the knowledge base built over the Hyper-Automaton. Also we present, in detail, an example depicting the information stored in the knowledge base and a possible interaction scenario.

2 Hyper-Automaton

Hyper-Automaton is a system and model which main purpose is to augment the WWW with a hypermedia service based on the application of concepts inherent in Computing Science, specially Automata Theory, Category Theory and Hyperdocuments technology. The system development was aimed at supporting the control of hyperdocuments, mainly for Web-based courses. The system itself consists of three main components: an automata authoring tool, a navigation interface, and a server.

The formal model of hypertext proposed was based on a Finite Automaton with Output representation of a hyperbase structure. It takes advantage of the fact that automata not only capture the descriptive power of Directed Graphs, known to be a useful abstraction in hypertext systems, but provide as well a mathematically precise abstract machine for control and analysis of hypertext execution or browsing and is also an universally known formalism. In this sense, automata give us a more computational model for hypermedia technology then directed graphs.

For completeness of our discussion we first provide a short set of definitions for the Hyper-Automaton. The notation style is that commonly used in Automata Theory, similar to [4].

A Deterministic Finite Automaton or Finite Automaton is defined as a 5-tuple $M = (Σ, Q, δ, q_0, F)$ where:
- $Σ$ input alphabet, set of symbols from which we may build up words/strings suitable for feeding into the automaton;
- $Q$ finite set of possible states;
- $δ : Q × Σ → Q$ next-state function, which is a partial function that determines a new state for the automaton based on the current state and the input symbol read;
- $q_0$ initial state, it belongs to $Q$;
- $F$ set of final states, $F$ is a subset of $Q$.

The Mealy Machine is represented by a 6-tuple $M = (Σ, Q, δ, q_0, F, Δ)$ where:
- $Σ$ input alphabet;
- $Q$ finite set of possible states;
\[ \delta : Q \times \Sigma \rightarrow Q \times \Delta^* \] next-state function with assigned output, which is a partial function that determines a new state for the automaton and an output based on the current state and the input symbol read;

- \( q_0 \) initial state;
- \( F \) set of final states;
- \( \Delta \) output alphabet.

The Moore Machine is analogous and is represented as a 7-tuple \( M = (\Sigma, Q, \delta, q_0, F, \Delta, \delta_s) \) where:

- \( \Sigma \) input alphabet;
- \( Q \) finite set of possible states;
- \( \delta : Q \times \Sigma \rightarrow Q \) next-state function, which is a partial function that determines a new state for the automaton based on the current state and the input symbol read;
- \( q_0 \) initial state;
- \( F \) set of final states;
- \( \Delta \) output alphabet;
- \( \delta_s : Q \rightarrow \Delta^* \) next-output function, which is a total function that assigns an output word to the automaton states.

The visual interface of the browser environment provides the user with a tangible interpretation of the Mealy/Moore Machines represented by the Hyper-Automaton.

The automaton n-tuples have counterparts in the structure of hyperdocuments in the Web. The output alphabet \( \Delta \) is annotated with units of information (hypermedia HTML pages) and, in that case, the result of the next-output function \( \delta_s \) (Moore) or the next-state function \( \delta \) (Mealy) is the display of documents (output words \( \Delta^* \)) in the browser window. The input alphabet \( \Sigma \) that labels the transitions between states in the automaton are displayed as links that can be selected. The link itself is the projection of the next-state function \( \delta \) in the hypertext environment. If a link is followed, then the current displayed contents are deactivated and the contents mapped to the output nodes (Moore) or transitions (Mealy) are activated, in accordance to the transition executed. One important note is that since we are dealing with hypermedia systems in the Web, the concept of initial and final states becomes a little fuzzy, or in a better sense it becomes an external nondeterminism. All states in the Hyper-Automaton are possibly initial or final. In other words, we may initiate the browsing process from any state and we may stop browsing in any state either.

With the use of Finite Automaton the links are implemented as transition functions and are stored in a matrix representing the source state and destination state, and they are not embedded in the HTML code. Such structure constitutes what is defined as external links and has some advantages: the linked files themselves are not changed by creating hypertext references between them, any file can be edited without altering the linking structure, and, in terms of reuse of hypermedia materials once there is no hard-coded links in the pages it is a straightforward procedure.
For an in depth discussion of the Hyper-Automaton and its relationship with hypermedia, including non-determinism, adaptation mechanisms, modelling of on-line adaptive exercises, and operations for hyperdocuments composition see [7][10][9][6].

3 Learning Environment on Euclidean Geometry

The Learning Environment on Euclidean Geometry - LEEG - is an Euclidean Deductive Geometry’s learning environment and its objective is helping in the process of demonstration construction of the Euclidean Plain Geometry.

The system proposes the learning of geometric demonstrations with the assistance of examples or counter-examples, that characterize messages from the system relative inconsistent statements or incorrect terms of Euclidean Geometry.

It follows a multi-agent architecture composed of five different agents: Mestre, Oráculo, Sonda, Cliente, Aprendiz. In a few words, the system works as follows.

Propositions of the Euclid’s first book are submitted to Aprendiz to be demonstrated. The demonstrations developed by Aprendiz are constructed on a table Statement/Deduction Rule (see Table 1), in which every step produced by Aprendiz is attended by the system (Mestre agent). The system sends hints to Aprendiz when it identifies incorrectness in the demonstration building process. The messages sent by the system (Oráculo and Sonda) to Aprendiz have the objective to alert about wrong deduction steps or to incentive the continuity of the demonstration. With a stored representation of the automaton knowledge base, it is possible to manage the execution of the individual users’ tracks through the analysis of the automaton paths. It is also possible to the users resume their interaction with the system. The users can continue from the point they left during the previous interaction with the learning environment. All we need to do is saving the automaton execution state.

Thus, the user’s learning process on Euclidean Geometry is entirely assisted by the LEEG system and its artificial agents (Mestre, Oráculo and Sonda). This three agents have the purpose of accompanying and helping the user’s (Aprendiz) knowledge building. The interaction among the several agents is presented in the next section.

Our focus is on using the Hyper-Automaton for organizing information resources to engage students in comprehending, interpreting, and evaluating materials, providing knowledge building and critical thinking.

4 Modelling the Interaction

The learning system for deductions in Geometry is composed of five agents, which are proposed in the learning protocol called MOSCA - Mestre, Oráculo, Sonda, Cliente, Aprendiz - in Portuguese (in English, respectively, Master, Oracle, Probe, Client, Apprentice). It was developed by Reitz [12] and is being used
to mould the interaction among the agents involved in the learning process. Briefly, the agents interact among each other in the following way:

- **Aprendiz**: receives a proposition from **Cliente** to be demonstrated. A proposition is composed by a set of hypotheses and a thesis. The hypotheses of the proposition are true statements, which can and must be used in its demonstration. The thesis, however, is the statement that must be proved, through a logic sequence of statements formed by the evident or already demonstrated assertions. The demonstration is deductive axiomatic. It starts from the hypothesis to construct an ordering of statements, which should be equivalent to the structure of the knowledge base. The arcs between the stages of the demonstration are established by analysis of examples received from **Oráculo** and **Sonda** agents.

- **Oráculo**: in agreement with the Mestre’s signaling, the agent interacts with Aprendiz through irrefutable examples, with the purpose of helping Aprendiz to establish the arcs between the stages of the demonstration, formulate and synthesize concepts and relationships between the statements. The examples are stored in a database, which is organized according to the arcs of the knowledge base.

- **Mestre**: constantly access the Aprendiz’s construction of the demonstration, checking its structure. The verification of the learning is done by comparison between the Aprendiz and environment’s knowledge base. This comparison is made step by step, for each new statement included by Aprendiz in the demonstration. When exists an equivalence, we say that the learning is put into effect. Aprendiz is stimulated and Oráculo is signalled to change to the new stage.

- **Sonda**: its function is to send examples to Aprendiz, but with refutable solution. The counter-examples can be sent when Mestre perceives that Aprendiz is concluding erroneously an arc in its knowledge structure. Aprendiz must reflect and argue with Mestre to establish the correct arcs.

- **Cliente**: starts the process through submission of a proposition to Aprendiz. It has two ways to access the proposition base. The first one consists of an inductive reasoning: it does not submit propositions that require results not demonstrated yet. The second approach concerns a recursive reasoning: propositions that make use of other propositions that were not yet proved by Aprendiz can be submited.

In the LEEG, the agents Mestre, Oráculo and Sonda are artificial and the agents Aprendiz and Cliente are human. The Fig. ?? shows the scheme of communication of the five agents of LEEG.

The system’s knowledge base is the complete set of all possible statements that would be used by Aprendiz in every proposition demonstration. Every proposition demonstration is developed at the system’s knowledge base and obey the logic sequence of the statements and deduction rules that will be reproduced by Aprendiz. In the LEEG, the knowledge base is composed by 23 definitions, 5 axioms, 5 postulates and 48 propositions demonstrated.
5 Modelling the Knowledge Base

The Euclidean Geometry is a classical example of axiomatic system and its formation structure is deducted from a set of basic premises that descend the others proceeding propositions [11].

Being a logic proposition a sequence of symbols built over well-defined (basic) rules, we define the set $Prop$ as the least set of logical propositions with properties admitted in the construction rules. A sequence $\varphi_0, \ldots, \varphi_n$ is a formation sequence of a logical proposition $\varphi$ if $\varphi_n = \varphi$ and for all $i \leq n$, $\varphi_i$ is uniquely determined by the previous propositions, according to the basic construction rules of a logical proposition. A derivation $\Gamma \vdash \varphi$, i.e., there is a derivation with conclusion $\varphi$ and all the hypotheses in $\Gamma$.

The terms involved in a deductive system are the following [11]:

1. Definition: an assertion that only requires a comprehension of the terms applied;
2. Postulates: principles or facts acknowledged but not demonstrated and admitted without demonstration;
3. Axioms: evident propositions and not subject to demonstration;
4. Propositions: object’s property assertions (theorems) or steps or its construction steps (problems) that must be subject to demonstration.
In other words, the definitions, postulates and axioms compose the evident statement set of a deductive axiomatic system, which are acknowledged as true with no need to prove it. The propositions, however, must be proved based on these statements. When a proposition is demonstrated, we can acknowledge it as being true, and use it for the demonstration of other previous propositions, that is, it starts to compose the set of evident statements.

A theorem is a statement of the type \( p \rightarrow q \) that proves to be true ever. The propositions \( p \) and \( q \) are denominated, respectively, hypothesis and thesis of the theorem. This way, a theorem’s demonstration starts with the hypothesis and through application of axioms, definitions and postulates, proves the thesis. This means that, given a theorem, it is fundamental, before starting its demonstration, to identify the hypothesis and the thesis. A demonstration will be a set of statements withdrawn from the definitions, postulates and axioms and the propositions already demonstrated, besides the hypothesis of the proposition, strictly structured and ordained in an hierarchical order. The structure and the demonstration hierarchy must be obeyed so that we can effectively have a deductive axiomatic system.

The deduction process is then a system over a set of propositions, which must be proved. The proof implies a logic sequence over the set of assertions, that corresponds to a derivation tree containing all possible sub-trees that leads to the desired thesis. Through the logic eyes, a proof would be a derivation tree in which the derivation rules are determined by the statements (basic axioms, postulates, definitions, and propositions already proved), and the sub-trees represent the intermediate paths of deduction. In fact, Godel proved that first order theories with equality is not complete. Similarly, the second order predicate calculus is not complete. As we can see in [3] even though higher order logic does not give us completeness, we can still do formal reasoning to prove the validity of many higher order well formed formulas. Hein [3] presents a familiar example in Euclidean Geometry to see how higher order logic comes into play when we discuss elementary geometry.

Such structure can also be seen as an automaton representing all possible deduction paths that leads to the correct thesis. In this sense, the use and interpretation of the Hyper-Automaton as the main knowledge base for the reasoning on Geometry is quite similar to the use in hyperdocuments. We will return to this point later.

The construction of a deductive demonstration can be associated to two representations: graph and text. The Fig. ?? and the Table ?? shows the demonstration of the first Euclid’s proposition, constructed over graph an textual representation, respectively. In the following, we present the process of building the knowledge base for the “Proposition 1: To construct an equilateral triangle on a given finite straight line.”.

In the graph representation, the bold faced nodes (first and last) correspond, respectively, to hypothesis and thesis of the proposition and must be identified in its enunciation. In the proposition demonstration, we start with the hypothesis (accepted as true) and, through a logic sequence of statements, we proved
the thesis to be true. Every statement included in Table 1 is strictly justified by axioms, definitions and postulates. The statements that do not present explicit justification are justified by previous statements of the current demonstration.

In the textual representation, the deduction rules accepted by LEEG are the 23 definitions, 5 axioms and 5 postulates of the Euclidean Plain Geometry, applied to the correct elements. We take as example of deduction rule the Postulate 1 ("To draw a straight line from any point to any point.") in order to illustrate the application. This rule makes reference to the construction of a segment from two points. This means that it must be applied to exactly two points. The first and last statements must be, respectively, the hypothesis and thesis of the proposition, as in the graph representation.

In the LEEG’s knowledge base, as previously pointed, the demonstrations structure is implemented through the Hyper-Automaton model, that permits a efficient data organization. The automaton represents all the possible reasoning of deduction that conduct from hypothesis to correct thesis. This means the language accepted by the automaton is a correct deduction in the system. The
set of assertions can be seen as the input alphabet for the automaton, labeling
the transitions. The set of states represents the current status in the reasoning
process: the intermediate objects that are being built to construct the thesis.
The next-state function is represented by links between statements, i.e., by each
justification that permits the transition from a state to other. The initial state
corresponds to proposition’s hypothesis and the final state is the proposition’s
thesis. The output alphabet has two roles inside the system: in the automata
representing all the proposition’s proofs, it is the result of applying an assertion;
in the automata representing incorrect deduction paths, the output corresponds
to critiques depicting the incorrectness. The Fig. 3 shows the proposition 1 struc-
tured as a automaton.

We perceive, at the automaton’s graph representation, that the demonstra-
tion can be developed by ten different reasoning, i.e., ten paths conduct to the
final state of the automaton. This occurs because the statements that occur
simultaneously in the demonstration in the graph structure, were represented
by all the possible sequential orderings in the automaton. This model permits
considering different reasonings produced by Aprendiz in the learning process,
which are equivalent and accepted as correct by LEEG. The transition function
of the respective automaton is represented in Table 2.

6 Concluding Remarks

By analyzing the structure of the knowledge base we observe the finite automata,
specially the Hyper-Automaton system, and the MOSCA protocol offer a useful
model as the basis for the reasoning system on Geometry. It is also simple to
implement and includes a set of efficient and costless algorithms. Even though
the agent system is not in the Web, there are some advantages in applying the
Hyper-Automaton to the demonstration environment:

---

Table 1. Demonstration of Proposition 1, textual representation

<table>
<thead>
<tr>
<th>Statement</th>
<th>Deduction Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment AB</td>
<td>(Hypothesis)</td>
</tr>
<tr>
<td>circle A, AB</td>
<td>Postulate 3 (A, AB)</td>
</tr>
<tr>
<td>circle B, AB</td>
<td>Postulate 3 (B, AB)</td>
</tr>
<tr>
<td>point C = intersection of two circles</td>
<td>(Statements 2 and 3)</td>
</tr>
<tr>
<td>segment AC</td>
<td>Postulate 1 (A, C)</td>
</tr>
<tr>
<td>segment BC</td>
<td>Postulate 1 (B, C)</td>
</tr>
<tr>
<td>segment AC = segment AB</td>
<td>Definition 15 (AC, AB)</td>
</tr>
<tr>
<td>segment BC = segment AB</td>
<td>Definition 15 (BC, AB)</td>
</tr>
<tr>
<td>segment AC = segment BC</td>
<td>Axiom 1 (AB, AC, BC)</td>
</tr>
<tr>
<td>segment AC = segment AB = segment BC</td>
<td>(Statements 7, 8 and 9)</td>
</tr>
<tr>
<td>triangle ABC = equilateral</td>
<td>Definition 20 (AB, AC, BC)</td>
</tr>
</tbody>
</table>
Fig. 3. Graph of the automaton
Table 2. Transition function of the automaton

<table>
<thead>
<tr>
<th>$q_0$</th>
<th>$q_1$</th>
<th>$q_2$</th>
<th>$q_3$</th>
<th>$q_4$</th>
<th>$q_5$</th>
<th>$q_6$</th>
<th>$q_7$</th>
<th>$q_8$</th>
<th>$q_9$</th>
<th>$q_{10}$</th>
<th>$q_{11}$</th>
<th>$q_{12}$</th>
<th>$q_{13}$</th>
<th>$q_{14}$</th>
<th>$q_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_3, A, B$</td>
<td>$P_3, A, B$</td>
<td>$C_1, C_2$</td>
<td>$P_1, A, C$</td>
<td>$P_1, B, C$</td>
<td>$D_15, A, B, C$</td>
<td>$D_15, A, B, C$</td>
<td>$A_1, A, B$</td>
<td>$A_1, A, B$</td>
<td>$A_1, A, B$</td>
<td>$D_{20}, A, B$</td>
<td>$A_1, A, B$</td>
<td>$A_1, A, B$</td>
<td>$A_1, A, B$</td>
<td>$A_1, A, B$</td>
<td>$f$</td>
</tr>
</tbody>
</table>

- Possibility of representing the knowledge using finite automata with output, which is useful for the design of the Mestre agent.
- Verification of the learning process through comparison of automata substructures.
- Modeling of the interaction between the agents Sonda/Oráculo and Aprendiz by the use of links that activate examples and counter-examples.
- Modeling propositions as transitions (links in the Hyper-Automaton) between hypothesis and thesis. In case a certain proposition has not yet being demonstrated, its associated transition becomes unavailable until further demonstration.
- They allow a certain computation redundancy that models the different ways of reasoning. As a consequence, we are able to study the Aprendiz agent reasoning process in its diversity in the demonstration of propositions. Such diversity may be modeled as concurrent events using formalisms for concurrency (as CCS), which is planned for future works.

References

   URL http://aleph0.clarku.edu/~djoyce/java/elements/elements.html
The PCS Prover
in TH\existsOREM\forall

Bruno Buchberger
Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria
Buchberger@RISC.Uni-Linz.ac.at

Acknowledgement: This paper was written in the frame of the Project "Scientific Computing" sponsored by the Austrian National Science Foundation (FWF), project number SFB 1302. I would also like to thank M. Rosenkranz for helping me in the preparation of this manuscript.

Abstract: In this paper, we present a new heuristic proving method for predicate logic, called the PCS method since it proceeds by cycling through various phases of proving (i.e. applying generic inference rules), computing (i.e. simplifying formulae), and solving (i.e. finding witness terms). Although not a complete proving calculus, it does produce very natural proofs for many propositions in elementary analysis like the limit theorems. Thus it appears to be a valuable contribution for many of the routine proofs encountered in exploring mathematical theorems.

1 Introduction

In this paper, we present the main ideas of a new, heuristic, proving method for predicate logic called the PCS method (Proving-Computing-Solving method). The method is particularly suited for proving theorems in theories whose main notions are defined by formulae with alternating quantifiers, i.e. formulae of the form \( \forall \exists \forall \ldots \). A typical example of such a notion is the notion of limit:

\[
\lim_{x \to a} f(x) \iff \forall \epsilon > 0 \exists N \forall n \geq N \forall x \in \mathbb{N} \quad |f(n) - a| < \epsilon
\]

The main emphasis of the PCS method is naturalness, i.e. the method imitates human proof style and generates proofs that are easy to understand. Also, in the cases the method works, it normally finds the proof with very little search.

In contrast to the resolution method and other well-known methods for automated theorem proving in predicate logic, the PCS is not complete and will fail in many cases. However, we believe that, for the acceptance of theorem proving as a tool for practical theorem proving, it is important to come up with special proof methods that deliver natural proofs in short time for the many nontrivial but not too difficult theorems that
occur in the usual exploration of mathematical theories. For this objective it seems that
the PCS can make a useful contribution. In fact, proving the propositions about the
elementary theory of analysis, e.g. propositions about the notion of limit, for general
predicate logic theorem provers still is a hard problem and, thus, we believe that the
PCS method is a decisive step forward.

Essentially, the PCS method, in a natural way, reduces proving to solving. In the
case of analysis, proofs are reduced to solving constraints over the real numbers.
Fortunately, by the work of Collins and others, see [4] and [5], there exist complete
algorithms for solving the most general class of constraints over real numbers. Thus, in
the cases we manage to reduce the proof of a theorem to constraint solving over the
reals, the proof can be established. In fact, we will see that this reduction by the PCS
method is "natural" and that the solution of the constraints is an "uninteresting" step
whose details people do not want to see when they are exploring analysis because, at
that stage, they already master the theory of real numbers and would like to concentrate
on the exploration of the notions of analysis like limit, derivative, etc. Thus, it is
methodologically appropriate to call constraint solvers as "black boxes" at this stage.

2 THEOREMA

The Theorema system is a software system that aims at automating proving in a
uniform logic and software frame for formal mathematics. It is programmed in Mathe-
matica and, hence, is available on all platforms on which Mathematica is available.

However, this does not entail that, when doing proofs in Theorema, any of the
implicit knowledge of Mathematica is used. All knowledge that is used in Theorema
proofs can be stated explicity. However, we also have means to state explicitly that
well-defined sections of Mathematica knowledge, i.e. algorithms for many mathemati-
cal functions, can be used in proofs. This gives maximum freedom for the user to
"believe" in the correctness of Mathematica knowledge or not.

Theorema is a multi-method system, i.e. we do not attempt to generate proofs in all
areas of mathematics with just one general predicate logic proving method. In fact, we
believe that having only one proof method for all of mathematics, although theoreti-
cally possible, is not practical. Thus, in Theorema, we provide a library of general and
special provers together with general and special solvers and general and special
simplifiers.

In Theorema, we emphasize the importance of readable proofs and nice output.
Thus, we do not only generate abstract proof objects but we also provide
post-processors that transform the abstract proof objects into proof text that can easily
be read by humans.

The Theorema system is based on research of the author in the area of computer
algebra, formal mathematics, and didactics since 1975 and is now a joint effort of the
Theorema Working Group directed by the author since 1996, see www.theorema.org.
More details about Theorema can be found in [2, 3].
3 Mathematical Texts in THEOREMA

Before we go into the details of the PCS method, we present an example formal text in Theorema which will later be used for demonstrating the method. We start with the definition of the notion of limit:

\[
\text{Definition} \quad \text{"limit": any } f, a, \quad \text{limit}(f, a) \iff \forall \varepsilon > 0 \exists N \in \mathbb{N} \forall n \geq N \quad |f[n] - a| < \varepsilon.
\]

The actual definition is the predicate logic formula

\[
\text{limit}(f, a) \iff \forall \varepsilon > 0 \exists N \in \mathbb{N} \forall n \geq N \quad |f[n] - a| < \varepsilon.
\]

that should be self-explanatory. The Theorema notation for formulae is close to the usual notation in mathematical textbooks. The only exception is that we use brackets instead of parentheses for function application, i.e. \(f[n]\) is the term with function symbol 'f' and argument term 'n'. The use of brackets instead of parentheses is taken over from Mathematica because, in fact, parentheses are ambiguous: For example, 'f(n+m)' could be understood as both 'f[m+n]' and 'f.(m+n)'. The 'any[f,a]' declares 'f' and 'a' as free variables. All identifiers (and function and predicate symbols) that are neither declared as free variables nor bound by a quantifier are considered to be constants. Note that, in the above example formula, 'f' is a higher-order variable: It occurs at the position of a function symbol in the term 'f[n]'.

The keyword 'Definition' and the label "limit" have no logical meaning. The are only used for easy reference: As soon as the above definition is entered into an input cell of Mathematica (after having loaded the Theorema system on top of Mathematica) one can refer to the entire definition by just 'Definition["limit"]', for example when building up theories (see below) or when referring to the use of definitions in proofs.

Now let us formulate an easy proposition on the notion of limit in the notation of Theorema:

\[
\text{Proposition} \quad \text{"limit of sum": any } f, a, g, b, \quad \text{limit}(f, a) \land \text{limit}(g, b) \implies \text{limit}(f + g, a + b).
\]

We will show later how a proof of this proposition can be generated automatically by the PCS prover of Theorema. Before we attempt to do this we must, of course, provide some knowledge on the notions +, −, <, etc. occurring in the definition of the notion of limit. First, we need the definition of + on sequences:

\[
\text{Definition} \quad \text{"+": any } f, g, x, \quad (f + g)[x] = f[x] + g[x].
\]

Also, we need a version of the "triangle inequality":

\[
\text{Lemma} \quad \text{"+\|": any } x, y, a, b, \delta, \varepsilon, \quad \left| (x + y) - (a + b) \right| < (\delta + \varepsilon) \iff \left( |x - a| < \delta \land |y - b| < \varepsilon \right)
\]
Finally, we will need some knowledge on the maximum function:

\[
\text{Lemma}[\text{"max"}, \text{any}[m, M_1, M_2], \\
m \geq \max[M_1, M_2] \Rightarrow (m \geq M_1 \land m \geq M_2)]
\]  

In this paper, we do not discuss the interesting question of how one knows which knowledge is appropriate for proving a given theorem. In fact, playing with a system like Theorema gives a lot of insight into the mechanism of how to "explore theories" instead of just "proving isolated theorems", see some ideas on this question in [1].

Now we can combine the individual formulae above in one knowledge base by the Theorema construct "Theory".

\[
\text{Theory}[\text{"limit"}, \\
\text{Definition}[\text{"limit:"}] \\
\text{Definition}[\text{"+:"]] \\
\text{Lemma}[\text{"+"}] \\
\text{Lemma}[\text{"max"]}
\]  

In fact, the 'Theory' construct can be applied recursively, i.e. one can build up hierarchically structured theories in Theorema and refer to them by a single label.

4 The PCS Proving Method

4.1 An Overview on the PCS Method

The PCS proof method was established by the author in 2000 and aims at generating "natural" proofs. In fact, the PCS method basically is a formalization of a heuristic method the author has been teaching for many years in his "Thinking, Speaking, Writing" course as a practical proof technique for humans.

Roughly, the PCS method proceeds by iteratively going through the following three phases:

- the P-phase ("Proving" phase)
- the C-phase ("Computing" phase)
- the S-phase ("Solving" phase)

In the P-phase, a couple of predicate logic rules are applied in the "natural deduction" style in order to decompose the proof problem into a couple of more elementary proofs. In the C-phase, definitions (and other equalities and equivalences) and implications are used in a "rewrite" (symbolic computation) style in order to reduce proof goals and to expand knowledge bases. By the P-phase and the C-phase, one arrives at proof situations in which the goals have the form of existentially quantified formulae, i.e. one
has to "find" terms that satisfy the conditions specified in the goals. In this moment, the proof can often be completed by calling algorithmic solvers for certain special theories, for example, the theory of real numbers. Hence, the PCS method brings together theorem proving with algebraic algorithms.

A first implementation of the PCS method, within *Theorema*, was sketched and tested by the author in 1999 and was then worked out in detail in the PhD thesis [6]. An implementation of the PCS method for the special case of set theory will be presented in the PhD thesis [7].

### 4.2 A Proof Generated by the PCS Prover

A proof of the above proposition can be found completely automatically, by entering the following *Theorema* call

Prove[Proposition["limit of sum"], using → Theory["limit"], by → PCS]. (9)

Below, we show the proof exactly as generated by the system. One can explain the essential ingredients of the method most easily by going through the individual steps of this example proof.

Prove:

(Proposition (limit of sum)) \( \forall f, a, g, b \) \( (\text{limit}[f, a] \land \text{limit}[g, b] \Rightarrow \text{limit}[f + g, a + b]) \),

under the assumptions:

(Definition (limit:)) \( \forall f, a \left( \text{limit}[f, a] \Leftrightarrow \forall \epsilon \in N \exists n \in N \exists \frac{n}{\epsilon} \left( |f[n] - a| < \epsilon \right) \right) \).

(Definition (+:)) \( \forall f, g, a \left( (f + g)[x] = f[x] + g[x] \right) \),

(Lemma (|+|)) \( \forall x, y, a, b, \delta, \epsilon \left( |(x + y) - (a + b)| < \delta + \epsilon \Leftrightarrow (|x - a| < \delta \land |y - b| < \epsilon) \right) \),

(Lemma (max)) \( \forall m, M_1, M_2 \left( m \geq \text{max}[M_1, M_2] \Rightarrow m \geq M_1 \land m \geq M_2 \right) \).

We assume

(1) \( \text{limit}[f_0, a_0] \land \text{limit}[g_0, b_0] \),

and show

(2) \( \text{limit}[f_0 + g_0, a_0 + b_0] \).

Formula (1.1), by (Definition (limit:)), implies:

(3) \( \forall \epsilon \in N \exists n \in N \exists \frac{n}{\epsilon} \left( |f_0[n] - a_0| < \epsilon \right) \).

By (3), we can take an appropriate Skolem function such that

(4) \( \forall \epsilon \in N \exists n \in N \exists \frac{n}{\epsilon} |f_0[n] - a_0| < \epsilon \).

Formula (1.2), by (Definition (limit:)), implies:

(5) \( \forall \epsilon \in N \exists n \in N \exists \frac{n}{\epsilon} |g_0[n] - b_0| < \epsilon \).

By (5), we can take an appropriate Skolem function such that
\[
\forall \varepsilon \geq 0 \forall n \geq N_1[\varepsilon] (|g_0[n] - b_0| < \varepsilon).
\]

Formula (2), using (Definition (limit:)), is implied by:
\[
\forall \varepsilon \geq 0 \forall n \geq N (f_0 + g_0)[n] - (a_0 + b_0) < \varepsilon).
\]

We assume
\[
\varepsilon_0 > 0,
\]
and show
\[
\exists N \forall n \geq N (f_0 + g_0)[n] - (a_0 + b_0) < \varepsilon_0).
\]

We have to find \( N_2 \) such that
\[
\forall n \geq N_2 \Rightarrow |(f_0 + g_0)[n] - (a_0 + b_0)| < \varepsilon_0).
\]

Formula (10), using (Definition (+)), is implied by:
\[
\forall n \geq N_2 \Rightarrow |f_0[n] + g_0[n]) - (a_0 + b_0)| < \varepsilon_0).
\]

Formula (11), using (Lemma (|+|)), is implied by:
\[
\exists \delta, \varepsilon \forall n \geq N_2 \Rightarrow |f_0[n] - a_0| < \delta \land |g_0[n] - b_0| < \varepsilon).
\]

We have to find \( \delta_0^*, \varepsilon_1^* \) and \( N_2^* \) such that
\[
\delta_0^* + \varepsilon_1^* = \varepsilon_0 \land \forall n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta_0^* \land |g_0[n] - b_0| < \varepsilon_1^*.
\]

Formula (13), using (6), is implied by:
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \forall n \geq N_2^* \Rightarrow \varepsilon_1^* > 0 \land n \geq \max(N_0[\delta_0^*], N_1[\varepsilon_1^*]),
\]

which, using (4), is implied by:
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \forall n \geq N_2^* \Rightarrow \delta_0^* > 0 \land \varepsilon_1^* > 0 \land n \geq \max(N_0[\delta_0^*], N_1[\varepsilon_1^*]),
\]

which, using (Lemma (max)), is implied by:
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \forall n \geq N_2^* \Rightarrow \delta_0^* > 0 \land \varepsilon_1^* > 0 \land n \geq \max(N_0[\delta_0^*], N_1[\varepsilon_1^*]).
\]

Formula (14) is implied by
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \forall n \geq N_2^* \Rightarrow \delta_0^* > 0 \land \varepsilon_1^* > 0 \land n \geq \max(N_0[\delta_0^*], N_1[\varepsilon_1^*]).
\]

Partially solving it, formula (15) is implied by
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \delta_0^* > 0 \land \varepsilon_1^* > 0 \land (N_2^* = \max(N_0[\delta_0^*], N_1[\varepsilon_1^*])).
\]

Now,
\[
(\delta_0^* + \varepsilon_1^* = \varepsilon_0) \land \delta_0^* > 0 \land \varepsilon_1^* > 0
\]
can be solved for \( \delta_0^* \) and \( \varepsilon_1^* \) by a call to Collins cad-method yielding the solution
\[
0 < \delta_0^* < \varepsilon_0,
\]
\[
\varepsilon_1^* \leftarrow \varepsilon_0 + 1 \cdot \delta_0^*.
\]

Let us take
\[
N_2^* \leftarrow \max(N_0[\delta_0^*], N_1[\varepsilon_0 + 1 \cdot \delta_0^*]).
\]

Formula (16) is solved. Hence, we are done.
4.3 The Essential Ideas of the PCS Method

Taking the above PCS-generated proof as an example, we now describe the essential steps of the PCS method in more detail:

The proof starts by echoing the proposition to be proved. In our example, this is the proposition with label (Proposition (limit of sum)). Then we echo the formulae in the initial knowledge base. In our example, these are the formulae with labels (Definition (limit:)), (Definition (+:)), (Lemma (|+|)), and (Lemma (max)).

**P-phase:** Now we start with a phase in which the "natural deduction" rules of predicate logic, except the ones for equalities, equivalences, and implications, are applied to the proof goal and the knowledge. By doing this, the given proof situation is reduced to one or more other, simpler, proof situations. In the above example, the P-phase produces formulae (1) and (2) by applying the "arbitrary but fixed" rule and the deduction rule of predicate logic.

**C-phase:** Now we try to use "rewrite knowledge" (equivalences, equalities, and implications in the knowledge base) in the "rewrite" style, i.e. we replace, in the goal formula and in formulae of the knowledge base, appropriate instances of the left-hand sides of the rewrite knowledge by the corresponding instances of the right-hand sides.

Note that, by doing so, goals are reduced to other goals that imply the given goals whereas formulae in the knowledge base are expanded to other formulæ in the knowledge base that are implied by the given knowledge. In our example, formulae (3) and (5) are generated from (1) by C-phase steps using (Definition (limit:)) as rewrite knowledge.

**P-phase with Skolemization:** Now we may be back in a P-phase, i.e. a phase in which natural deduction steps can be applied. In this phase, we apply, in addition to the usual natural deduction rules of predicate logic, Skolemization, i.e. for formulæ of the form \( \forall x \exists y \, F[x, y] \) in the knowledge base we introduce new function constants ("Skolem" function constants) and assert \( \forall x \, F[x, f[x]] \). This step is crucial for having the possibility in the later S-phase to construct solving terms for existentially quantified formulæ in an explicit way. In our example, formulæ (4) and (6) are derived from formulæ (3) and (5), respectively, by Skolemization.

**C-phase:** Now we may again be in a C-phase in which rewrite knowledge is applicable in rewrite style. In our example, (7) is now obtained from (2) by using again (Definition (limit:)) as a rewrite rule.

**P-phase:** Now again a P-phase brings us to the additional assumption (8) and the new goal (9).

**S-phase:** Now the goal is an existentially quantified formula and we must start "solving", i.e. finding an appropriate term that satisfies the condition stated in the goal
formula. We start solving by, first, introducing a "find constant", i.e. a new constant whose value will be determined later as the proof proceeds. We use constants with asterisks for this purpose. Introducing these constants is important in order to be able to decompose goals further, i.e. to work inside the existentially quantified formula in a couple of alternating P- and C-phases. In our example, we introduce now $N_2^*$ and obtain the new goal (10).

**P- and C-phases:** In our example, by using (Definition (+:)) as a rewrite rule, goal (10) can be reduced to goal (11).

**C-phase with existential rewriting:** Now we are at an important proof situation. Namely, the conclusion of goal (11)

$$\forall (n \geq N_2^* \Rightarrow |(f_0[n] + g_0[n]) - (a_0 + b_0)| < \epsilon_0).$$

is very close to being an instance of the conclusion in (Lemma (|+|)),

$$\forall (x, y, a, b, \delta, \epsilon) ((x + y) - (a + b) < \delta + \epsilon \iff (|x - a| < \delta \land |y - b| < \epsilon)),$$

so that a reduction of the goal by rewriting would be possible. However, '$\epsilon_0$' is a constant and, thus, we cannot find a substitution for '$\delta$' and '$\epsilon$' such that, by this substitution, '$\delta + \epsilon$' would be transformed into '$\epsilon_0$'. For handling this situation, we propose "existential rewriting": We reduce goal (11), by using (Lemma (|+|)), to goal (12);

$$\exists \delta, \epsilon \forall (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta \land |g_0[n] - b_0| < \epsilon).$$

It is easy to prove that this generalized form of rewriting is correct. By existential rewriting, we are able to handle the above proof situation in a natural way on the expense of introducing existential quantifiers in the goal.

**S-phase:** Now we are again in an S-phase, which we handle by introducing extra find constants. In our case '$\delta_0^*$' and '$\epsilon_1^*$' are introduced as new find constants yielding the new goal (13).

**C-phase:** Goal (13) can now be reduced by a couple of rewrite steps, using the Skolemized formulae (6) and (4) and also (Lemma (max)), to formula (14).

**P-phase:** Now P-steps are possible that bring the formulae '$\delta_0^* > 0$' and '$\epsilon_1^* > 0$', which do not contain variable '$n$', outside the scope of the $\forall_n$ quantifier.

**S-phase:** The resulting formula (15) has now the property that it is the conjunction of two independent solve problems, the first one asking to find '$\delta_0^*$' and '$\epsilon_1^*$' in dependence on '$\epsilon_0$' and the second one asking to find $N_2^*$ in dependence on '$\delta_0^*$' and '$\epsilon_1^*$. The second problem can be solved by simple predicate logic rules and yields
as a possible solution. The first problem is a problem that is a constraint solving problem over the real numbers and, hence, can be solved by a call to any complete real number constraint solver. We use Collins’ algorithm for this purpose, which is available in the extended Mathematica library. We obtain a general answer back, namely

\[ 0 < \delta_0 < \epsilon_0, \]  
\[ \epsilon_1 \leftarrow \epsilon_0 - \delta_0. \]

This means that any \( \delta_0 \) satisfying \( 0 < \delta_0 < \epsilon_0 \) is a possible solution and that then \( \epsilon_1 \) must be chosen as \( \epsilon_0 - \delta_0 \). This concludes the proof.

Note that the proof generated by the PCS prover, in addition to showing that the proposition is a consequence of the formulae in the knowledge base, yields interesting information on the convergence of the sum sequence \( f + g \): The solving terms for \( \delta_0, \epsilon_1 \), and \( N_2 \) that are constructed during the proof of the proposition tell us that, given \( \epsilon_0 > 0 \), one can find an index \( N_2 \) such that, from \( N_2 \) on, the elements of the sequence \( f + g \) stay closer than \( \epsilon_0 \) to \( a + b \) by the following procedure:

1. Choose an arbitrary \( \delta_0 \) such that \( 0 < \delta_0 < \epsilon_0 \).
2. Then compute \( \epsilon_1 := \epsilon_0 - \delta_0 \).
3. Finally compute \( N_2 := \max\{N_0[\delta_0], N_1[\epsilon_1]\} \).

Here \( N_0 \) is a procedure by which, given an arbitrary \( \epsilon > 0 \), one can find an index from which on \( f \) stays closer to \( a \) than \( \epsilon \), and, similarly, \( N_1 \) gives an index bound for \( g \). Thus the solving terms constructed in the proof can be viewed as a procedure for the index bound of \( f + g \) with index bounds for \( f \) and \( g \) as "black-box" sub-procedures. In other words, the PCS prover is not only a prover but also a procedure synthesizer. In case one has algorithmic procedures \( N_0 \) and \( N_1 \) for finding the index bounds for \( f \) and \( g \), the procedure synthesizer synthesizes an algorithm for computing the index bound for \( f + g \). Thus, the PCS prover does not only generate proofs but also provides interesting constructive information on the notions involved in the proposition.

5 Conclusion

The PCS prover combines, in a natural way, proving by a restricted set of inference rules, simplifying, and solving. In fact, also other general and special automated provers combine restricted proving, simplifying and solving. For example, proving geometrical theorems by the Gröbner bases method, essentially is also a reduction, by certain proving and simplifying steps, of deciding the truth of formulae to deciding the solvability of certain related sets of algebraic equations. Also, the famous resolution method for general predicate logic proving, is essentially a reduction of proving, by simplifying, to solving certain sets of standard predicate logic formulae, namely clauses.
In a future version of Theorema, the flexible interplay between proving, solving, and simplifying will be our main design feature so that Theorema will appear as a library of built-in provers, solvers, and simplifiers from which the user can build provers, solvers, and simplifiers for the particular given application in an easy, flexible and general way.

References

Higher-Order Lazy Narrowing Calculus: A Solver for Higher-Order Equations

Tetsuo Ida¹, Mircea Marin¹, and Taro Suzuki²

¹ Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan
{mmarin,ida}@score.is.tsukuba.ac.jp
² Department of Computer Software
University of Aizu, Aizu Wakamatsu 965-8580, Japan
taro@u-aizu.ac.jp

Abstract. This paper introduces a higher-order lazy narrowing calculus (HOLN for short) that solves higher-order equations over the domain of simply typed \(\lambda\)-terms. HOLN is an extension and refinement of Prehofer’s higher-order narrowing calculus LN using the techniques developed in the refinement of a first-order lazy narrowing calculus LNC. HOLN is defined to deal with both unoriented and oriented equations. It keeps track of the variables which are to be bound to normalized answers. We discuss the operating principle of HOLN, its main properties, i.e. soundness and completeness, and its further refinements. The solving capability of HOLN is illustrated with an example of program calculation.

1 Introduction

Proving, solving and computing are the essence of mathematicians’ activities. Correspondingly, modern programmers’ role can be thought of as automating proving, solving and computing by defining specifications called programs. Traditionally, computing is the main concern of many programmers, and relatively smaller emphasis has been placed on the other two aspects of our activities. As computer science has become matured and demand for clarity and rigor is ever increasing as information technologies penetrate into our daily life, more and more programmers become concerned with proving and solving.

In this paper, we are concerned with the solving aspect of programming and present a solver which is a computation model for a programming language built upon the notion of equational solving. Let us start with functional programming. In functional programming we are interested in specifying a set \(\mathcal{R}\) of rewrite rules as a program and then compute the normal forms of a term \(t\), if the normal form exists. Formally, the problem statement is to prove the following formula:

\[ \exists s.t \rightarrow^*_{\mathcal{R}} s \text{ and } s \text{ is a normal form}. \]

Usually, \(\mathcal{R}\) is assumed to be confluent, and hence \(s\) is unique.
Proving the above statement is easy since all we have to do is to rewrite the term $t$ repeatedly by applying the rewrite rules in $\mathcal{R}$ until it no longer gets rewritten. So the main concern here is not how to prove the statement, but how to rewrite the term $t$ efficiently to its normal form if the normal form exists. The problem is generalized as follows.

Let $t$ and $t'$ be terms that may contain multiple free occurrences of a variable $X$. Prove $\exists X. t \leftrightarrow_{\mathcal{R}}^* t'$ such that $X$ is a normal form.

Proving an existentially quantified formula by presenting a value that instantiates $X$ is called solving. In particular, when an equality is defined as the reflexive, transitive and symmetric closure of $\rightarrow_{\mathcal{R}}$ as above, we call this equational solving (with respect to $\mathcal{R}$). Solving an equation is significantly difficult since (i) rewriting is not uni-directional, and (ii) we have to find a value for $X$ before we perform rewriting. Indeed, various specialized methods have been developed for solving equations defined over specific domains, e.g. Gaussian elimination for solving a system of linear equations defined over reals.

In this paper we are primarily interested in solving equations over purely syntactic domains consisting of terms of simply typed $\lambda$-calculus. It is a domain of choice when we are reasoning about programs. Therefore the main theme of our paper is to show a method for solving high-order equations.

In the first-order setting where the domain is the Herbrand universe, methods for solving equations called paramodulation and narrowing are known. Narrowing is an $E$-unification procedure ($E$ for Equational theory), and hence it can be naturally specified as a set of inference rules extending the rule-based specification of the unification algorithm \cite{unification}. The inference rules are used to recursively transform an equation into (hopefully) simpler equations.

There are pioneering works on extending narrowing to the higher-order case. A first systematic study of higher-order narrowing appeared in Prehofer’s thesis \cite{Prehofer}. It presents a higher-order lazy narrowing calculus that can be implemented relatively easily. It also has been shown that higher-order lazy narrowing is highly nondeterministic. Whereas various refinements have been developed to reduce the search space for solutions of first-order narrowing, the situation is much more complicated and difficult in the higher-order case.

With these observations in mind, we will first present a higher-order lazy narrowing calculus to be called HOLN in a general setting in order to expose its essential ingredients, as well as to enable further systematic refinements.

The rest of this paper is organized as follows. In Sect. \ref{sect:main} we introduce our main notions and notations. In Sect. \ref{sect:calculus} we define our main calculus HOLN and outline its properties. In Sect. \ref{sect:refinements} we describe the refinements of HOLN towards more deterministic computation. In Sect. \ref{sect:example} we illustrate by an example the solving capabilities of HOLN. Finally, in Sect. \ref{sect:conclusion} we draw some conclusions and outline directions of further research.
2 Preliminaries

We use a slightly modified framework of simply typed λ-terms proposed in [10]. The main ingredients of our framework are:

- the set of all types $T$ generated by a fixed set of base types and the function space constructor $\rightarrow$,
- an algebra $T(F, \mathcal{V})$ of simply typed λ-terms generated from a set $F$ of $T$-typed constants and a set $\mathcal{V}$ of $T$-typed variables. We denote terms by the letters $s, t, l, r, u$ possibly with a subscript. Instead of $\lambda x_1 \ldots \lambda x_n.s$ we write $\lambda \overline{x_n}.s$, where the $x_i$ are assumed to be distinct. Similarly, instead of $(\ldots (s t_1) \ldots) t_n$ we write $s(\overline{t_n})$. The subscript $n$ will be omitted when irrelevant. The set of free variables in a term $t$ is denoted by $\text{vars}(t)$.
- a fully extended pattern rewrite system (EPRS for short) $\mathcal{R}$, which is a finite set of pairs $l \rightarrow r$ such that
  - $l$ and $r$ are λ-terms of the same base type,
  - $\text{vars}(r) \subseteq \text{vars}(l)$,
  - $l$ is of the form $f(\overline{l_n})$, where $f \in F$ and $l_1, \ldots, l_n$ are fully extended patterns. A pattern is a term such that all its free variable occurrences have distinct bound variables as arguments. A fully extended pattern is a pattern such that all its free variable occurrences take as arguments all the variables that are λ-abstracted above its position.
  
  Given an EPRS $\mathcal{R}$, we regard $F$ as a disjoint union $F_d \uplus F_c$, where $F_d = \{ f \in F \mid \exists (f(\overline{l_n}) \rightarrow r) \in \mathcal{R} \}$ is the set of defined symbols, and $F_c = F \setminus F_d$ is the set of constructors.
- equations $e, e_1, e_2, \ldots$, which are pairs of terms of the same type. We distinguish oriented equations denoted by $s \triangleright t$ and unoriented equations denoted by $s \approx t$. A equational goal (goal for short) is a pair $E|_W$ where $E$ is a sequence of equations $e_1, \ldots, e_n$, abbreviated $\overline{e_n}$, and $W$ is a set of free variables. The elements of $W$ are called the solution variables of the given goal.

Unoriented equations are the usual equations used in everyday mathematics, and oriented equations were introduced in our formulation of narrowing to mark equations generated in the process of solving equations. Although the latter can be confined intermediate, we rather give them a first-class status by allowing oriented equations in an initial goal. Having both oriented and unoriented equations as syntactically distinct objects of study gives us more freedom for writing equational programs and also facilitates the understanding of the solving process.

We regard a goal as a pair consisting of a sequence of equations which denotes the existential closure of their logical conjunction, and a set of variables that we want to have bound to $\mathcal{R}$-normalized solutions. The reasons for this notion of goal are (i) that we are only interested in computing $\mathcal{R}$-normalized solutions, and (ii) that it allows us to keep track of the free variables that have to be instantiated to $\mathcal{R}$-normalized terms, as we will see later.

We use the following naming conventions: $X, Y, Z, H$, possibly primed or with a subscript, denote free variables; $x, y, z$, possibly primed or with a subscript,
denote bound variables; and \( v \) denotes a constant or a bound variable. A sequence of syntactic objects \( ob_1, \ldots, ob_n \) where \( n \geq 0 \) is abbreviated \( \overline{ob_n} \).

We identify any \( \lambda \)-term \( t \) with its so called long \( \beta \eta \)-normal form defined by:

\[
t_{\beta}^{\eta} := (t_{\beta})^{\eta},
\]

where \( t_{\beta} \) denotes the \( \beta \)-normal form of \( t \), and \( t^{\eta} \) the \( \eta \)-expanded normal form of \( t \). The transformation of \( t \) to \( t_{\beta}^{\eta} \) is assumed to be implicit. With this convention, every \( \lambda \)-term \( t \) can be uniquely written as \( \lambda x_n.a(s_m) \) where \( a \) is either a constant, bound variable, or free variable. The symbol \( a \) is called the head of \( t \) and is denoted by \( \text{head}(t) \). A term \( t \) is flex if \( \text{head}(t) \in \text{vars}(t) \), and rigid otherwise. To simplify the notation, we will often relax the convention mentioned above and represent terms by their \( \eta \)-normal form.

An EPRS \( R \) induces a rewrite relation \( \rightarrow_R \) as usual. In each step of rewriting we employ an \( \pi \)-lifted rewrite rule instead of a rewrite rule \([10]\). From now on we assume that \( R \) is an EPRS.

The size \( |t| \) of a term \( t \) is the number of symbols occurring in \( t \), not counting \( \lambda \)-binders. In the case of an equation \( e \), its size \( |e| \) is the sum of the sizes of the terms of both sides. For a sequence \( e_n \) of equations, its size is \( \Sigma_{i=1}^{n} |e_i| \).

A substitution is a mapping \( \gamma : V \rightarrow T(\mathcal{F}, V) \) such that its domain \( \text{Dom}(\gamma) \) is finite, where \( \text{Dom}(\gamma) \) is the set \( \{X \in V \mid \gamma(X) \neq X\} \). When \( \text{Dom}(\gamma) = \{X_1, \ldots, X_n\} \), we may write \( \gamma \) as \( \{X_1 \mapsto \gamma(X_1), \ldots, X_n \mapsto \gamma(X_n)\} \). The empty substitution \( \varepsilon \) is the substitution with empty domain. The homomorphic extension of a substitution is defined as usual, and we abuse the notation and use \( \gamma \) for its homomorphic extension. We denote by \( t\gamma \) the image of a term \( t \) via the homomorphic extension of a substitution \( \gamma \).

A substitution \( \gamma \) is \( R \)-normalized iff \( \gamma(X) \) is an \( \rightarrow_R \)-normal form for all \( X \in \text{Dom}(\gamma) \). Given a finite set of free variables \( V \), we define the restriction of \( \gamma \) to \( V \) by \( \gamma|_V(X) = \gamma(X) \) if \( X \in V \), and \( \gamma|_V(X) = X \) if \( X \in V \setminus V \). We define the relation \( \gamma_1 \leq \gamma_2 |_V \) as \( \exists \theta, \forall X \in V \setminus V. X \gamma_2 = X \gamma_1 \theta \).

A substitution \( \gamma \) is a solution of an equation \( e \), notation \( R \vdash e \gamma \), if there exists a rewrite derivation \( D \) of the form (i) \( s \gamma \rightsquigarrow R \ t \gamma \), if \( e = s \triangleright t \), and (ii) \( s \gamma \rightsquigarrow^* R \ t \gamma \), if \( e = s \bowtie t \). Such an \( R \) is called a rewrite proof that \( \gamma \) is a solution of \( e \). \( \gamma \) is a solution of a goal \( e_n|_W \), notation \( \gamma \in \text{Sol}_R(e_n|_W) \), if \( e \gamma \) is an \( R \)-normalized substitution and \( R \vdash e_k \gamma \) for all \( 1 \leq k \leq n \). Given \( \gamma \in \text{Sol}_R(E|_W) \), a rewrite proof of \( \gamma \in \text{Sol}_R(E|_W) \) is a mapping \( \rho \) which maps every equation \( e \) to a rewrite proof that \( \gamma \) is a solution of \( e \). We denote by \( |R| \) the length of a rewrite derivation \( R \).

3 The Calculus HOLN

Now we are ready to formulate our problem.

**Narrowing problem.** Given an EPRS \( R \) and a goal \( E|_W \), find a set \( \text{Ans}_R(E|_W) \subseteq \text{Sol}_R(E|_W) \) such that for any solution \( \gamma \) of \( E|_W \) there exists \( \theta \in \text{Ans}_R(E|_W) \) and \( \theta \leq \gamma |_V \text{vars}(E) \).
HOLN (Higher-Order Lazy Narrowing calculus) is designed to give an answer to this narrowing problem.

### 3.1 Inference Rules of HOLN

HOLN consists of three groups of inference rules: **preunification rules, narrowing rules,** and **removal rules of flex equations.** The inference rules are relations of the form

\[(E_1, e, E_2) \mid W \Rightarrow_{\alpha, \theta} (E_1\theta, E, E_2\theta) \mid W',\]

where \(\alpha\) is the label of the inference rule, \(e\) is the selected equation, \(\theta\) is the substitution computed in this inference step, \(W' = \bigcup_{X \in W} \text{vars}(X\theta)\), and \(E\) is a sequence of equations called the descendants of \(e\). We adopt the following notational conventions: \(s \geq t\) stands for \(s \approx t\) or \(t \approx s\) or \(s \triangleright t\); and \(s \equiv t\) stands for either \(s \approx t\), \(t \approx s\), or \(s \triangleright t\) or \(t \triangleright s\). We assume that the usage of the symbols \(\geq\) and \(\equiv\) in both sides of an inference rule preserves the orientation of equations. Whenever used, \(H, H_1, H_2, \ldots\) are assumed to be fresh variables.

#### Preunification Rules

[i] Imitation.
   If \(g \in \mathcal{F}\) then
   \[(E_1, \lambda x.X(s_m) \not\triangleright= \lambda x.g(\overline{t_n}), E_2) \mid W \Rightarrow_{[i], \theta} (E_1, \lambda x.H_n(s_m) \not\triangleright= \lambda x.t_n, E_2)\theta \mid W',\]
   where \(\theta = \{X \mapsto \lambda y_m.g(H_n(\overline{y}_{m}))\}\).

[p] Projection.
   If \(\lambda x.t\) is rigid then
   \[(E_1, \lambda x.X(s_m) \not\triangleright= \lambda x.t, E_2) \mid W \Rightarrow_{[p], \theta} (E_1, \lambda x.X(s_m) \not\triangleright= \lambda x.t, E_2)\theta \mid W',\]
   where \(\theta = \{X \mapsto \lambda y_m.t(H_n(\overline{y}_{m}))\}\).

[d] Decomposition.
   \[(E_1, \lambda x.v(s_m) \triangleright= \lambda x.v(\overline{t_n}), E_2) \mid W \Rightarrow_{[d], \epsilon} (E_1, \lambda x.s_n \triangleright= \lambda x.t_n, E_2) \mid W',\]

#### Lazy Narrowing Rules

[on] Outermost narrowing at nonvariable position.
   If \(f(\overline{t_n}) \rightarrow r\) is an \(\overline{x}\)-lifted rewrite rule of \(\mathcal{R}\) then
   \[(E_1, \lambda x.f(s_m) \triangleright= \lambda x.t, E_2) \mid W \Rightarrow_{[on], \epsilon} (E_1, \lambda x.s_n \triangleright= \lambda x.t_n, \lambda x.r \triangleright= \lambda x.t, E_2) \mid W',\]

[ov] Outermost narrowing at variable position.
   If \(f(\overline{t_n}) \rightarrow r\) is an \(x\)-lifted rewrite rule of \(\mathcal{R}\) and \([X \not\in W\) then
   \[(E_1, \lambda x.X(s_m) \triangleright= \lambda x.t, E_2) \mid W \Rightarrow_{[ov], \theta} (E_1\theta, \lambda x.H_n(s_m\theta) \triangleright= \lambda x.t_n, \lambda x.r \triangleright= \lambda x.t\theta, E_2\theta) \mid W',\]
   where \(\theta = \{X \mapsto \lambda y_m.f(H_n(\overline{y}_{m}))\}\).
Removal Rules
A *flex equation* is an equation both sides of which are flex terms.

- **[t]** Trivial equations.
  \[(E_1, \lambda \tau. X(\overline{s})) \cong \lambda \tau. X(\overline{s}), E_2)|_W \Rightarrow [t]_\epsilon (E_1, E_2)|_W\]

- **[fs]** Flex-same.
  If \(\lambda \tau. X(\overline{y}_n)\) and \(\lambda \tau. X(\overline{y}_m)\) are patterns, and \(X \in W\) then
  \[(E_1, \lambda \tau. X(\overline{y}_n) \triangleright \lambda \tau. X(\overline{y}_m), E_2)|_W \Rightarrow [fs]_\theta (E_1, E_2)\theta|_W',\]
  where \(\theta = \{X \mapsto \lambda \overline{y}_n.H(\overline{z})\} \) with \(\{\overline{z}\} = \{y_i | y_i = y'_i, 1 \leq i \leq n\}\).

- **[fd]** Flex-different.
  If \(\lambda \tau. X(\overline{y})\) and \(\lambda \tau. Y(\overline{y'})\) are patterns, and \(\{X \in W \text{ and } Y \in W \text{ if } \triangleright \text{ is } \cong \} \) then
  \[(E_1, \lambda \tau. X(\overline{y}) \triangleright \lambda \tau. Y(\overline{y'}), E_2)|_W \Rightarrow [fd]_\theta (E_1, E_2)\theta|_W',\]
  where \(\theta = \{X \mapsto \lambda \overline{y}.H(\overline{z}), Y \mapsto \lambda \overline{y'}.H(\overline{z})\} \) with \(\{\overline{z}\} = \{\overline{y}\} \cap \{\overline{y'}\}\).

### 3.2 Main Property

An HOLN-refutation is a sequence of HOLN-steps
\[E|_W = E_0|_{W_0} \Rightarrow_{\alpha_1, \vartheta_1} E_1|_{W_1} \Rightarrow_{\alpha_2, \vartheta_2} \cdots \Rightarrow_{\alpha_n, \vartheta_n} E_n|_{W_n}\]
such that there is no HOLN-step starting with the goal \(E_n|_{W_n}\). We abbreviate this sequence by \(E_0|_{W_0} \Rightarrow^*_\vartheta E_n|_{W_n}\), where \(\vartheta = \vartheta_1 \ldots \vartheta_n\). The set of partial answers computed with HOLN for a goal \(E|_W\) is
\[PreAns^{HOLN}_R(E|_W) = \{\langle \vartheta, E'|_{W'} \rangle | \exists \text{ HOLN-refutation } E|_W \Rightarrow^*_\vartheta E'|_{W'}\},\]
and the set of answers of HOLN is
\[Ans^{HOLN}_R(E|_W) = \{\vartheta \gamma'|_{\text{vars}(E)} | \langle \vartheta, E'|_{W'} \rangle \in PreAns^{HOLN}_R(E|_W) \text{ and } \gamma' \in \text{Sol}_R(E'|_{W'})\}\]

HOLN is designed not to solve all equations: most of the flex equations are not transformed by HOLN-refutations. The reasons for not solving all flex equations are (i) that a flex equation always has a solution, and (ii) that in general there is no minimal complete set of unifiers for a flex equation. Therefore, the result of an HOLN-refutation is a pair \(\langle \vartheta, E'|_{W'} \rangle\) where \(E'\) is a sequence of unsolvable flex equations. This design decision is similar to the one which underlies Huët’s higher-order preunification procedure, where flex equations are kept unsolved \[3\].

HOLN enjoys the following properties:

**soundness:** \(Ans^{HOLN}_R(E|_W) \subseteq \text{Sol}_R(E|_W)\)

**completeness:** for any \(\gamma \in \text{Sol}_R(E|_W)\) there exists \(\theta \in Ans^{HOLN}_R(E|_W)\) such that \(\theta \leq \gamma \{\text{vars}(E)\}\)
Soundness follows easily from an inductive proof by case distinction on the inference rule used in the first step of the HOLN-refutation.

The main ideas of our completeness proof of HOLN are:

(a) We define the set $\mathsf{Cfg}$ of tuples of the form $\langle E/\gamma, \rho \rangle$ with $\rho$ a rewrite proof of $\gamma \in \mathsf{Sol}_R(\Gamma_E)$. Such tuples are called configurations.

(b) We identify a well founded ordering $\succ \subseteq \mathsf{Cfg} \times \mathsf{Cfg}$ such that whenever $\langle E/\gamma, \rho \rangle \in \mathsf{Cfg}$ and $e \in E$ can be selected in an HOLN-step, then there exists a pair $\langle \pi, \langle E'/\gamma', \rho' \rangle \rangle$ with $\pi$ an HOLN-step of the form $E \Rightarrow \theta E', \langle E/\gamma, \rho \rangle \succ \langle E'/\gamma', \rho' \rangle$, and $\gamma = \theta \gamma [\mathsf{vars}(E)]$.

We can define such $\succeq$ for HOLN as the lexicographic combination of the orderings $\succeq_A, \succeq_B, \succeq_C$, where:

- $\langle E/\gamma, \rho \rangle \succeq_A \langle E'/\gamma', \rho' \rangle$ iff $\Sigma_{e \in E} |\rho(e\gamma)| \geq \Sigma_{e' \in E'} |\rho'(e'\gamma')|$,  
- $\langle E/\gamma, \rho \rangle \succeq_B \langle E'/\gamma', \rho' \rangle$ iff $\{|X\gamma| \mid X \in \mathsf{Dom}(\gamma)\} \succeq_{\mathsf{mul}} \{|X'\gamma'| \mid X' \in \mathsf{Dom}(\gamma')\}$,  
- $\langle E/\gamma, \rho \rangle \succeq_C \langle E'/\gamma', \rho' \rangle$ iff $|E\gamma| \succeq_{\mathsf{mul}} |E'\gamma'|$

The restriction $\succ$ of $\succeq$ is obviously well-founded, and its existence implies the completeness of HOLN.

HOLN can be regarded as an extension of the first-order lazy narrowing calculus LNC [8,9] to higher-order one in the framework of EPRSs. This framework was first used by Prehofer [10] in the design of his higher-order lazy narrowing calculus LN for solving goals consisting of directed equations. HOLN can also be viewed as an extension of LN using the techniques developed in the refinements of LNC.

There are three sources of nondeterminism in computations with HOLN-derivations: the choice of the equation in the current goal, the choice of the inference rule of HOLN, and the choice of the rewrite rule of $\mathcal{R}$ when narrowing steps are performed. The completeness proof outlined above reveals a stronger result: HOLN is strongly complete, i.e., completeness is independent of the choice of the equation in the current goal.

In the sequel we will investigate the possibility to reduce the nondeterminism of computations with HOLN-derivations by reducing the choices of inference rules applicable to a given goal.

## 4 Refinements of HOLN

The main source of nondeterminism with HOLN-derivations is due to the many choices of solving an equation between a rigid term and a flex term. We call such an equation a flex/rigid equation. For example, to solve an equation of the form $\lambda x.X(\overline{s_n}) \triangleright \lambda x.t$ where $\lambda x.t$ is a rigid term, we have to consider all possible applications of rules [ov], [p], [i] (if $\mathsf{head}(t) \notin \mathcal{F}_d$) and [on] (if $\mathsf{head}(t) \in \mathcal{F}_d$). Also, the application of rule [ov] is a source of high nondeterminism, as long as we have large freedom to choose the defined symbols whose inference rules are employed in performing the [ov]-step.
In the sequel we describe two refinements of HOLN towards more deterministic versions, by restricting the narrowing problem to particular classes of EPRSs.

4.1 HOLN: Refinement for Left-Linear EPRSs

The restriction of programs to left-linear TRSs is widely accepted in the declarative programming community. It is well known that for left-linear confluent TRSs the standardization theorem holds [11]. This result allows to avoid the application of the outermost narrowing at nonvariable position to certain parameter-passing descendants in the case of the first-order lazy narrowing calculus LNC, without losing completeness [9]. As a consequence, the search space of LNC is reduced when the given TRS is left-linear and confluent.

In this subsection we show that a similar refinement is possible for confluent LEPRSs. This result is based on the fact that the standardization theorem holds for confluent LEPRSs as well. In the sequel we assume that \( \mathcal{R} \) is an LEPRS.

To explain our result, we need some more definitions. A parameter-passing equation of a goal \( E'/W' \) in an HOLN-derivation \( \Pi : E/W \Rightarrow^* \theta E'/W' \) is either

(a) an equation \( \lambda \overline{x}. s_k \triangleright \lambda \overline{x}. l_k \) \((1 \leq k \leq n)\) if the last step of \( \Pi \) is of the form:

\[
(E_1, \lambda \overline{x}. f(s_n) \triangleright \lambda \overline{x}. t, E_2)|W \Rightarrow_{\text{on}}, \epsilon (E_1, \lambda \overline{x}. s_n \triangleright \lambda \overline{x}. l_n, \lambda \overline{x}. r \triangleright \lambda \overline{x}. t, E_2)|W',
\]

(b) an equation \( \lambda \overline{x}. H_k(s_m \theta) \triangleright \lambda \overline{x}. l_k \) \((1 \leq k \leq n)\) if the last step of \( \Pi \) is of the form:

\[
(E_1, \lambda \overline{x}. X(s_m) \triangleright \lambda \overline{x}. t, E_2)|W \Rightarrow_{\text{ov}}, \theta (E_1 \theta, \lambda \overline{x}. H_n(s_m \theta) \triangleright \lambda \overline{x}. l_n, \lambda \overline{x}. r \triangleright \lambda \overline{x}. t \theta, E_2 \theta)|W'.
\]

A parameter-passing descendant of a goal \( E'/W' \) in an HOLN-derivation \( \Pi : E/W \Rightarrow^*_\theta E'/W' \) is either a parameter-passing equation or a descendant of a parameter-passing equation. Note that parameter-passing descendants are always oriented equations. To distinguish them from the other oriented equations, we will write \( s \rhd t \) instead of \( s \triangleright t \).

Positions in \( \lambda \)-terms are sequences of natural numbers which define the path to a subterm of a \( \lambda \)-term. We denote by \( \epsilon \) the empty sequence, by \( i \cdot p \) the sequence \( p \) appended to an element \( i \), and by \( p + p' \) the concatenation of sequences \( p \) and \( p' \). A position \( p \) is above a position \( p' \), notation \( p < p' \), if there exists \( q \neq \epsilon \) such that \( p' = p + q \). The subterm of \( s \) at position \( p \), written \( s|_p \), is defined as

- \( s|_\epsilon = s \), \( \nu(\overline{t}_n)|_i \triangleright_p t_i|_p \) if \( 1 \leq i \leq n \), \( (\lambda \overline{x}_m.t)|_{1..p} = (\lambda x_2 \ldots x_m.t)|_p \),
- undefined otherwise.
Higher-Order Lazy Narrowing Calculus

The set of positions of a term $t$ is denoted by $\text{Pos}(t)$. $p$ is a pattern position of a term $t$, notation $p \in \text{Pos}(t)$, if $p \in \text{Pos}(t)$ and $\text{head}(t|_q) \not\in \text{vars}$ for all $q < p$.

A rewrite proof $\rho$ of $\gamma \in \text{Sol}_R(E|_W)$ is outside-in if the following conditions are satisfied for all equations $e$ of $E$:

(a) $\rho(e\gamma)$ is an outside-in reduction derivation, that is, if $\rho(e\gamma)$ rewrites at positions $p_1, \ldots, p_n$ with the $\overline{x}$-lifted rewrite rules $l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n$ respectively, then the following condition is satisfied for all $1 \leq i \leq n - 1$: if there exists $j$ with $i < j$ such that $p_i = p_j + q$ then $q \in \text{Pat}(l_j)$ for the least such $j$.

(b) If $e = s \triangleright t \in E$ and $\rho(e\gamma)$ has a rewrite step at position $1 \cdot p$ such that no later rewrite steps take place above position $1 \cdot p$ then $p \in \text{Pat}(t)$.

The following theorem can be proved using the the standardization theorem for confluent LEPRSs [12]:

**Theorem 1** Let $R$ be a confluent LEPRS and $\gamma \in \text{Sol}_R(E|_W)$. Then there exists an outside-in rewrite proof of $\gamma \in \text{Sol}_R(E|_W)$.

Theorem 1 with its constructive proof states that for any rewrite proof $\gamma$ we can construct an outside-in rewrite proof. Recall the proof sketch of completeness of HOLN. Even if we consider only configurations of the form $\langle E|_W, \gamma, \rho \rangle$ with $\rho$ an outside-in rewrite proof of $\gamma \in \text{Sol}_R(E|_W)$, the proof of strong completeness of HOLN remains valid when $R$ is restricted to a confluent LEPRS. This implies that the HOLN-refutations considered in the proof do not contain [on]-steps applied to equations of the form

$$\lambda \overline{x}.f(\overline{s_n}) \triangleright \lambda \overline{x}.X(\overline{y}) \text{ where } f \in F_d. \quad (1)$$

Now we design HOLN$_1$ as follows.

HOLN$_1$ is the same as HOLN except that the inference rule [on] is not applied to the (selected) equation of form (1).

For HOLN$_1$, we have the following main result.

**Main Result:** HOLN$_1$ is sound and strong complete for confluent LEPRSs [6].

4.2 HOLN$_2$: Refinement for Constructor LEPRSs

This refinement is inspired by a similar refinement of LNC with leftmost equation selection strategy for left-linear constructor TRSs [8]. It addresses the possibility to avoid the generation of parameter-passing descendants of the form $s \triangleright t$ with $t \not\in T(F_c, V)$. The effect of this behavior is that the nondeterminism between the inference rules [on] and [d] disappears for parameter-passing descendants.

In the first-order case, it is shown that LNC with leftmost equation selection strategy $S_{\text{left}}$ does not generate parameter-passing descendants $s \triangleright t$ with $t \not\in T(F_c, V)$. Unfortunately, this property is lost in the higher-order case mainly because the leftmost equation may be a flex equation to which no inference rule is applicable. Therefore, $S_{\text{left}}$ can not be adopted. To restore this property, we need to modify HOLN and to introduce a new equation selection strategy.
We define a new calculus HOLN\textsubscript{2} as the calculus consisting of all the inference rules of HOLN\textsubscript{1} and of the inference rule [c] defined as follows.

[c] Constructor propagation.
If \( \exists s : \lambda x_n.X(y_n) \in E_1 \) and \( s' = \lambda y_n.s(x_n) \) then

\[
(E_1, \lambda x.X(t_n) \triangleright \lambda x.u, E_2) \triangleright [c, \epsilon] (E_1, \lambda x.s'(t_n) \triangleright \lambda x.u, E_2) | W'.
\] (2)

We give to [c] the highest priority.

Note that the application of [c] replaces the outermost occurrence of \( X \) in the selected equation by \( \lambda y_n.s(x_n) \).

We define a strategy \( S_c \) as follows. Let \( e \) be a selected equation of the form

\[
\lambda x.X(t_n) \triangleright \lambda x.u
\]
in a goal \( (E_1, e, E_2) | W \). An HOLN\textsubscript{2}-step

\[
(E_1, e, E_2) | W \triangleright [\alpha, \theta] (E_1, E, E_2) \theta | W',
\] (3)

respects strategy \( S_c \) if the inference step (3) is enabled only when all the parameter-passing descendants \( s \triangleright t \) in \( E_1 \) have \( t \) as a flex term.

We can easily prove the following lemma [6].

**Lemma 1** Let \( R \) be a confluent constructor LEPRS and \( \Pi \) be an HOLN\textsubscript{2}-derivation that respects strategy \( S_c \). All the equations \( s \triangleright t \) in \( \Pi \) satisfy the property \( t \in T(F_c, V) \).

We have the following result for HOLN\textsubscript{2}.

**Main Result:** HOLN\textsubscript{2} with strategy \( S_c \) is sound and complete for confluent constructor LEPRSs.

Soundness follows from the fact that both rule [c] and the inference rules of HOLN are sound when strategy \( S_c \) is obeyed. The completeness proof works along the same lines as the completeness proof of HOLN\textsubscript{1}, but the definition of the ordering between configurations is much more involved than the definition of \( \triangleright \) [6].

5 Extensions of the Computational Model of HOLN

Many applications from the area of scientific computing require capabilities for solving constraints such as systems of linear equations, polynomial equations, or differential equations. Since HOLN can solve equations only over the domain of simply typed \( \lambda \)-terms, we investigated the possibility to extend HOLN to solve equations over some specific constraint domains equipped with well known solving methods. The results of our investigation are incorporated in our system CFLP (Constraint Functional Logic Programming system) [5]. CFLP is a distributed constraint solving system implemented in *Mathematica*, which extends the solving power of HOLN with methods to solve:
– systems of linear equations and equations with invertible functions,
– systems of multivariate polynomials, using Buchberger algorithm,

The computational capabilities of CFLP go beyond the guarantee of our completeness results. This naturally points to the area of further research, namely the study of completeness of HOLN combined with complete external solvers.

6 Application

We will explain by an example how CFLP employs HOLN to compute solutions of problems formalized in higher-order equational programs.

Program Calculation. This example was briefly discussed in [5] to give a flavor to the capability of CFLP. Here we describe how HOLN works to compute efficient functional programs from less efficient but easily understandable ones. Such derivations are typical computations of higher-order equational programming.

We pose a question in a form of a goal that involves a higher-order variable. Then HOLN operates on the goal and transforms it successively into subgoals that are in turn solved. The computation is completed when HOLN finds no more subgoals to solve. HOLN delivers a substitution in which the higher-order variable is bound to the desired program.

Consider the problem of writing a program to check whether a list of numbers is steep. We say a list is steep if each element is greater than or equal to the average of the elements that follow it. By default, the empty list is steep.

With CFLP, such a test can be done by the function steep defined via the program Prog given below:

\[
\text{Prog} = \{\text{steep}[^\{} \rightarrow \text{True},
\text{steep}[a \mid x] \rightarrow (a \times \text{len}[x] \geq \text{sum}[x]) \wedge \text{steep}[x],
\text{sum}[^\{} \rightarrow 0, \text{sum}[x \mid y] \rightarrow x + \text{sum}[y],
\text{len}[^\{} \rightarrow 0, \text{len}[x \mid y] \rightarrow 1 + \text{len}[y],
\text{tupling}[x] \rightarrow c3[\text{sum}[x], \text{len}[x], \text{steep}[x]]\}\;
\]

where

– the underlined symbols denote free variables,
– $[^\{}]$ denotes the empty list and $[H \mid T]$ denotes a list with head $H$ and tail $T$,
– $c3$ is a data constructor defined by

\[
\text{TypeConstructor}[\text{Tuple} = c3[\text{Float}, \text{Float}, \text{Float}]];
\]

This command defines the type constructor Tuple with associated data constructor $c3$ of type $\text{Float} \times \text{Float} \times \text{Float} \rightarrow \text{Tuple}$, and the corresponding
data selectors \texttt{sel.c3.1}, \texttt{sel.c3.2}, \texttt{sel.c3.3}. CFLP assumes that, for a given constructor \( c_n \), the following axioms hold for any \( x, x_1, \ldots, x_n \) and \( 1 \leq k \leq n \):

\[
c_n(\texttt{sel.c3.1}(x), \ldots, \texttt{sel.c3.3}(x)) = x,
\]

\[
\texttt{sel.c3.3}(c_n(x_1, \ldots, x_n)) = x_k.
\]

CFLP implements partially these axioms via the following additional inference rule:

\[ [\textit{fl}] \text{ Flattening.} \]

If \( \lambda \overline{x}.t \) is rigid then

\[
(E_1, \lambda \overline{x}.X(s_m, c(t_n), u_p) \equiv \lambda \overline{x}.t, E_2)|_W \Rightarrow [\textit{fl}], \theta
\]

\[
(E_1 \theta, \lambda \overline{x}.t \theta \triangleright \lambda \overline{x}.H_n(\overline{x}), \lambda \overline{x}.H(s_m \theta, H_n(\overline{x}), u_p \theta) \equiv \lambda \overline{x}.t \theta, E_2 \theta)|_W,
\]

where \( \theta = \{ X \mapsto \lambda \overline{x}_m, y, \overline{x}_p, H(\overline{x}_m, \texttt{sel.c3.3}(y), \overline{x}_p) \} \).

In addition, CFLP replaces automatically by \( t \) all the terms of the form

\[
c_n(\texttt{sel.c3.1}(t), \ldots, \texttt{sel.c3.3}(t)).
\]

\texttt{Prog} is modular and easy to understand, but it is rather inefficient because the computation of \texttt{steep} for a given list has quadratic complexity. It is desirable to have a means to automatically compute the efficient version of the function \texttt{steep} defined above. Such a computation can be described via the so called fusion calculational rule shown below:

\[
\frac{f(e) = e'}{f(\texttt{foldr}(g, [n \mid ns]))} = h(n, f(ns)) \quad \text{if } f(\texttt{foldr}(g, e, ns)) = \texttt{foldr}(h, e', ns)
\]

(4)

where \texttt{foldr} is the usual fold function on lists.

In \[ [\textit{fl}] \], the expression \( f(\texttt{foldr}(g, e, ns)) \) describes the inefficient computation, and \( \texttt{foldr}(h, e', ns) \) is its efficient version. In our particular case, the inefficient computation of \texttt{steep}([\( n \mid ns \)]) is described by \texttt{sel.c3.3(tupling([\( n \mid ns \)]))}. To find its efficient version, we employ rule \[ [\textit{fl}] \] with \( f = \texttt{tupling} \) and \( g = \texttt{Cons} \) to the inefficient computation \texttt{tupling}([\( n \mid ns \)]) and compute an appropriate answer for the higher-order variable \( \texttt{H} \) to describe its efficient version \( \texttt{H}(n, \texttt{tupling}(ns)) \):

\[
\text{TSolve[}
\lambda[{n, ns}, \texttt{tupling}([n \mid ns])] \approx \lambda[{n, ns}, \texttt{H}[n, \texttt{c3}][\texttt{sum}[ns], \texttt{len}[ns], \texttt{steep}[ns]]]]],
\]

\[
\{h1, \},
\]

\[
\text{DefinedSymbol} \rightarrow \{
\text{steep} : \texttt{TyList[Float]} \rightarrow \texttt{Bool},
\text{sum} : \texttt{TyList[Float]} \rightarrow \texttt{Float},
\text{len} : \texttt{TyList[Float]} \rightarrow \texttt{Float},
\text{tupling} : \texttt{TyList[Float]} \rightarrow \texttt{Tuple},
\}
\]

\[
\text{EnableSelectors} \rightarrow \text{True},
\]

\[
\text{Rules} \rightarrow \text{Prog};
\]

\[
\text{Type checking program ...}
\]

\[
\text{Type checking goal ...}
\]

\[
\{ \texttt{H} \rightarrow \lambda[{x$1865, x$1866},
\}

\]
The \texttt{TSo\textsubscript{olve}} call of CFLP expects three arguments: the list of equations to be solved, the list of variables for which we want to compute normalized values, and the list of other variables. The computation performed during the execution of \texttt{TSo\textsubscript{olve}} call can be controlled via the following options:

- \texttt{Rules}: specifies the LEPRS,
- \texttt{DefinedSymbol}: specifies the list of possibly typed-annotated defined symbols,
- \texttt{EnableSelectors}: specifies whether to enable or disable the usage of data selectors in the solving process.

In this case, the goal submitted to the underlying calculus of CFLP is

\[
\lambda\textit{n,ns.tu\textsubscript{pling}}([n | ns]) \approx \lambda\textit{n,ns.H(n, c3(sum(ns), len(ns), steep(ns))})\vert_{\{H\}}
\]

To compute the binding for \texttt{H}, CFLP performs the following derivation:

\[
\lambda\textit{n,ns.tu\textsubscript{pling}}([n | ns]) \approx \lambda\textit{n,ns.H(n, c3(sum(ns), len(ns), steep(ns))})\vert_{\{H\}}
\]

\[
\psi_{[\textit{on}]} \quad (\lambda\textit{n,ns.[n | ns]} \triangleright \lambda\textit{n,ns.X(n,ns)}, \lambda\textit{n,ns.c3(sum(X(n,ns)), len(X(n,ns)), steep(X(n,ns)))) \approx \lambda\textit{n,ns.H(n, c3(sum(ns), len(ns), steep(ns))})\vert_{\{H\}}
\]

\[
\psi_{[\textit{fl}]} \quad \{\triangleright \lambda\textit{x,y,H1(x,sel-c3\textsubscript{1}(y),sel-c3\textsubscript{2}(y),sel-c3\textsubscript{3}(y))} \begin{array}{l}
\lambda\textit{n,ns.sum(ns)} \triangleright \lambda\textit{n,ns.X1(n,ns)}, \\
\lambda\textit{n,ns.len(ns)} \triangleright \lambda\textit{n,ns.X2(n,ns)}, \\
\lambda\textit{n,ns.steep(ns)} \triangleright \lambda\textit{n,ns.X3(n,ns)}, \\
\lambda\textit{n,ns.c3(sum([n | ns]), len([n | ns]), steep([n | ns]))} \approx \\
\lambda\textit{n,ns.H1(n, X1(n,ns), X2(n,ns), X3(n,ns)))\vert_{\{H1\}}
\end{array}
\]

\[
\psi_{[\textit{fl}]} \quad \{\triangleright \lambda\textit{x,y,H1(x,ns,sum(ns),X2\rightarrow\lambda\textit{ns},len(ns),X3\rightarrow\lambda\textit{ns},steep(ns)} \begin{array}{l}
\lambda\textit{n,ns.c3(sum([n | ns]), len([n | ns]), steep([n | ns]))} \approx \\
\lambda\textit{n,ns.H1(n, sum(ns), len(ns), steep(ns))}\vert_{\{H1\}}
\end{array}
\]

\[
\psi_{[\textit{fl}]} \quad \{\triangleright \lambda\textit{x,y,ns,sum(ns),X2\rightarrow\lambda\textit{ns},len(ns),X3\rightarrow\lambda\textit{ns},steep(ns)} \begin{array}{l}
\lambda\textit{n,ns.c3(sum([n | ns]), len([n | ns]), steep([n | ns]))} \approx \\
\lambda\textit{n,ns.H3(n, sum(ns), len(ns), steep(ns))}\vert_{\{H2,H3\}}
\end{array}
\]

\[
G \approx (\lambda\textit{n,ns.n + sum(ns)} \approx \lambda\textit{n,ns.H2(n, sum(ns), len(ns), steep(ns))}, \lambda\textit{n,ns.1 + len(ns)} \approx \lambda\textit{n,ns.H3(n, sum(ns), len(ns), steep(ns))}, \lambda\textit{n,ns.(n * len(ns)} \geq \text{sum(ns))} \land \text{steep(ns))} \\
\quad \approx \lambda\textit{n,ns.H4(n, sum(ns), len(ns), steep(ns))}\vert_{\{H2,H3,H4\}}
\]
Finally, CFLP solves the goal $G$ produced by the derivation depicted above by employing the inference rules [i], [p], [d], [fs], and [fd] of HOLN to compute the unifier \{ $H_2$ $\mapsto$ $\lambda x_4.x_1 + x_2$, $H_3$ $\mapsto$ $\lambda x_4.1 + x_3$, $H_4$ $\mapsto$ $\lambda x_4.((x_1 \times x_3 \geq x_2) \land x_4)$ \} of the equational part of $G$.

In this way CFLP computes the answer

$$\{ H \mapsto \lambda n, ns.c3(n + sel\cdot c3\cdot 1(ns)), 1 + sel\cdot c3\cdot 2(ns), n \times sel\cdot c3\cdot 2(ns) \geq sel\cdot c3\cdot 1(ns) \land sel\cdot c3\cdot 3(ns)\}$$

which corresponds to the Mathematica representation of the answer produced by CFLP.

7 Conclusions and Future Work

We have presented a new lazy narrowing calculus HOLN for EPRS designed to compute solutions which are normalized with respect to a given set of variables, and then have presented two refinements to reduce its nondeterminism. Those refinements result in two calculi which are sound and complete.

The results presented in this paper owe largely to a new formalism in which we treat a goal as a pair consisting of a sequence of equations and a set of variables for which we want to compute normalized answers. This formulation of narrowing has the following advantages:

- it clarifies problems and locates points for optimization during the refutation process of goals,
- it simplifies the soundness and completeness proofs of the calculi,
- it simplifies and systematizes the implementation of the lazy narrowing calculus as a computational model of a higher-order functional logic programming system.

All the calculi given in this paper have been implemented as part of our distributed constraint functional logic system CFLP[1,4,5].

An interesting direction of research is to extend HOLN to conditional EPRSs. A program specification using conditions is much more expressive because it allows the user to impose equational conditions under which rewrite steps are allowed. Such an extension is quite straightforward to design, but it introduces many complexities for proving completeness.

Acknowledgements. The work reported in this paper has been supported in part by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Scientific Research (B) 12480066, 2000-2002. Mircea Marin has been supported by JSPS postdoc fellowship 00096, 2000-2001.
References


Classifying Isomorphic Residue Classes

Andreas Meier, Martin Pollet*, and Volker Sorge**

Fachbereich Informatik, Universität des Saarlandes, Germany,
{ameier|pollet|sorge}@ags.uni-sb.de
http://www.ags.uni-sb.de/~{ameier|pollet|sorge}

Abstract. We report on a case study on combining proof planning with
computer algebra systems. We construct proofs for basic algebraic prop-
erties of residue classes as well as for isomorphisms between residue
classes using different proving techniques, which are implemented as stra-
tegies in a multi-strategy proof planner. We show how these techniques
help to successfully derive proofs in our domain and explain how the
search space of the proof planner can be drastically reduced by employ-
ling computations of two computer algebra systems during the planning
process. Moreover, we discuss the results of experiments we conducted
which give evidence that with the help of the computer algebra systems
the planner is able to solve problems for which it would fail to create a
proof otherwise.

1 Introduction

We report on a case study that combines proof planning with computer al-
gebra systems. We classify residue class sets over the integers together with
given binary operations in terms of their basic algebraic properties and addi-
tionally into sets of isomorphic structures. A residue class set over the inte-
gers, $RS_n$, is either the set of all congruence classes modulo an integer $n$, i.e.,
$\mathbb{Z}/n\mathbb{Z}$, or an arbitrary subset of $\mathbb{Z}/n\mathbb{Z}$. Concretely, we are dealing with sets of the
form $\mathbb{Z}/3\mathbb{Z}, \mathbb{Z}/5\mathbb{Z}, \mathbb{Z}/3\mathbb{Z}\setminus\{\bar{1}_3\}, \mathbb{Z}/5\mathbb{Z}\setminus\{\bar{0}_5\}, \{\bar{1}_6, \bar{3}_6, \bar{5}_6\}, \ldots$ where $\bar{1}_3$ denotes the congru-
ence class 1 modulo 3. If $c$ is an integer we write also $cm(c)$ for the congruence
class $c$ modulo $n$. A binary operation $\cdot$ on a residue class set is usually written
in $\lambda$-function notation. $\cdot$ can be of the form $\lambda x y \cdot x$, $\lambda x y \cdot y$, $\lambda x y \cdot c$ where $c$ is a
constant congruence class (e.g., $\bar{1}_3$), $\lambda x y \cdot x + y$, $\lambda x y \cdot x y$, $\lambda x y \cdot x - y$, where $+$, $\cdot$, $-$
denote addition, multiplication, and subtraction on congruence classes over
the integers, respectively. Furthermore, $\cdot$ can be any combination of the basic
operations with respect to a common modulo factor, (e.g., $\lambda x y \cdot (x + \bar{1}_3) - (y + \bar{2}_3)$).

The case study was carried out in the ΩMEGA theorem proving environ-
ment [2]. It consisted of two parts: (1) To examine the basic algebraic properties
of given residue class structures and classify them into terms of the algebraic
structure they form (e.g., group, monoid, or quasi-group). (2) Structures of the
same type and cardinality are then further investigated to identify isomorphism

* The author’s work was supported by the ‘Graduiertenförderung des Saarlandes’.
** The author’s work was supported by the ‘Studienstiftung des Deutschen Volkes’.
classes. The first part of the case study was reported in [13]. In this paper we concentrate on how we determine and prove whether residue class structures are isomorphic to each other or not. For an extensive report on the whole case study, including a detailed presentation of the constructed proofs, we refer the reader also to [12].

The proof constructions are used within a tutor system for an interactive mathematical course in algebra. For tutoring purposes it was necessary to have proofs in human-oriented reasoning style using different proving techniques. Therefore, we chose multi-strategy proof planning as our main tool for constructing proofs. On the one hand, it allows us to easily model different proving techniques by different strategies. On the other hand, it enables us to exploit the power of computer algebra systems in a sound way [9,18] to speed up the proving process. The aim of the paper is to present how multi-strategy proof planning is used to determine and verify isomorphism of residue class structures and how computer algebra is employed to guide and shorten the search during the proof planning process. We do, however, not explain how the examples are actually used in a tutor system; we instead refer the reader to [3], where a system description of the ActiveMath learning environment is given.

The paper is organized as follows: we first give a brief overview of multi-strategy proof planning in the ΩMega system and the integration of computer algebra with proof planning. Section 3 contains a summary of the classification of residue class structures with respect to their simple algebraic properties, which is described in detail in [13]. Section 4 is the major part giving the details of how isomorphic residue class structures are identified, and how the necessary isomorphism and non-isomorphism proofs are constructed. We then give, in Sec. 5, a brief account of some of the experiments we carried out and conclude with a discussion of some of the related work.

2 Proof Planning and Computer Algebra

2.1 Multi-strategy Proof Planning

Proof planning [11] considers mathematical theorems as planning problems where an initial partial plan is composed of the proof assumptions and the theorem as open goal. A proof is then constructed with the help of abstract planning steps, called methods, that are essentially partial specifications of tactics known from tactical theorem proving. In order to ensure correctness, proof plans have to be executed to generate a sound calculus level proof.

In the ΩMega system [2] the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see [15] for details). That is, methods can encode general proving steps as well as knowledge particular to a mathematical domain. Moreover, control rules specify how to traverse the search space by influencing the ordering of method application and the choice of the next goal depending on certain domains or proof situations. ΩMega’s new proof planner, MULTI [14], allows also for the specification of different planning strategies to control the overall planning behavior.
Methods in ΩMEGA are essentially tactics known from tactical theorem proving augmented with pre- and postconditions, so-called premises and conclusions. Premises and conclusions indicate the inference of the method (the conclusions should follow from the premises) and indicate the role of the method in the planning process. For instance, Indirect is a method whose purpose is to prove a goal \( P \) by contradiction. If Indirect is applied to a goal \( P \) then it closes this goal and introduces the new goal \( \bot \). Moreover, it adds the new hypothesis \( \neg P \) to the proof plan such that the contradiction \( \bot \) can be constructed using also \( \neg P \). Thereby, \( P \) is the conclusion of the method, whereas \( \bot \) is the premise of the method.

Control rules provide the possibility to introduce mathematical knowledge on how to proceed in the proof planning process. They can influence the planners behavior at choice points (e.g., which goal to tackle next or which method to apply next) by preferring, rejecting, or enforcing members of the corresponding alternative lists (e.g., the list of possible goals or the list of possible methods). This promotes certain promising search paths and can thus prune the search space. In particular, we employ control rules to prefer a particular instance from a list of possible variable instantiations. As example we present the select-instance control rule in the next section.

In ΩMEGA different proof techniques for a problem class can be realized by different planner strategies [14]. The planner strategies can employ different sets of methods and control rules and can thus allow to tackle the same problem in different ways. The reasoning about which strategy to employ on a problem (provided there are several applicable strategies) and about the switching of strategies is an additional choice point in Multi. Therefore, the planner can also backtrack from applied strategies and perform search on the level of strategies.

### 2.2 Using Computer Algebra in Proof Planning

When proof planning in the domain of residue classes we employ symbolic calculations to guide and simplify the search for proof plans. In particular, we use the mainstream computer algebra system Maple [16] and GAP [5], a system specialized on group theory. We are not concerned with the technical side of the integration since we exploit previous work, in particular [9] that presents the integration of computer algebra into proof planning, and [18] that exemplifies how the correctness of certain limited computations of a large-scale computer algebra system such as Maple can be guaranteed within the proof planning framework. In this paper we rather concentrate on the cooperation between the systems in the context of exploring residue class properties.

We use symbolic calculations in two ways: (1) in control rules hints are computed to help guiding the planning process, and (2) within the application of a method, equations are solved with Maple to simplify the proof. As side-effect both cases can restrict possible instantiations of meta-variables.1

---

1 Meta-variables are place-holders for terms whose actual form is computed at a later stage in the proof search.
(1) is implemented, for instance, in the control rule \texttt{select-instance} (used in the strategy \texttt{TryAndError}, see next section). The rule is triggered after decomposition of an existentially quantified goal which results in the introduction of a meta-variable as substitute for the actual witness term. After an existential quantifier is eliminated the control rule computes a hint with respect to the remaining goal that is used as a restriction for the introduced meta-variable. For instance, when proving that the residue class set $RS_n$ is not closed under the operation $\circ$, that is, there exist $a, b \in RS_n$ such that $a \circ b \not\in RS_n$, the control rule would supply a hint as to what $a$ and $b$ might be. If hints can be computed the meta-variables are instantiated before the proof planning proceeds.

To obtain suitable hints \texttt{select-instance} sends corresponding queries to GAP and MAPLE (a detailed description of the hint system for simple algebraic properties is given in [13], the hint system when classifying isomorphic structures is described in Sec. 4). However, all such computations by MAPLE and GAP are treated as a hint by the proof planner; that is, in case the proving attempt fails for a particular instantiation computed by the computer algebra systems the planner falls back to its regular search.

In (2), the use of calculations is realized within the \texttt{Solve-Equation} method. Its purpose is to justify an equational goal using MAPLE and, if necessary, to instantiate meta-variables. In detail, it works as follows: if an open goal is an equation, MAPLE’s function \texttt{solve} is applied to check whether the equality actually holds. Should the equation contain meta-variables then these are considered as the variables the equation is to be solved for, and they are supplied to \texttt{solve} as additional arguments. In case the equation involves modulo functions with the same factor on both sides, MAPLE’s function \texttt{msolve} is used instead of \texttt{solve}. If MAPLE can successfully solve the equation, the method is applied and possible meta-variables are instantiated accordingly. The computation is then considered correct for the rest of the proof planning process. However, once the proof plan is executed MAPLE’s computation is expanded into low level logic derivations to check its correctness. This is done with the help of a small self-tailored computer algebra system that provides detailed information on its computations in order to construct the expansion. This process is extensively described in [18].

3 Checking Simple Properties

First, we are interested in classifying residue class sets over the integers together with given binary operations in terms of what algebraic structure they form. We automatically classify structures of the form $(RS_n, \circ)$ in terms of magma (also called groupoid), semi-group, monoid, quasi-group, loop, group, and whether they are Abelian. The classification is done by first checking successively if the properties: closure, associativity, existence of the unit element, existence of inverse elements, and the quasi-group axiom (i.e., that for each two elements $a, b \in RS_n$ there exist elements $x, y \in RS_n$ such that $a \circ x = b$ and $y \circ a = b$) hold and then constructing and discharging an appropriate proof obligation. The properties are checked mainly with GAP and proofs for the constructed obliga-
tions are planned with Multi. For instance, GAP is used to check whether a given structure contains a unit element; depending on the result, a proof obligation is constructed stating there exists or there does not exist a unit element in the structure. Multi then tries to produce a proof plan for this statement. If it succeeds the next property is checked; if it fails Multi tries to prove the negation. For discharging proof obligations we have implemented three different proving techniques with strategies in Multi, which use symbolic computations to a varying degree.

The simplest strategy is TryAndError which performs a naïve exhaustive case analysis. This technique is possible since we are in a finite domain and can always enumerate all occurring cases. The planning process usually starts with the expansion of defined concepts such as unit, associative, etc. For resulting universally quantified goals ranging over a residue class a case split on all elements of the structure is performed. For existentially quantified goals all possible instantiations for the quantified variable are successively checked. The latter is done by introducing a meta-variable that is bound successively to the different elements of the residue class set. Here the search space can be reduced by providing the (probably) correct instantiation immediately as a hint with the control rule select-instance (see last section). For instance, when showing that for each element in the structure there indeed exists an inverse, GAP can compute the respective inverses. When the subsequent subgoals cannot be proved, Multi backtracks to the instantiation of the meta-variable and chooses the next element. After the quantifiers are eliminated, the statements about residue classes are transformed to statements about integers which can be solved by numerical simplifications.

The second proving technique is EquSolve. This strategy employs as much as possible equational reasoning. Problems are decomposed to the level of equations; universally quantified variables are replaced by constants and existentially quantified variables by meta-variables. The property then holds, when all equations can be solved by the Solve-Equation method. The strategy employs Maple to check the universal validity of the equation or, in case the equation contains meta-variables, if there is an instantiation of these meta-variables, such that the equation is universally valid. The technique can, however, be applied only to those problems that can be reduced to equations (associativity, unit element, inverse elements, and quasi-group axiom). In particular, it cannot be applied to refute properties or to show closure. EquSolve, like TryAndError, reduces statements on residue classes to statements on integers. For instance, the equation for the inverse element \( cl_n(c) + cl_n(mv) = \bar{0}_n \) containing congruence classes (where \( c \) is a constant and \( mv \) is a meta-variable) is reduced to the corresponding equation on integers \((c + mv) \mod n = 0 \mod n\) before Maple returns a general solution for \( mv \).

The last technique is ReduceToSpecial which applies already known theorems. Here Multi uses theorems from Omega's knowledge-base to reduce a given problem. This strategy does not depend on the help of a computer algebra system. Moreover, the theorems are applied to statements about residue class
structures directly; a reduction to statements about integers as in TryAndError and EquSolve is not necessary.

When automatically discharging proof obligations MULTI attempts first the application of theorems, then equational reasoning and lastly the exhaustive case analysis. That is, we start with the strategy that is generally the most efficient one and end with the most reliable strategy.

In order to test our approach we constructed a large testbed of automatically generated examples from the possible subsets of the residue classes modulo \( n \), where \( n \) ranges from 2 to 10, together with operations that are systematically constructed from the basic operations. We classified 14337 structures with one operation so far. We found 5810 magmas, 109 Abelian magmas, 2064 semi-groups, 1670 Abelian semi-groups, 1018 quasi-groups, 461 Abelian quasi-groups, 93 Abelian monoids, and 780 Abelian groups (the other structures we tested are not closed). Note, that these figures do not mean that we have so many distinct algebraic entities, since our testbed contains many isomorphic structures. For the proofs of the single properties that were tested during the classification, MULTI successfully employed ReduceToSpecial at the rate of 19% and EquSolve to a different set accounting for 21% of the examples. The remaining 60% of the examples could be solved only by the TryAndError strategy. For a more detailed report on the exploration of simple properties of residue structures see [13,12].

4 Identifying Classes of Isomorphic Structures

Checking simple algebraic properties of residue class structures as described in the preceding section allows to classify given structures in terms of the algebraic entity they form. This, however, does not indicate how many of these structures are actually different (i.e., not isomorphic to each other) or are just different representations of the same structure. In this section we present how we classify given sets of residue class structures into equivalence classes of isomorphic structures. To ease the task we already exclude structures that are trivially not isomorphic to each other. Hence, we only examine structures which are of the same algebraic category (e.g., monoids are only compared with other monoids and not with groups) and we consider only structures of the same cardinality since for finite sets structures of different size are trivially not isomorphic.

The idea of the isomorphism classification algorithm is to partition a set of residue class structures into disjoint classes of isomorphic structures. Given such a set we initialize the first isomorphism class with the very first structure of the set. Each further structure \( S \) is checked whether it belongs to an already existing isomorphism class. If we can prove that there is a corresponding isomorphism class, \( S \) is added to this class otherwise a new class is initialized. Whether or not \( S \) belongs to a certain isomorphism class is tested first with MAPLE by constructing a pointwise defined isomorphism mapping \( S \) to a structure \( S' \) of an existing class (the actual computation of this pointwise function is described in more detail in Sec. [11]). If MAPLE’s computation suggests that \( S \) is isomorphic to \( S' \) the corresponding proof obligation is constructed and passed to MULTI.
If \textsc{Maple} cannot find any existing isomorphism class for \( S \) then the classification algorithm assumes that \( S \) belongs to a new isomorphism class. Hence, for each existing isomorphism class it constructs a proof obligation stating that \( S \) is not isomorphic to a structure \( S' \) of this class and sends these proof obligations to \textsc{Multi}. If \textsc{Multi} succeeds to prove all of these obligations, then a new isomorphism class is initialized with \( S \).

In the following we describe how \textsc{Multi} discharges isomorphism and non-isomorphism proof obligations. In particular, we describe how \textsc{Maple} and \textsc{Gap} are employed to support the proof planning process.

4.1 Isomorphism Proofs

In this section we present how \textsc{Multi} plans isomorphism proofs. It employs the same three strategies mentioned in Sec. 3, namely \texttt{TryAndError}, \texttt{EquSolve}, and \texttt{ReduceToSpecial}. We just had to add two methods for the introduction of isomorphism mappings to the \texttt{TryAndError} and \texttt{EquSolve} strategies and one additional theorem for the \texttt{ReduceToSpecial} strategy. Contrary to the proofs of simple properties of structures that could be solved in most cases within one strategy, for isomorphism proofs different subproofs can be solved by different strategies. This means that the strategy \texttt{EquSolve} switches to \texttt{TryAndError}, while \texttt{ReduceToSpecial} uses \texttt{EquSolve} and \texttt{TryAndError} to prove some of the occurring subgoals.

**TryAndError** To prove that two given structures \((RS^1_n, o^1)\) and \((RS^2_m, o^2)\) are isomorphic we have to show that there exists a function \( h:(RS^1_n, o^1) \rightarrow (RS^2_m, o^2) \) such that \( h \) is injective, surjective, and homomorphic. As described in Sec. 3 \texttt{TryAndError} checks for existentially quantified goals all possible instantiations for the quantified variable successively. In the context of finite sets each possible mapping \( h \) can be represented as a pointwise defined function, where the image of each element of the domain is explicitly specified as an element of the codomain.

\textsc{Multi} abbreviates the search for the right instantiation of \( h \) by computing a hint. For an isomorphism \( h:(RS^1_n, o^1) \rightarrow (RS^2_m, o^2) \), a system of equations is generated by instantiating the homomorphism equation \( h(cl_n(i) o^1 cl_n(j)) = h(cl_n(i)) o^2 h(cl_n(j)) \) with all elements of the residue class set \( RS^1_n \). When we take \( cl_n(k) \) to be the result of \( cl_n(i) o^1 cl_n(j) \), we obtain a system of equations of the form \( h(cl_n(k)) = h(cl_n(i)) o^2 h(cl_n(j)) \). Now, \textsc{Maple} is asked to give a solution for the corresponding system of equations \( x_k = x_i o^2 x_j \) with respect to the modulo factor \( m \) using \textsc{Maple}'s function \texttt{msolve}, where \( h(cl_n(l)) \) becomes the variable \( x_l \). When \textsc{Maple} returns a solution for the variables containing only elements from the integer set corresponding to \( RS^2_m \), we have found a homomorphism between the structures. When there is a disjoint solution with \( x_i \neq x_j \), for all \( i \neq j \), we have a candidate for the isomorphism.

\footnote{Observe that we avoid confusion between indices and modulo factors by writing indices as superscripts, except in indexed variables such as \( x_i, y_j \) as they are clearly distinct from congruence classes of the form \( cl_i(x) \).}
As a simple example we consider the proof that $(\mathbb{Z}_2, +)$ is isomorphic to $(\mathbb{Z}_2, \lambda xy, x + y + 1_2)$. The possible pointwise functions $h : \mathbb{Z}_2 \rightarrow \mathbb{Z}_2$ are:

- $h_1(x) = \begin{cases} 0_2, & \text{if } x = 0_2 \\ 1_2, & \text{if } x = 1_2 \end{cases}$
- $h_2(x) = \begin{cases} 1_2, & \text{if } x = 0_2 \\ 0_2, & \text{if } x = 1_2 \end{cases}$
- $h_3(x) = \begin{cases} 0_2, & \text{if } x = 0_2 \\ 1_2, & \text{if } x = 1_2 \end{cases}$
- $h_4(x) = \begin{cases} 1_2, & \text{if } x = 0_2 \\ 0_2, & \text{if } x = 1_2 \end{cases}$

During the proof MAPLE is asked to give a solution for the equations $x_0 = x_0 + x_0 + 1$, $x_1 = x_0 + x_1 + 1$, $x_1 = x_1 + x_0 + 1$, $x_0 = x_1 + x_1 + 1$ with modulo factor 2 and returns $\{x_0 = 1, x_1 = x_1\}$. The solutions are analyzed by the hint system, and $x_0 = 1, x_1 = 0$ is suggested because it is both a disjoint solution and all elements are in the codomain. Therefore, $h_2(x)$ is inserted as the pointwise defined isomorphic function. The subsequent subproofs for the properties injectivity, surjectivity, and homomorphy of the pointwise defined function are then performed in the regular fashion of the TryAndError strategy as already discussed in Sec. 3: defined concepts are expanded, quantifiers are eliminated by introducing case splits for all possible values, and statements about residue classes are rewritten into statements about integers.

Proving the properties injectivity, surjectivity, and homomorphy with the TryAndError strategy has the complexity $n^2$ where $n$ is the cardinality of the structures involved. However, if no suitable hint can be computed there are $n^n$ pointwise defined functions to check, which becomes infeasible already for relatively small $n$.

**EquiSolve** During the isomorphism proof we have to show injectivity, surjectivity, and the homomorphism property for the introduced mapping. Doing so by a complete case analysis can become quite lengthy and therefore it is desirable to represent the isomorphism function in a more compact form. Often this can be realized by computing a polynomial that interpolates the pointwise defined function. If we can compute such an interpolation polynomial the EquiSolve strategy has a chance to find the subproofs for surjectivity and the homomorphism property. Namely these subproblems can then be reduced to equations which could be solvable with the Solve-Equation method. In the subproof for injectivity we have to show for each two distinct elements that their images differ, which cannot be concluded by equational reasoning.

For the construction of the interpolation polynomial we employ again MAPLE. However, we do not use one of the standard algorithms from the literature for interpolating sparse polynomials (see for example [20,19]), as they do not necessarily give us the best possible interpolation polynomial. Moreover, some of the implemented algorithms, for instance in MAPLE, do not always suffice for our purposes. We have thus decided to implement a simple search algorithm to

---

3 The proof of each of these properties results in formulas with two nested quantifications ranging over sets of cardinality $n$. This results into $n^2$ possible cases.

4 MAPLE’s algorithms interp and Interp cannot always handle the interpolation of functions where a non-prime modulo factor is involved.
find a suitable interpolation polynomial of minimal degree. This is feasible as we have to handle only relatively small mappings.

In detail, the interpolation proceeds as follows: Given a pointwise defined isomorphism function \( h: cl_n(x_i) \in RS_n^1 \rightarrow cl_m(y_i) \in RS_m^2 \), we let MAPLE solve the system of equations \((a_d x_i^d + \cdots + a_1 x_i + a_0) \mod m = y_i \mod m\) for all \( x_i, y_i \). When MAPLE returns a solution for \( a_d, \ldots, a_0 \) we have found an interpolating polynomial. If there is no solution, a polynomial with degree \( d + 1 \) will be sent to MAPLE. This procedure terminates at the latest when \( d = m - 1 \).

We illustrate the approach of the EquSolve strategy using once more the example of the proof that \((\mathbb{Z}_2, +)\) is isomorphic to \((\mathbb{Z}_2, \lambda x y x + y + 1_2)\). The corresponding pointwise isomorphism mapping is \( h(0_2) = 1_2, h(1_2) = 0_2 \) for which the interpolation polynomial \( x \rightarrow (x + 1 \mod 2) \) can be computed. This polynomial is introduced into the proof instead of the pointwise defined function. The properties of injectivity, homomorphy, and surjectivity, are then shown for the polynomial. During the subproofs of the latter two properties, the problem is reduced to an equation over integers that can be generally solved by the Solve-Equation method. As already mentioned the proof for injectivity cannot be constructed within the EquSolve strategy. Therefore, MULTI switches either to the strategy ReduceToSpecial or TryAndError to prove this property. How the former is applied in this context is described in the next paragraph.

The success of EquSolve depends on the capability of MAPLE. Often equations in isomorphism proofs contain terms with different modulo factors nested inside, resulting from the mapping between residue class sets \( RS_n \) and \( RS_m \) with \( n \neq m \), which are not solvable by MAPLE. So EquSolve is limited to proofs for residue class sets with the same modulo factor.

ReduceToSpecial  The strategic control rules in MULTI specify that on residue class problems the strategies ReduceToSpecial, EquSolve, and TryAndError are always tested in this order. This holds for isomorphism or non-isomorphism problems as well as for possible arising subproblems such as to show injectivity, surjectivity, or homomorphy. For instance, if EquSolve can introduce a suitable polynomial function but fails to prove the arising injectivity, surjectivity, or homomorphy subgoals, MULTI has to deal with those subproblems again on the strategy level. Since we do not have theorems to handle isomorphism problems in general, ReduceToSpecial is not applicable to the original theorem, but it comes into play when a subgoal, in particular the injectivity subgoal, has to be proved. Here we can exploit the simple mathematical fact that in finite domains surjectivity implies injectivity and vice versa with the following theorem: A surjective mapping between two finite sets with the same cardinality is injective.

Thus, the proof of injectivity consists only of the application of this theorem, if we can prove that our mapping is surjective. Hence, the idea for the most efficient isomorphism proofs is to start with EquSolve on the whole isomorphism problem, prove the surjectivity and homomorphy subproblem, if possible, with equational reasoning, and, since EquSolve will always fail on the injectivity subgoal, to let ReduceToSpecial finish the proof.
4.2 Non-isomorphism Proofs

During the classification process it is also necessary to prove that two given structures are not isomorphic. To discharge this proof obligation we use again the already introduced strategies ReduceToSpecial and TryAndError. While the latter can be applied independently by performing the usual exhaustive case analysis, the former is combined with TryAndError to deduce non-isomorphism of two structures by showing the existence of substructures of different order. Additionally, we have implemented another strategy, NotInjNotIso, which is specialized on proving non-isomorphism problems. It constructs an indirect proof by showing that no homomorphic mapping between the two given residue class structures can be injective.

TryAndError Proving that two structures are not isomorphic results in proving a universally quantified goal: MULTI has to prove that each possible mapping between the two structures involved is either non-injective, non-surjective, or non-homomorphic. When dealing with universally quantified statements the TryAndError strategy performs an exhaustive case split on all possible instantiations. As described in Sec. 4.1 all possible mappings can be presented by pointwise defined functions. Hence, in the case of non-isomorphism proofs the top-most case split in the TryAndError strategy is on every possible pointwise defined function and for each function a subproof is constructed, to show that it is either non-injective, non-surjective, or non-homomorphic. The subproofs are constructed with an exhaustive case analysis similar to the proofs of the simple properties in Sec. 3.

The application of this naïve technique suffers from combinatorial explosion on the possibilities for the pointwise defined function. For two structures whose sets have cardinality $n$ we have to consider $n^n$ different possible functions. Thus, in practice this strategy is not feasible if structures of cardinality larger then four are involved. Despite this fact the strategy is our fallback if the other techniques presented in the sequel should fail.

ReduceToSpecial If two structures are isomorphic, they share the same algebraic properties. Thus, in order to show that two structures are not isomorphic it suffices to show that one particular property holds for one structure but not for the other. In this paragraph we discuss two such properties and explain how MULTI combines the strategies ReduceToSpecial and TryAndError to establish that two structures are not isomorphic. Thereby ReduceToSpecial contains theorems that can reduce the original goal to subgoals stating that a property does not hold for one structure whereas it holds for the other structure. These subgoals can then be proved with TryAndError.

First we introduce the concepts order, trace, and order of the trace of elements of a structure $(S, \circ)$, where $S$ is a finite set:
An element \( a \in S \) has the order \( n \) if \( n \in \mathbb{N} \) is the smallest positive integer such that \( a^n = e \), where \( e \in S \) is the unit element with respect to \( \circ \). In the following we write this as \( \text{order}(a) \).

The trace of an element \( a \in S \) is the set \( \{a^n | n \in \mathbb{N}\} \). The cardinality of this set is referred to as the order of the trace of \( a \). This is written as \( \text{ordertr}(a) \) in the following.

The latter concept is a generalization of the former so we can also deal with elements that do not have an order or with structures that do not have a unit element. Note also, that both the order of an element \( a \) and the order of its trace always range between 1 and the cardinality of \( S \).

For two structures \((S^1, \circ^1)\) and \((S^2, \circ^2)\) we know that if they are isomorphic then for each element \( a_1 \in S^1 \) with order \( n \) there exists an element \( a_2 \in S^2 \) with the same order. Moreover, we know an analogous statement for the order of the traces. Thus, to prove that two structures are not isomorphic it is sufficient to prove that one structure contains an element \( a_1 \) such that the other structure contains no element \( a_2 \) whose order (order of the trace) is equal to the order (order of the trace) of \( a_1 \). This can be formalized in the following theorems, where \([1, \text{card}(S^1)]\) denotes the integer interval from 1 to the cardinality of \( S^1 \):

\[
- (\exists n: [1, \text{card}(S^1)]) \land (\exists x_1: S^1) \cdot \text{order}(x_1, S^1, \circ^1) = n \land \\
(\neg \exists x_2: S^2 \cdot \text{order}(x_2, S^2, \circ^2) = n)) \Rightarrow \neg \text{iso}(S^1, \circ^1, S^2, \circ^2)
\]

\[
- (\exists n: [1, \text{card}(S^1)]) \land (\exists x_1: S^1) \cdot \text{ordertr}(x_1, S^1, \circ^1) = n \land \\
(\neg \exists x_2: S^2 \cdot \text{ordertr}(x_2, S^2, \circ^2) = n)) \Rightarrow \neg \text{iso}(S^1, \circ^1, S^2, \circ^2)
\]

The ReduceToSpecial strategy can apply these two theorems to reduce non-isomorphism goals and then TryAndError takes over to complete the proof. For instance, the application of the second theorem to the problem to prove that the two Abelian semi-groups \((\mathbb{Z}_4, \lambda xy, x \ast y = \overline{2}_4)\) and \((\mathbb{Z}_4, \lambda xy, \overline{2}_4)\) are not isomorphic results in the problem to prove that there exists an integer \( n \) such that: (1) there exists an \( x_1 \in \mathbb{Z}_4 \) such that the cardinality of the trace of \( x_1 \) with respect to the first operation \( \lambda xy, x \ast y = \overline{2}_4 \) equals \( n \), (2) for all \( x_2 \in \mathbb{Z}_4 \) the cardinality of the trace of \( x_2 \) with respect to the second operation \( \lambda xy, \overline{2}_4 \) is not \( n \). Since the order of the trace can be at most the cardinality of the involved set \( \mathbb{Z}_4 \), the possible values for \( n \) can be restricted to 1, 2, 3, and 4. Since \( n \) ranges also over a finite set we can apply the TryAndError strategy to prove this goal. To restrict the search, Multi obtains hints for suitable instantiations for \( n \) and \( x_1 \). The hints are computed by constructing the traces with GAP. In our example the suitable instantiations are \( n=3 \) and \( x_1 = \overline{1}_4 \) (the trace of \( \overline{1}_4 \) in \((\mathbb{Z}_4, \lambda xy, x \ast y = \overline{2}_4)\) is \( \{\overline{1}_4, \overline{2}_4, 0_4\} \)) since the traces of all elements of \((\mathbb{Z}_4, \lambda xy, \overline{2}_4)\) are either of order 1 or 2.

In contrast to employing TryAndError alone, proofs constructed with the combination of TryAndError and ReduceToSpecial have only polynomial complexity in the cardinality of the involved sets. Moreover, the search is reduced significantly by providing hints. But this technique is only applicable when structures involved contain elements suitable for our purpose in the sense that either
their order or the order of their trace is not reflected in the respective other structure.

**NotInjNotIso** The strategy NotInjNotIso was particularly implemented for non-isomorphism proofs. Its idea is to construct an indirect proof, which shows that two structures \( (S^1, o^1) \) and \( (S^2, o^2) \) are not isomorphic. We first assume that there exists a function \( h: S^1 \to S^2 \) which is an isomorphism. Then \( h \) is an injective homomorphism. The strategy NotInjNotIso tries to find two elements \( c_1, c_2 \in S^1 \) with \( c_1 \neq c_2 \) such that we can derive the equation \( h(c_1) = h(c_2) \). This contradicts the assumption of injectivity of \( h \) where \( h(c_1) \neq h(c_2) \) has to hold if \( c_1 \neq c_2 \). Note, that the proof is with respect to all possible homomorphisms \( h \) and we do not have to give a particular mapping.

We explain the NotInjNotIso strategy for our example that \((\mathbb{Z}_4, \lambda xy_s^2)\) is not isomorphic to \((\mathbb{Z}_4, \lambda xy_s x^s y^s 2_4)\). The strategy first constructs the situation for the indirect argument. From the hypothesis that the two structures are isomorphic follow the two assumptions that there exists a function \( h \) that is (1) injective and (2) a homomorphism. By the first assumption a contradiction can be concluded, when we are able to show that \( h \) is not injective.

MULTI continues by applying a method to the second assumption, that introduces the homomorphism equation \( h(x o^1 y) = h(x) o^2 h(y) \) instantiated for every element of the domain as new assumptions. In the above example 16 equations like

\[
h(\bar{0}_4) = 2_4 \quad \text{for } x = \bar{0}_4, y = \bar{0}_4, \quad h(\bar{2}_4) = 2_4 \quad \text{for } x = \bar{1}_4, y = \bar{1}_4, \quad \ldots
\]

are introduced, where the actual operations \( o^1 = \lambda xy_s x^s y^s 2_4 \) and \( o^2 = \lambda xy_s 2_4 \) are already applied to the given arguments.

From the introduced system of equations the NotInjNotIso strategy tries to derive that \( h \) is not injective. To prove this we have to find two witnesses \( c_1 \) and \( c_2 \) such that \( c_1 \neq c_2 \) and \( h(c_1) = h(c_2) \). In our example \( \bar{0}_4 \) and \( \bar{2}_4 \) are chosen for \( c_1 \) and \( c_2 \), respectively, which leads to \( h(\bar{0}_4) = h(\bar{2}_4) \). This goal is transformed into an equation that can be solved in a general way, by successively applying equations from the equation system. In our example \( h(\bar{0}_4) = h(\bar{2}_4) \) is reduced to \( \bar{2}_4 = \bar{2}_4 \) by two substitutions using the equalities given above. The last equation is justified by the reflexivity of equality and the proof is finished.

In order to restrict the search for appropriate \( c_1 \) and \( c_2 \) NotInjNotIso employs a control rule to obtain a hint. The control rule calls MAPLE to compute all possible solutions for the system of instantiated homomorphism equations with respect to the corresponding modulo factor using MAPLE's function msolve. Then the solutions are checked for whether there is a pair \( c_1 \) and \( c_2 \) such that in all of the solutions \( h(c_1) = h(c_2) \) holds. If there is such a pair it is provided has a hint. Although the control rule cannot always come up with a hint, our experiments have shown that the NotInjNotIso strategy is also often successful when no hint can be computed.

In our example the equational reasoning involved is still relatively simple and could be done by a more specialized system such as a term rewriting system. However, this is not possible in the general case. Then the equations contain
more complex terms involving addition, multiplication, and subtraction of constant congruence classes of the form \( h(cl_n(i)) \) and thus additionally have to be performed with respect to the correct modulo factor. The solution of the equations is therefore beyond the scope of any term rewriting system but requires symbolic computation.

\textsc{NotInjNotIso} can produce very short proofs even for structures with large sets. However, to construct an appropriate sequence of equality substitutions is generally the hard part of proofs with \textsc{NotInjNotIso}. In fact, for problems with the same complexity (i.e., problems involving structures of the same cardinality) the length of the proofs can vary drastically. Moreover, the equational reasoning process does not have to terminate. Therefore, we experimented with randomization and restart techniques known from Artificial Intelligence \cite{6}. It turned out, that the introduction of a stochastic element when choosing the next instantiated homomorphism equation to apply, coupled with restarts based on statistical measures (i.e., a proving attempt is interrupted after a certain time interval and a new proving attempt is started instead) can significantly increase the efficiency and robustness of proof planning with the \textsc{NotInjNotIso} strategy. A complete description of the performed experiments as well as how randomization and restarts are realized in \textsc{Multi} can be found in \cite{11}.

\textsc{ReduceToSpecial} is the first strategy that is tried when automatically discharging non-isomorphism proof obligations. If it fails first the \textsc{NotInjNotIso} strategy is tried before \textsc{TryAndError}. The \textsc{EquSolve} strategy is not applicable to non-isomorphism problems.

5 Experiments and Results

The proving techniques presented in this paper mainly build on the strategies already constructed for the proofs of simple properties of the residue class structures as presented in \cite{13}. There we used a total of 21 examples to construct the basic versions of the \textsc{ReduceToSpecial}, \textsc{TryAndError}, and \textsc{EquSolve} strategies. To develop the extensions of these strategies to handle isomorphism and non-isomorphism proofs we used 15 examples and another 4 examples to build the \textsc{NotInjNotIso} strategy.

To show the validity of the techniques for isomorphism and non-isomorphism proofs we applied our classification process to 8128 structures with the set \( \mathbb{Z}_6 \). Here, we found 4152 magmas, 73 Abelian magmas, 1114 semi-groups, 1025 Abelian semi-groups, 738 quasi-groups, 257 Abelian quasi-groups, 50 Abelian monoids, and 419 Abelian groups. For the quasi-groups and the Abelian quasi-groups we found that they belong to two different classes, respectively. All Abelian monoids and the Abelian groups belong to the same class, respectively. Furthermore, we found 12 non-isomorphic classes of Abelian semi-groups, eight classes of semi-groups, five classes of Abelian magmas, and seven classes of magmas. 90% of the necessary isomorphism proofs were done with the \textsc{EquSolve} strategy, the other 10% were done with \textsc{TryAndError}. During the automatic classification 121 non-isomorphism proofs were constructed. Here 80% of the proofs
were done with the `NotInjNotIso` strategy and the remaining 20% with the combination of `TryAndError` and `ReduceToSpecial`. In addition to the automatic classification process we did separate experiments with 800 non-isomorphism proofs to obtain suitable cutoff values (i.e., when to restart the `NotInjNotIso` strategy) by analyzing the search spaces \[11\].

### 6 Related Work and Conclusions

We have presented an experiment in exploring properties of residue classes over the integers with the combined effort of the multi-strategy proof planner `MULTI` and the two computer algebra systems `Maple` and `GAP`. In our experiments we classify residue class sets over the integers together with binary operations in terms of what algebraic structure they form and then we divide structures of the same algebraic category into isomorphism classes. Arising proof obligations are discharged by `MULTI` with several strategies that realize different proving techniques of the problem domain. The proof planning in our problem domain benefits considerably from the possibilities `MULTI` provides. Using `MULTI` we were not only able to encode several different proving techniques in a conceptually clear way into different strategies, but could also combine and interleave these strategies flexibly. We employed the computer algebra systems to guide and simplify both the classification and the proof planning process. We have tested the validity of our techniques with a large number of experiments. It turned out that the implemented machinery is not only robust but that the elaborate strategies are successful on a large number of examples. Overall a considerable part of the problems have been proved with various usages of computer algebra. Although not explicitly mentioned in this paper we can also deal with direct products of residue class sets and classify structures with two operations \[12\].

There are various accounts on experiments of combining computer algebra and theorem proving in the literature (see \[8\] for just a few). However, they generally deal with the technical and architectural aspects of those integrations as well as with correctness issues and not with the application of the combined systems to a specific problem domain. A possibly fruitful cooperation between the deduction system Nuprl and the computer algebra system Weyl in the domain of abstract algebra is sketched in \[7\]. Our paper in contrast presents the application of an already existing combination of proof planning and computer algebra to a specific problem domain. We thereby exploit work previously done in \textsc{Omega} \[9,18\].

More concrete work in exploration in finite algebra is reported in \[4,10,17\] where model generation techniques are used to tackle quasi-group existence problems. In particular, some open problems in quasi-group theory were solved. The motivation for all this work is roughly to specify certain properties of an algebra and then to try to automatically construct a structure that satisfies the required properties. Thus, the constructed algebra might actually be a new discovery. Our work is diametrical in the sense that we start out with given structures and classify them with respect to their algebraic properties and whether they
are isomorphic. Likewise, our automatic exploration processes depend on sets of pre-constructed residue class sets and operations. In addition both classification and exploration is currently not designed to intentionally discover new algebraic structures.

References

CAL: A Computer Assisted Learning System for Computation and Logic

Masahiko Sato¹, Yukiyoshi Kameyama², and Takeuti Izumi¹

¹ Graduate School of Informatics, Kyoto University
{masahiko,takeuti}@kuis.kyoto-u.ac.jp
² Institute of Information Sciences and Electronics, University of Tsukuba
kam@is.tsukuba.ac.jp

Abstract. In this paper, we describe a computer system CAL which assists students’ understanding of a lecture course in logic and computation. We introduce a new foundational theory of expressions, which is inspired by the theory of expressions due to Martin-Löf [6], and use the expressions determined by the theory to implement various syntactic objects in CAL. We also introduce the concept of derivation game, and show that various logical and computational systems given in the lecture course can be defined uniformly by derivation games.

1 Introduction

In this paper, we describe a computer assisted learning system for computation and logic. The system is intended to complement a lecture course on Computation and Logic given to undergraduate students of Kyoto University. All the students are asked to use the system in order to deepen their understanding of the contents of the lecture. The lecture course is an introduction to formal logics. Since most of the students are familiar with a number of programming languages but not familiar with formal logics, we designed our lecture course by stressing the similarity between programming and proving. We used the Curry-Howard isomorphism to achieve this design goal.

Traditionally, courses in logic are taught in class rooms and exercises are done by paper and pencil. However, in view of the facts that the subject matter of the course consists of formal syntactic entities which can be manipulated formally, we think that it is important and necessary to provide a computer environment in which students can solve problems and the computer can tell if the students solved the problems correctly. Although there are already several computer software like Tarski’s World [2] for learning logic, we have designed and implemented our own system since existing software does not cover the materials we wish to teach.

The name of the lecture course is “Computation and Logic” and it covers the following formal systems.

1. Intuitionistic propositional logic.
2. Simply typed λ-calculus.
3. Reduction of derivations in the intuitionistic propositional logic and terms in the simply typed $\lambda$-calculus.
4. Heyting arithmetic.
5. Typed $\lambda$-calculus with dependent types.
6. Reduction of derivations in the Heyting arithmetic and terms in the dependently typed $\lambda$-calculus.

The name of the computer environment for the course is CAL. CAL implements all the formal systems we teach in the course. Therefore students can use CAL to play with all the systems listed above. CAL is entirely implemented in Emacs Lisp and runs under GNU Emacs. GNU Emacs is a very powerful text editor and it has its own programming language Emacs Lisp which we used to implement CAL. A student can invoke CAL from within Emacs, and when CAL is invoked, a special buffer is created and then the user and the CAL system can interact with each other in this buffer. As a system, CAL consists of the following three parts.

1. User interface
2. Parser
3. Checker

We will explain each part in more detail in section 4.

The fundamental purpose of our lecture course is to convey students the facts that formal systems taught at the course are, or at least should be, reflections of the informal logical or mathematical systems they have already learned in (informal) mathematics. Therefore, we tried to reflect the informal mathematical practices we carry out in daily mathematics as naturally and precisely as possible. This approach forced us to formally present several mathematical practices that are usually left at the informal meta level. For example, in traditional logic text books, the concept of abbreviations are only informally sketched as a meta level technical means to simplify notations for formal syntactic objects like terms or formulas. In our course we have defined abbreviations formally so that students can actually check on a computer whether they understand what expressions abbreviate what expressions.

For this lecture we prepared both the course note which describes the formal systems given above and the software system CAL which implements these formal systems. Although these formal systems are standard ones, we have fine tuned them so that they can be presented uniformly and also they can be smoothly manipulated formally on a computer. In particular, to achieve the fundamental purpose mentioned above, we have identified informal logic and mathematics as human mental activities to arrive at sound logical or mathematical judgments, and we formalized judgments as formal expressions which we can derive by constructing derivations whose conclusions are the given judgments. Each formal system we introduce in the course has its own unique form of judgments and derivations, but they are presented in a uniform way as we will describe below.
In order to achieve the uniform presentations of the formal systems, we use two general defining mechanisms to define formal systems. The first one is what we call definition games. Definition games are formalizations of a simple class of informal inductive definitions commonly used in mathematics. Each definition game, say $G$, defines a class of $G$-derivations and a class of $G$-judgments. A $G$-judgment $J$ is said to be provable (or, derivable) if there is a $G$-derivation whose conclusion is $J$. For example, in NatNum, which is the first definition game we teach in our course, we can derive the NatNum-judgment $s(s(0)) : \text{Nat}$ as follows.

\[
\begin{align*}
0 & : \text{Nat} \\
\text{succ } s(0) & : \text{Nat} \\
\text{succ } s(s(0)) & : \text{Nat}
\end{align*}
\]

The second defining mechanism is called derivation games. While definition games are used to derive judgments without hypotheses, derivation games are used to derive hypothetical judgments which are judgments that depend on several hypotheses. Derivation games have built-in mechanisms to introduce and discharge hypotheses. Using derivation games we can define several natural deduction systems in a uniform way.

The paper is organized as follows. In section 2, we explain expressions which provide us with an abstract way of uniformly presenting syntactic objects we teach in the course. The same abstract syntax is used to implement the CAL system, so that we do not have the discrepancy usually found in the implementations of syntactic objects using de Bruijn notation. In section 3, we define definition games and derivation games, and give some examples of them. Section 4 explains the CAL system from the implementation point of view, and section 5 concludes the paper.

2 Theory of Expressions

In informal mathematics, we often explain concepts using “dots”. For example:

Natural numbers are defined as 0, $s(0)$, $s(s(0))$, $\cdots$.

While this description helps readers’ intuitive understanding, it does not constitute a precise definition, and there is always possibility of misunderstanding by the readers. In fact, students can easily get lost in manipulating formal objects, and the reason is often that they could not understand the definitions correctly since they were presented to the students in an informal and ambiguous way. In particular, we believe that the concepts of variable binding and substitution are the most fundamental concepts we have to teach in any logic course; however, they are rarely taught rigorously for the technical reasons we explain later in this section.

Based on this observation, we had decided to design our lecture course so that each and every mathematical concept will be presented to students in a
rigid and formal manner. A trade-off of the decision is that we must use a great number of formal inductive definitions when giving each formal system, and we were naturally led to introduce two definition mechanisms of formal definitions and a theory of expressions which provides us with a natural and convenient data structure for defining the concepts of variable binding and substitution.

The two definition mechanisms we introduced are definition games and derivation games. Definition games are used to define operations and relations on syntactic objects in a manner similar to that of structured operational semantics. For example, expressions and substitution operation on expressions are defined by the definition games $\text{Exp}$ and $\text{Subst}$, respectively. Derivation games are used to define various logical systems and we will explain them in the next section.

The above design decision enabled us to present various formal systems uniformly and naturally. This design decision also brought us the following good side-effects. Namely, the course note we prepared served not only as a text book but also both as the user’s manual for students when they use the CAL system and as the formal specification of the CAL system which had to be implemented by us.

### 2.1 Definition Games

A definition game $G$ consists of a finite number of rule schemata. Each schema is of the following form where $k$ is a non-negative integer, $C, P_1, \ldots, P_k$ are expressions possibly containing schematic variables ranging over sequences of words, and $N$ is a word:

$$
\frac{P_1 \ \cdots \ \ P_k}{C} \ N
$$

Here, $N, C, P_i$ are called the name, the conclusion, and the assumptions of the rule schema, respectively. For example, the definition game $\text{NatNum}$ is determined by the following two rule schemata, where $n$ is a schematic variable.

$\text{NatNum}$

<table>
<thead>
<tr>
<th>Rule Schemata</th>
<th>Name</th>
<th>Conclusion</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 : \text{Nat}$</td>
<td>zero</td>
<td>$n : \text{Nat}$</td>
<td>$s(n) : \text{Nat}$</td>
</tr>
</tbody>
</table>

Given a definition game $G$, a $G$-derivation and its conclusion is defined as follows. Let $D_1, \ldots, D_k$ ($k \geq 0$) be $G$-derivations whose conclusions are $P_1, \ldots, P_n$ respectively and

$$
\frac{P_1 \ \cdots \ \ P_k}{C} \ N
$$

be an instance of a rule schema of $G$, then

$$
\frac{D_1 \ \cdots \ \ D_k}{C} \ N
$$

is a $G$-derivation whose conclusion is $C$. As an example, we already gave a derivation of $s(s(0)) : \text{Nat}$ in the introduction.

---

1 We assume that “sequences of words” are appropriately defined.
2.2 Expressions

In CAL, we introduce the concept of expression to uniformly represent formal syntactic objects we teach. The set of expressions is freely generated from constants and variable references by means of the two operations of list formation (list(\_)) and abstraction (abs\(_x\)(\_)) of an expression with respect to a variable \(x\).

Two expressions are defined to be definitionally equal if they are generated in exactly the same way by these operations. We write \(E \equiv F\) if \(E\) and \(F\) are definitionally equal. Thus, if \(E_1, \ldots, E_n\) and \(F_1, \ldots, F_m\) are expressions, then \(\text{list}(E_1, \ldots, E_n) \equiv \text{list}(F_1, \ldots, F_m)\) iff \(m = n\) and \(E_i \equiv F_i\) for each \(i\) (\(1 \leq i \leq m\)), and if \(x, y\) are variables and \(E, F\) are expressions, then \(\text{abs}_x(E) \equiv \text{abs}_y(F)\) iff \(x \equiv y\) and \(E \equiv F\).

We can then define the operations of substitution (\(\text{subst}(\_, \_, \_)\)) and instantiation (\(\text{inst}(\_, \_)\)) in such a way that they enjoy, among others, the equality:

\[
\text{inst}(\text{abs}_x(E), F) \equiv \text{subst}(F, x, E).
\]

Existing methods of constructing a structure on which abstraction and substitution operations are properly defined are not at all easy and therefore very difficult to teach. For example, the classical way of teaching the syntax and the \(\beta\)-reduction rule of the untyped \(\lambda\)-calculus is like this. First, expressions are introduced as strings of characters, where a character is either a variable or a constant like \('\text{\lambda}'\), and a string is of the form \(c_1 \cdots c_n\) (\(n \geq 0\)) where each \(c_i\) is a character. Two strings \(c_1 \cdots c_n\) and \(d_1 \cdots d_m\) are defined to be definitionally equal if \(m = n\) and \(c_i\) and \(d_i\) are the same characters. The concatenation (\(\text{conc}(\_, \_)\)) of two expressions is defined by juxtaposition and \(\text{conc}(E, F)\) is simply written as \(EF\). The abstraction operation is then defined by putting \(\text{abs}_x(E) \equiv \text{conc}(x, \text{conc}(\text{'}\text{\lambda}', E)) \equiv x.E\). The operation of substituting \(F\) for \(x\) in \(E\) is defined as the operation of textual replacement of each free occurrence of \(x\) in \(E\) with \(F\). But, this operation works well only when no free variables in \(E\) become bound after the substitution. It is therefore necessary to re-define the definitional equality relation by introducing the concept of \(\alpha\)-equivalence. Even if we could successfully introduce the syntax in this way, we must continue to use concrete syntax when we actually write down \(\lambda\)-terms. Therefore, we must define substitution operation on terms in concrete syntax, but this definition requires us to check that the operation respects \(\alpha\)-equivalence.

There are other methods by de Bruijn and by us. They both remove the variable \(x\) from \(E\) when \(\text{abs}_x(E)\) is constructed and do not rely on the concept of \(\alpha\)-equivalence when defining expressions. But these methods are not suitable for teaching for the following reasons. First of all, if we write concrete \(\lambda\)-terms using these notations, then they are almost unreadable. We can make them readable again by introducing traditional syntax with the definitional equality as strings and define a translation mapping from the terms in traditional syntax

\(^2\) We will explain variable references later.

\(^3\) The second operation is actually a schema of operations since it contains \(x\) as a parameter.
to the terms in de Bruijn notation or in our notation \[7,9\]. But, if we do so, we must again define substitution operation on terms written in concrete syntax and must show that the operation is well-defined.

Although de Bruijn notation is not suitable for humans, it is often used in the implementations of logical systems because of the simplicity of the substitution operation. But such implementations have the following drawback. When the system prints out an expression, it must use the external syntax, but since bound variables have no names in de Bruijn notation, the system must choose an arbitrary name like X3141.

Our approach uses the expressions which we informally introduced at the beginning of this subsection. It is a hybrid of the traditional approach and de Bruijn’s approach and it inherits only good points of these approaches. That is, our syntax is human readable and substitution can be defined without relying on the concept of \(\alpha\)-equivalence. Moreover, unlike de Bruijn notation, \(\lambda\)-terms in traditional syntax can be expressed in our syntax without any modifications. Because of these good properties, our expressions are implemented by the CAL system as Lisp symbolic expressions, and the implementation establishes an isomorphism between expressions we teach and their representations by Lisp symbolic expressions. So, we could implement substitution on a computer in an isomorphic way as we teach substitution in the lecture course.

Having this mechanism in mind, we can define expressions and substitution over them as definition games. We first assume that there are countably many variables denoted by \(x, y, z\) (possibly with subscripts) and constants denoted by \(c\). A variable reference is a pair \((n, x)\) of a natural number \(n\) called reference counter and a variable \(x\), and we will write \(\#^nx\) for the variable reference \((n, x)\). A reference counter plays a similar role as a de Bruijn index. A declaration is a finite sequence of variables \(x_1, \ldots, x_n (n \geq 0)\) denoted by \(\Gamma\) where duplication of a variable is allowed.

The definition game \(\text{Exp}\) consists of the following four rule schemata:

\[
\begin{align*}
\Gamma \exp \#^nx & \quad \text{var} \\
\Gamma \exp c & \quad \text{const} \\
\Gamma \exp E_1 \quad \cdots \quad \Gamma \exp E_n & \quad \text{struct} \\
\Gamma, x \exp E & \quad \text{abs}
\end{align*}
\]

where in the var rule, \(n\) is a non-negative integer and \(\Gamma\) must contain at least \(n + 1\) occurrences of the variable \(x\). In the var rule, \(\#^nx\) refers to the \(n + 1\)-st occurrence of \(x\) in \(\Gamma\) counting from the right. We used dots in the struct rule schema, but since the meaning of these dots can be explained rigorously by induction on \(n\), the usage is permitted. If there is an Exp-derivation whose conclusion is \(\Gamma \exp E\), we say that \(E\) is an expression under \(\Gamma\). Note that in this case any variable reference \(\#^nx\) occurring in \(E\) refers either to a unique occurrence of \(x\) in \(\Gamma\) or to the parameter \(x\) of a unique binder \(\text{abs}_x(\omega)\) that contains the \(\#^nx\) in its scope. We say that the occurrence of \(\#^nx\) in \(E\) is free in the first case, and bound in the second case.

Similarly, substitution can be defined by a definition game. However, due to lack of space, we explain substitution here informally. Let \(E\) be an expression
under $\Gamma$, and $\Delta$ be a declaration. Then, we can construct an expression $F$ under $\Gamma, \Delta$ as follows. For each occurrence of variable references $\#^n x$ in $E$, we compute a non-negative integer $k$ as follows. If the occurrence is bound in $E$, then we put $k = 0$. If it is free in $E$, then we put $k$ to be the number of occurrences of $x$ in $\Delta$. We then replace the occurrence of $\#^n x$ in $E$ with $\#^{n+k} x$. $F$ is obtained from $E$ in this way. It is easy to check that $F$ is an expression under $\Gamma, \Delta$. We will call $F$ the result of pushing $E$ through $\Delta$, and write $E \uparrow \Delta$ for $F$. For example, if $x$ and $y$ are distinct variables and $E \equiv \text{abs}_x(\text{list}(\#^0 x, \#^1 x, \#^2 x, \#^0 y))$, then $E$ is an expression under $x, x, y$ and we have:

$$F \equiv E \uparrow x, x \equiv \text{abs}_x(\text{list}(\#^0 x, \#^3 x, \#^4 x, \#^0 y)),$$

which is an expression under $x, x, y, x, x$.

Now, given a variable reference $\#^n x$ and expressions $E$ and $F$, we define an expression $[\#^n x := F](E)$ inductively as follows.

1. $[\#^n x := F](\#^m y) \equiv \begin{cases} 
\#^m y & \text{if } y \neq x, \\
\#^m y & \text{if } y \equiv x \text{ and } m < n, \\
F & \text{if } y \equiv x \text{ and } m = n, \\
\#^{m-1} y & \text{if } y \equiv x \text{ and } m > n.
\end{cases}$

2. $[\#^n x := F](c) \equiv c$.

3. $[\#^n x := F](\text{list}(E_1, \ldots, E_m)) \equiv \text{list}(\#^n x := F](E_1), \ldots, \#^n x := F](E_m))$.

4. $[\#^n x := F](\text{abs}_y(E)) \equiv \text{abs}_y([\#^n x \uparrow y := F \uparrow y](E))$.

We define the substitution operation by putting $\text{subst}(F, x, E) \equiv [\#^0 x := F](E)$. For example, we have:

$$\text{subst}(\#^1 y, x, \text{abs}_x(\text{list}(\#^0 x, \#^1 x))) \equiv \text{abs}_x(\text{list}(\#^0 x, \#^1 y))$$

and

$$\text{subst}(\#^2 y, x, \text{abs}_x(\text{list}(\#^0 x, \#^1 x))) \equiv \text{abs}_x(\text{list}(\#^0 x, \#^3 y)).$$

Note that, if there are no name clashes in variables (as is usual when we write mathematical expressions), all variable references are in the form $\#^0 x$, so we do not have to care about the reference counters.

In the concrete syntax of CAL, we write $\langle E_1, \ldots, E_n \rangle$ for list($E_1, \ldots, E_n$) and $(x)[E]$ for $\text{abs}_x(E)$, and $\#^n x$ is written by prefixing $n$ #’s in front of $x$.

## 3 Derivation Games

Derivation games are the second defining mechanism in CAL, and are used to derive hypothetical judgments. As advocated in Martin-Löf’s type theory, we view a derivation in object logics as a derivation of hypothetical judgements, hence, all object logics (and computational systems) in CAL will be defined by derivation games. This applies not only to the derivability but also the formations of propositions and terms in arithmetic.

In this section we first introduce the general mechanism of derivation games, and after that we quickly overview the definitions of several logical systems in terms of derivation games.
3.1 Derivation Games in General

A hypothesis is an expression \( x : E \) where \( x \) is a variable and \( E \) is an expression. A hypothesis sequence, denoted by \( \Gamma \), is a finite sequence of hypotheses. Then a hypothetical judgment is an expression \( \Gamma \vdash E \) where \( E \) is an expression.

A derivation rule is in the following form:

\[
\frac{E_1 \ldots E_n}{E} \quad R
\]

where \( E, E_1, \ldots, E_n \) are expressions and \( R \) is a constant.

A derivation game \( G \) is determined in two steps. First, we specify a set of hypotheses in this derivation game, which we call \( G \)-hypotheses. Then, we specify, for each \( G \)-hypothesis-sequence, a set of derivation rules which can be used in this derivation game.

One may consider this definition is too complex, but the dependency of the second step on \( G \)-hypothesis-sequence cannot be eliminated in our setting. Let us take a simple example. Suppose we want to define the notion of provability of propositions in propositional logic as a derivation game. Then the hypotheses we may use in this game are either \( A : \text{Prop} \) (\( A \) is a proposition) or \( x : P \) (\( x \) is a proof of \( P \)). Under the hypothesis sequence \( \Gamma \equiv A : \text{Prop}, x : A \), we can apply the implication-introduction rule to prove \( A \supset A \), but not \( B \supset A \), since the latter is not a proposition under \( \Gamma \). So the applicability of a derivation rule depends on the hypothesis sequence under which we apply the rule.

Given a derivation game \( G \), a \( G \)-derivation under a \( G \)-hypothesis-sequence and its conclusion are defined by the following three clauses:

1. If \( x : E \) is in \( \Gamma \), then \( x \) is a \( G \)-derivation under \( \Gamma \), and its conclusion is \( E \).

2. If

\[
\frac{E_1 \ldots E_n}{E} \quad R
\]

is a derivation rule in \( G \), and \( D_i \) is a \( G \)-derivation under \( \Gamma \) whose conclusions are \( E_i \) (for \( 1 \leq i \leq n \)), then

\[
\frac{D_1 \ldots D_n}{E} \quad R
\]

is a \( G \)-derivation under \( \Gamma \) whose conclusion is \( E \).

3. If \( D \) is a \( G \)-derivation under \( \Gamma, \Delta \) whose conclusion is \( E \), then

\[
(\Delta)[D]
\]

is a \( G \)-derivation under \( \Gamma \) and its conclusion is \( (\Delta)[E] \).

\[4\] This is actually an abbreviation of the expression list(‘;’\( x, E \)) where ‘;’ is a constant. We will use similar abbreviations without explaining what expressions they will stand for, since such details will not concern us. In CAL, even abbreviations are formally defined using definition games.

\[5\] We cannot show that \( B : \text{Prop} \) from \( \Gamma \).
The first clause is a start-rule like one in the natural deduction style logic. Namely, assuming \( A \), we have a proof \( A \) with conclusion \( A \). The second is an ordinary inference rule. The third is a rule for discharging hypotheses. We shall show examples to illustrate their usage in the following subsections.

If there is a \( G \)-derivation \( D \) under \( \Gamma \) whose conclusion is \( E \), then we say that \( D \) is a derivation of the hypothetical judgment \( \Gamma \vdash E \) in \( G \).

### 3.2 Propositional Logic

To define propositional logic, we need two derivation games, \( \text{PropFrmn} \) for the formation of propositions and \( \text{PropProof} \) for the provability of propositions.

In the derivation game \( \text{PropFrmn} \), an expression of the form \( A : \text{Prop} \) is allowed as a hypothesis where \( A \) is a variable, and \( \text{Prop} \) is a constant. Instead of enumerating all the rules of this game, we show an example of \( \text{PropFrmn} \)-derivation under the hypothesis sequence \( A : \text{Prop}, B : \text{Prop} \):

\[
\frac{B \quad A}{A \land (B \supset A) : \text{Prop}} \quad \text{imp_frmn}
\]

\[
\frac{B \quad A}{A \land (B \supset A) : \text{Prop}} \quad \text{and_frmn}
\]

This is a \( \text{PropFrmn} \)-derivation of the following hypothetical judgment:

\[ A : \text{Prop}, B : \text{Prop} \vdash A \land (B \supset A) : \text{Prop} \]

For the derivation game \( \text{PropProof} \), a hypothesis can be either of the form \( A : \text{Prop} \) or of the form \( x : P \) where \( A \) and \( x \) are variables and \( P \) must be a proposition. Again, we avoid enumerating all the derivation rules, and only show a few cases of \( \text{PropProof} \)-derivations.

Let \( \Gamma \) be a \( \text{PropProof} \)-hypothesis-sequence, \( P, Q \) be propositions under \( \Gamma \) and \( x \) be a variable. Then, the following two rules are the introduction and elimination rules of the implication under \( \Gamma \).

\[
\frac{(x : P) \quad [Q]}{P \supset Q} \quad \text{imp_intro}
\]

\[
\frac{P \supset Q \quad P}{Q} \quad \text{imp_elim}
\]

The hypothesis \( x : P \) in the \( \text{imp_intro} \) rule is discharged as in the ordinary natural deduction system. A characteristic point in our formulation is that we may choose any variable for \( x \) while in the ordinary formulation, the label for the discharged assumption should be different from other labels, although this condition is often left implicit.

As an example, here is a \( \text{PropProof} \)-derivation of the hypothetical judgment \( A : \text{Prop}, B : \text{Prop} \vdash A \supset B \supset A \).

\[
\frac{(x : A) \quad [(y : B) [x]] \quad \text{imp_intro}}{B \supset A \quad \text{imp_intro}}
\]

\[
\frac{B \supset A \quad \text{imp_intro}}{A \supset B \supset A}
\]

\(6\) Besides the derivation game for abbreviations.
3.3 Simply Typed Lambda Calculus

It is now straightforward to define the simply typed lambda calculus by means of a derivation game. To pick up the corresponding rules to propositional logic, we have the following rules:

\[
\frac{(x : P)[M : Q]}{\lambda(x : P)[M] : P \supset Q} \quad \frac{M : P \supset Q \quad N : P}{M(N) : Q} \quad \text{lambdafrmn} \quad \text{applyfrmn}
\]

where \(\lambda(x : P)[M]\) is \(\lambda\)-abstraction and \(M(N)\) is functional application.

3.4 Computational Aspect of Logic

One of the main goals of our lecture course is to establish the Curry-Howard isomorphisms between certain logical systems and computational systems. In section 1, we listed 6 items we teach in the course. Item 3 establishes the Curry-Howard isomorphism between the intuitionistic propositional logic and the simply typed lambda calculus we teach in 1 and 2 respectively. In item 6, we extend this isomorphism between the Heyting arithmetic and the dependently typed lambda calculus we teach in 4 and 5 respectively. We show that the isomorphisms are not only bijective but also compatible with the reduction relations defined on the logical systems and computational systems. In this way, we can show students that proving and programming can be seen as essentially the same mental activities under a suitable setting.

More precisely, in item 3, we introduce two definition games \text{LambdaTermRed} and \text{PropProofRed}. The former defines the reduction relation on lambda terms given in 3.3 and the latter defines that of derivations of propositional logic given in 3.2. Here, we pick up the \(\beta\)-reduction rule as a typical example. In \text{LambdaTermRed}, let \(\Gamma\) be a hypothesis sequence, let \(P\) be a type and \(M\) be a simply typed lambda term under \(\Gamma\), let \(x\) be a variable and \(L\) be a simply typed lambda term under \(\Gamma, x : P\), and let \(N \equiv \text{subst}(M, x, L)\). Then we have the following rule schema:

\[
\frac{\lambda(x : P)[L](M) \text{ red } N}{\beta}
\]

In \text{PropProofRed}, let \(\Gamma\) be a hypothesis sequence, let \(P\) and \(R\) be propositions under \(\Gamma\), let \(D\) be a \text{PropProof}-derivation under \(\Gamma\) whose conclusion is \(P\), and let \(x\) be a variable and \(D_R\) be a \text{PropProof}-derivation under \(\Gamma, x : P\) whose conclusion is \(R\). Then letting \(E\) be the following derivation:

\[
\frac{(x : P)[D_R]}{P \supset R} \quad \frac{D}{R} \quad \text{imp_intro} \quad \text{imp_elim}
\]

we have the following rule schema:

\[
\frac{E \text{ red subst}(D, x, D_R)}{\text{imp}}
\]
These two rule schemata are identical (except the names of rule schemata) if we identify the construction of lambda terms and derivations of propositional logic. Although the Curry-Howard isomorphism is not easy to understand, students can gradually grasp the essence by repeatedly solving problems given in both games.

We note that the reduction rule schema `imp` uses the substitution operation defined for expressions. In contrast to this, in most logical systems, some special substitution operation for derivations is necessary to define the reduction of derivations.

## 3.5 Heyting Arithmetic

Heyting arithmetic is defined as the derivation game `ArithProof` in CAL. Among the rule schemata in `ArithProof`, a typical rule schema is `∃`-elimination:

\[
\exists(x)[P] \quad (k: \text{Nat}, X:\text{inst}(x)[P] \uparrow k, k) \quad [R \uparrow k]
\]

\[
\begin{array}{c}
\Rightarrow R \exists \text{elim}
\end{array}
\]

where `Γ` is a hypothesis sequence, `x, k, X` are variables, `P` is an arithmetical proposition under `Γ`, `x : \text{Nat}`, and `R` is an arithmetical proposition under `Γ`.

We note that, in the traditional presentation of first-order logic, a side condition (the so-called eigen variable condition) must be satisfied to apply the `∃`-elimination rule. Since the eigen variable condition is given as a side condition and it is difficult to check, one often writes a derivation in which the condition is violated. In contrast to this, our rule schema has no side conditions as is shown above. Instead of having meta-level side conditions, we internalized the hidden conditions using the push-operation.

## 4 CAL System

As we have explained briefly in the introduction, the CAL system is implemented entirely in Emacs Lisp and it consists of the three components: the user interface, the parser and the checker. In this section we describe each of these three components.

### 4.1 User Interface

The user interface provides an interactive command interpreter in a special buffer created by the CAL system. When a user starts up the CAL system, the user interface first evaluates the user’s record file. The record file contains useful values such as the list of problems the user solved in the previous CAL sessions, the list of theorems the user proved so far, or the list of news already read by the user. The interface then checks if there is any unread news, and if so issues the message saying that there are some unread news. After that, the interface prompts the user for a command by issuing the prompt `CAL<1>`.

The user can now type in a CAL command. Each CAL command is one of the following forms:
A command in the first form takes a single argument and a command in the second form takes two arguments. Currently about 30 CAL commands are available. Most of them are commands for checking the correctness of derivations, but there are several commands that help users such as help or news. The help command is a command that takes a keyword as its argument. For example, help[help] explains the help command itself, and help[key] explains the key bindings local to the CAL buffer. The news command is used to read news provided by us. By these commands, users can learn the usage of the CAL system.

For each game we have a command that checks whether a given derivation is a correct derivation in the given game. For example, PropProof[D] checks if D is a correct derivation in the derivation game PropProof. When inputting D, the user interface provides a number of supports. The local key bindings are set up so that special symbols: ⊢ (proves), ⊃ (implication), ∧ (conjunction), ∨ (disjunction), ¬ (negation), ∀ (all) and ∃ (exists) can be input by typing /p, /i, /c, /d, /n, /a and /e, respectively. Also, since a derivation has a tree-like shape, the interface automatically inserts appropriate number of blank characters so that the text will be indented properly. The interface then sends the derivation D to the parser. The parser parses the character string D and translates it into an expression, say, E. The parser will report a syntax error if D cannot be parsed successfully. If the parser returns an expression E, the user interface then sends E to the checker and asks to check if E is a correct derivation of the derivation game PropProof or not. The checker returns an error if E is not a correct derivation and just returns nil (a Lisp constant) if E is a correct derivation. The user interface prints out the result of the proof checking into the CAL buffer.

The most important pair of CAL commands is Problem and Answer. There are about 220 problems CAL provides and the students are expected to solve most of them. The usage of these command is best explained by listing part of a sample CAL session. In the following listing, underlined texts are inserted by the system and other texts are inserted by the user.

CAL <1> Problem[41]

Construct a derivation of the following hypothetical judgment in the derivation game PropProof:

A:Prop, B:Prop, x:¬A, y:¬B ⊢ ¬(A ∨ B)

CAL <2> Answer[41]

A:Prop, B:Prop, x:¬A, y:¬B ⊢ ¬(A ∨ B) in PropProof since

¬(A ∨ B) by imp_intro {
(z:A ∨ B) {
    ⊥ by or_elim {
        z;

7 English sentences in the example below are actually Japanese.
(zl:A)[⊥ by imp_elim {x; zl}];
(zr:B)[⊥ by imp_elim {y; zr}]
})

Correct!

To solve problems about Heyting arithmetic, the Definition command and the Theorem command are indispensable. For example:

CAL <3> Definition[LessThanOrEq][
x:Nat, y:Nat ⊢ ∃(z)[x + z = y]:Prop in ArithPropFrmn since
∃(z)[x + z = y]:Prop by exist_frmn {
  (z:Nat)[x + z = y]:Prop by eq_frmn {}
}]
LessThanOrEq defined as a 2-ary predicate.

defines LessThanOrEq as a binary predicate. Note that the user must supply a proof that ∃(z)[x + z = y] is indeed a proposition of Heyting arithmetic under the hypotheses x:Nat and y:Nat. The Theorem command can be used to prove a new theorem, give it a name, and use it later as shown below. Note also the usage of the defined predicate LessThanOrEq.

CAL <4> Theorem[LessThanOrEqIsReflexive][
x:Nat ⊢ LessThanOrEq(x, x) in ArithProof since
LessThanOrEq(x, x) by def_intro {
  ∃(z)[x + z = x] by exist_intro {
    x + 0 = x by plus_0 {}
  }
}]
You have just proved a new theorem: LessThanOrEqIsReflexive.

CAL <5> ArithProof[
⊢ LessThanOrEq(0, 0) in ArithProof since
LessThanOrEq(0, 0) by LessThanOrEqIsReflexive {}
]
Yes, that is certainly a correct derivation in ArithProof!

Definitions made so far and theorems proved so far can be listed by the commands list-definition and list-theorem.

The user can terminate the CAL session by typing in the command bye[].
The user interface, then updates the record file and also writes out the session log in a file. The log file provides us with useful information as to the points where typical users make mistakes.
4.2 Parser

When the CAL parser receives a character string $S$ from the user interface, it first decomposes it into a list of tokens $W$, and then it tries to construct a CAL expression $E$ such that the judgment $W$ denotes $E$ is derivable in the definition game $\text{Abbrev}$. Thus the CAL parser may be regarded as an automatic theorem prover for $\text{Abbrev}$-judgments. The definition game $\text{Abbrev}$ is a union of about 20 sub-definition games and each sub-definition game defines a syntactic class. (Each sub-definition game usually consists of several sub-sub-definition games, and they also define corresponding syntactic classes.) Therefore, whenever the user interface calls the parser, the UI sends not only $S$ but also the name of the syntactic class to which the result of parsing $S$ is expected to belong. For example,

\begin{verbatim}
(ParseString "s(s(0))" 'NatNum-exp)
\end{verbatim}

asks the parser to parse the string $s(s(0))$ into an expression in the syntactic class $\text{NatNum-exp}$, where the class is defined by the context free grammar:

\[
\text{NatNum-exp} ::= "0" \mid "s" "(" \text{NatNum-exp} ")"\]

This grammar is represented in a slightly modified form in CAL as the following macro definition.

\begin{verbatim}
(defclass NatNum-exp
  (or
   "0"
   (prefix "s" (paren "(" NatNum-exp))
   (throw "Not a valid NatNum-exp!"))
)
\end{verbatim}

The parser tries to match the list of tokens $W$ against the pattern which is the body of the definition. A pattern of the form $(\text{prefix } "s" \ P)$ matches $W$ if the first element of $W$ is $s$ and the tail (\text{cdr} in Lisp) of $W$ matches the pattern $P$. The pattern $(\text{paren } "(" \text{NatNum-exp})$ matches $W$ if the first and the last elements of $W$ is a pair of parentheses ‘‘ and ‘’ and the list of the remaining tokens matches $\text{NatNum-exp}$. There is a special file that defines all the CAL syntactic classes, and this file is loaded by the UI when the user starts up the CAL system. So the parser can get the necessary syntactic definitions from the Lisp memory. In the present case, the parser returns the CAL expression $<s, <s, 0>>$ as the result of parsing $W$ into an expression in $\text{NatNum-exp}$.

4.3 Checker

The CAL checker is a simple proof-checker; it receives a derivation and checks if it is a correct derivation in the specified game. If it is correct, it just returns a non-significant value. Otherwise, the checker analyzes the sources of incorrectness and returns an error message indicating a source of the incorrectness. Since the
CAL system is used by students, it is important to give helpful error messages, and we have tried to give effective suggestions to students. Yet, much more are still left to be done. In the current implementation, the checker traverses each derivation tree in the bottom-up (from leaves to the root), left-to-right direction.

5 Conclusion

We started our lecture course, Logic and Computation, in 1998 and we have been using the CAL system since then. Students attended the course liked the CAL system very much and used CAL intensively. They also gave us useful suggestions which greatly contributed to improve the system.\footnote{We would like to acknowledge them sincerely.} By utilizing such feedbacks from students we could gradually polish our system into the current form. Although the system is not perfect, our system has proven to be a good system to complement our lecture course. Through the teaching and implementing experiences we now strongly feel that education of formal logics should always be supported by computer environments like the CAL system.

There are a number of proposals which aim to give frameworks for presenting logical systems, and they are called logical frameworks. CAL is one such framework, and we compare below our system with other frameworks.

Among such frameworks, Martin-Löf’s theory of expressions is based on the concept of arity and our system is strongly influenced by his theory. The theory is used as an underlying system, and an object logic (Intuitionistic Type Theory in this case) is presented on top of it, but it is general enough to be able to implement other logical systems on top of it. The theory has a built-in abstraction mechanism and instantiation mechanism, and the definitional equality relation on the expressions is defined by a set of reduction rules on expressions. The reduction relation is Church-Rosser and strongly normalizing and hence the equality is decidable. In Martin-Löf’s theory of expressions, $\alpha$-equivalent expressions are definitionally equal.

Our theory of expressions has stronger notion of definitional equality in the sense that we can distinguish two $\alpha$-equivalent and indistinguishable expressions in Martin-Löf’s theory. However, it is easy to add a weaker notion of equality on our expressions by identifying $\alpha$-equivalent expressions, and under this weak equality, our theory and Martin-Löf’s theory become isomorphic.

The reason why our theory is much simpler than Martin-Löf’s and yet more expressive than his theory is that we defined our expressions only by using constructors (introduction rules) while Martin-Löf’s theory has both constructors and destructors for abstracts and lists. In case of abstracts, Martin-Löf’s theory has both an abstraction rule which constructs (introduces) abstracts and an application rule which destructs (eliminates) abstracts. In our theory, we have only an introduction rule for abstracts and elimination of abstracts is defined not within the theory but as a meta-level operation of instantiation. Our philosophy is that expressions must be designed as a data structure and we showed
that it is possible to do so naturally by using constructors only as, for example, Lisp symbolic expressions are constructed from atoms only by means of the cons operation.

The reason why we could develop our theory as a type-free theory is again that our theory has only introduction rules. Martin-Löf's theory uses the notion of arity to avoid self application which might make the theory not strongly normalizing. Our theory is strongly normalizing and Church-Rosser in the very trivial sense that we have no notion of reduction in our theory. Martin-Löf's theory requires a reduction relation to define the notion of definitional equality, but we can define definitional equality in our theory without the notion of reduction as we saw in section 2 and this was possible since our theory has only introduction rules. The notion of reduction, that is instantiation, is introduced as a meta-level concept only after we defined the theory.

Edinburgh Logical Framework [4] is a powerful framework for representing various logics in the type-theoretic setting, and is considered as a generalization of Martin-Löf's theory. But precisely because it is a generalization based on type theory, it cannot serve as a foundational framework for building logical theories on top of it. A foundational framework on top of which logical systems and type theories can be built must itself be built without using type theories. Our theory of expressions is such a foundational framework.

References

A Set Theory Prover
Within Theorema

Wolfgang.Windsteiger@RISC.Uni-Linz.ac.at
RISC Institute
Hagenberg, Austria.

Abstract. In this paper, we present the Theorema Set Theory Prover. This prover is designed for proving statements involving notions from set theory using natural deduction inference rules for set theory. Moreover, it applies the PCS paradigm (Proving-Computing-Solving) for generating natural proofs that has already been used in other provers in the Theorema system, notably the prover for automated proofs in elementary analysis. We show some applications of this prover in a case study on equivalence relations and partitions, which also nicely shows the interplay between proving, computing, and solving during an exploration of some mathematical theory.

1 Introduction

In this paper we report on the implementation of a Set Theory prover within the Theorema system. Particular emphasis has been put on the integration of set theory proving into the PCS heuristic for proving in predicate logic that has been invented by B. Buchberger, see [1], [2] or [3], and that has already been successfully applied to proofs in the area of elementary calculus using Theorema in [4]. It is shown how prove ("P"), compute ("C"), and solve ("S") phases contribute to natural style proofs of theorems involving notions of set theory. Opposed to an immediate approach of reducing set constructions to predicate logic "by definition" we provide special inference rules for all kinds of formulae involving sets in order to produce "elegant proofs". For instance, the proof problem of showing \( A \subseteq B \) reduces by definition of "\( \subseteq \)" to showing \( \forall x \in A \Rightarrow x \in B \), which we could then pass to the predicate logic prover. Instead, a human mathematician would probably just assume \( x_0 \in A \) and show \( x_0 \in B \) for some arbitrary \( x_0 \). This is the type of reasoning employed in the "P" phase by the Theorema Set Theory PCS Prover.

This work has been supported by the "Spezialforschungsbereich for Numerical and Symbolic Scientific Computing" (SFB F013) at the University of Linz, the "PROVE Project" supported by the regional government of Upper Austria, and the european union "CALCULEMUS Project" (HPRN-CT-2000-00102).
In addition, we try to incorporate the capabilities of the semantics of the *Theorema* language in order to deal with finitary set constructions during the compute (simplify) phase. The semantics of set operations available in *Theorema* computations not only allows computations with finite sets (such as \{1, 2, 3\}) but also with finite versions of the set quantifier (such as \{x \in A \mid P_x\} where A is a finite set and P_x is some formula depending on x). We present an approach to combining proving and computing by doing semantic based computations during the "C" phase of proving in order to simplify proof situations.

Other constructions in set theory, however, directly lead to "solve problems". For example, proving \(8 \in \{n^2 - 1 \mid n \in \mathbb{N}\}\) amounts to solving the equation \(8 = n^2 - 1\) (over the integers). We present an approach to combining proving and solving by reducing typical "solve patterns" to "easier prove problems" using known solving techniques available in Mathematica.

It will be demonstrated in numerous examples, that the approach of viewing each proof as a sequence of *alternating proof phases* ("PCS") gives quite promising results in many applications. In this version of the paper, we give some examples of fully automatically generated proofs by *Theorema*, for more examples and a more detailed description of the prover, see [5].

### 2 The P-C-S Paradigm for *Theorema* Provers

Proving is seen (mainly) as a sequence of alternating phases of

- Proving
- Computing
- Solving

It is a *heuristic method* that aims on *automated proofs in natural style* trying to mimick what is actually done in proofs by humans. In the following sections, we will try to give some more details on the notions "Proving", "Computing", and "Solving", in particular, their interaction in the development of a generally applicable prove strategy.
2.1 Proving

**Problem Specification (Proving):**

Given: a collection of formulae $A$ (the assumptions),

a formula $G$ (the goal, the conclusion).

Find: $P$, such that:

if $G$ is a consequence of $A$ then $P$ should contain a
convincing argument (logically correct, sound, understandable),

*why* $G$ necessarily follows from $A$.

We call the $P$ to be found in a proving problem a *proof of* $G$ (*from* $A$).

**Example**

*Goal:* $\{1\} \subseteq \mathbb{N}$. *Assumptions:* Definition of $\subseteq$, $\mathbb{N}$, $\{\ldots\}$, and $1$.

*Proof:* Take $a \in \{1\}$, which forces $a = 1$, hence $a \in \mathbb{N}$.

**The Role of Proving Within PCS**

The proving phase in a PCS-cycle is the phase where standard inference techniques from logic are applied in order to reduce the goal or infer new knowledge from known assumptions. Typically, these rules eliminate quantifiers, split up formulae, or even split up an entire proof into several branches.

2.2 Computing

**Problem Specification (Computing, Simplifying):**

Given: a collection of formulae $A$ (the assumptions),

a term $T$ or a formula $F$.

Find: a term $\tilde{T}$ or a formula $\tilde{F}$, such that:

(a) $T = \tilde{T}$ is a consequence of $A$ and

$\tilde{T}$ is (in some sense) simpler than $T$.

or

(b) $F \iff \tilde{F}$ is a consequence of $A$ and

$\tilde{F}$ is (in some sense) simpler than $F$.

We name the *process* of finding $\tilde{T}$ or $\tilde{F}$ *computation (simplification) w.r.t.* $A$, and we call $\tilde{T}$ and $\tilde{F}$ respectively the *result of the computation w.r.t.* $A$.
Example

Compute: \( \{ x \mid \text{is-prime}[x] \} \cap \{ x \mid \text{Mod}[x, 5] \leq 3 \} \).
Result: \{2, 3, 7\} or ??

Compute: \( 3 \in \{ x \mid \text{is-prime}[x] \} \cap \{ x \mid \text{Mod}[x, 5] \leq 3 \} \).
Result: \( 3 \in \{2, 3, 7\} \) or True or ?? (depending on the available knowledge \( A \)).

The Role of Computing Within PCS

The computing phase in a PCS-cycle is typically the phase where terms or formulae, either in the goal or in the knowledge base, are simplified using knowledge from the semantics of the Theorema language. The Theorema language semantics covers logical connectives and the algorithmic language constructs for quantifiers, sets, and tuples. For details on the Theorema language and its semantics, we refer to [6] and [7].

2.3 Solving

Problem Specification (Solving):

Given: a formula \( G \) of the form \( \exists_{\vec{x}} P_{\vec{x}} \),
a collection of formulae \( A \) (the assumptions),
such that \( G \) is a consequence of \( A \).
Find: a proof of \( G \) by giving \( S \), such that
\( S \subseteq \{ s \mid P_s \text{ can be derived from } A \text{ and } P_s \text{ is free of } \vec{x} \} \) and \( |S| \geq 1 \).

We call the process of finding \( S \) solving \( G \) w.r.t. \( A \), each element of \( S \) is called a solution for \( x \) of \( G \) w.r.t. \( A \), and we call \( S \) a set of solutions for \( x \) of \( G \) w.r.t. \( A \).

(In above problem specification, \( \vec{x} \) stands for a sequence of one or more variables, \( P_s \) denotes the result of applying the substitution \( s \) to the formula \( P \), and a term or formula is called free of \( \vec{x} \) if it contains none of the variables in \( \vec{x} \). We assume, by the way, that the reader is familiar with the notion of substitution, we will use it in a standard way.)
Example

\[ \exists_{x,y \in \mathbb{R}} \frac{x^2 + y^2}{P_{x,y}} = 1, \]

and the assumptions are "the basic properties of the domain of real numbers" and some knowledge about trigonometric functions "sin" and "cos". One solution for \( x \) and \( y \) would be

\[ S_1 = \{ x \to 0, y \to 1 \} \]

since \( P_{(x \to 0, y \to 1)} \) gives \( 1 = 1 \), which is of course true. Of course, also

\[ S_2 = \left\{ \{ x \to 0, y \to 1 \}, \{ x \to 1, x \to 0 \}, \left\{ x \to \frac{\sqrt{2}}{2}, y \to \frac{\sqrt{2}}{2} \right\}, \left\{ x \to \frac{\sqrt{2}}{2}, y \to -\frac{\sqrt{2}}{2} \right\} \right\} \]

is a set of solutions for \( x \) and \( y \) of \( G \) according to our specification, as can be easily checked by applying the four substitutions to \( P_{x,y} \) and performing basic arithmetic calculations. Another type of solution, which also meets the specification, is

\[ S_3 = \{ \{ x \to \cos(t), y \to \sin(t) \} | 0 \leq t < 2 \pi \} \]

since \( P_{(x \to \cos(t), y \to \sin(t))} \) gives \( \cos(t)^2 + \sin(t)^2 = 1 \), which holds for every \( 0 \leq t < 2 \pi \). This type of solution set is typically called a parametrized solution.

The Role of Solving Within PCS

Solving comes into play during a PCS-cycle as soon as we have an existential goal to be proven. A well-known technique for handling existential quantifiers in the proof goal is the introduction of solve constants, sometimes also called meta-variables, see [8], used also in M. Beeson’s system MathXpert, see [9], and in the Theorema PCS prover for elementary analysis, see [4]. Finding an appropriate value for a solve constant usually amounts to solve a system of equations or inequalities, where it is, of course, problem-dependent what type of solving technique becomes necessary.

3 Provers in Theorema

In general, a Theorema Prover is structured as shown in Figure 1:
4 The Set Theory PCS Prover

User Prover: The SetTheoryPCSProver combines:

Set theory specific special provers:

- **STP**: The Set Theory Prove module handles set theoretic expressions in the goal.
- **STKBR**: The Set Theory KnowledgeBase Rewriting module infers new knowledge from set theoretic expressions in the assumptions.
- **STC**: The Set Theory Compute module performs (set theoretic) simplifications in the goal and in the assumptions.
- **STS**: The Set Theory Solve module tries to solve existential goals. In the context of set theory, such goals appear naturally e.g. in goals of the form \( y \in \{ T_x \mid P_x \} \), which immediately lead to \( \exists x \ y = T_x \land P_x \).

Other special provers involved from general PCS proving:

- Basic natural deduction.
- Recognition of terminal proof cases, see [5].
- Rewriting with respect to formulae in the knowledge base.
5 System Demonstration

5.1 General Outline of the Case Study

We investigate interactions of the following notions:

- Relations as sets of pairs.
- Equivalence relations and equivalence classes.
- The set of all equivalence classes in a set.
- Partitions of a set.
- The relation induced by a partition of a set.

5.2 Computing

The semantics of the Theorema language allows computations on certain expressions.

\[
\text{Definition} \quad \text{"reflexivity", any}[\text{Rel}, \text{S}], \\
\text{reflexives}_{\text{S}}[\text{Rel}] : \iff \forall_{x \in \text{S}} \langle x, x \rangle \in \text{Rel}
\]

\[
\text{Definition} \quad \text{"symmetry", any}[\text{Rel}, \text{S}], \\
\text{symmetric}_{\text{S}}[\text{Rel}] : \iff \forall_{x,y \in \text{S}} (\langle x, y \rangle \in \text{Rel} \Rightarrow \langle y, x \rangle \in \text{Rel})
\]

\[
\text{Definition} \quad \text{"Equivalence class", any}[x, A, R], \\
\text{class}_{A,R}[x] := \{a \in A \mid \langle a, x \rangle \in R\}
\]

\[
\text{Compute} \quad \text{reflexive}_{\text{\{0,10\}}} \{\langle i, j \rangle \mid (\text{Mod}[i, 5] = \text{Mod}[j, 5])\}, \\
\text{using } \rightarrow (\text{Definition["reflexivity"]}), \text{ built-in } \rightarrow \\
(\text{Built-in["Sets"]}, \text{Built-in["Quantifiers"]}, \text{Built-in["Connectives"]})
\]

\[
\langle 0, 0 \rangle \in \{\langle 0, 0 \rangle, \langle 0, 5 \rangle, \langle 0, 10 \rangle, \langle 1, 1 \rangle, \langle 1, 6 \rangle, \langle 2, 2 \rangle, \langle 2, 7 \rangle, \\
\langle 3, 3 \rangle, \langle 3, 8 \rangle, \langle 4, 4 \rangle, \langle 4, 9 \rangle, \langle 5, 0 \rangle, \langle 5, 5 \rangle, \langle 5, 10 \rangle, \langle 6, 1 \rangle, \langle 6, 6 \rangle, \\
\langle 7, 2 \rangle, \langle 7, 7 \rangle, \langle 8, 3 \rangle, \langle 8, 8 \rangle, \langle 9, 4 \rangle, \langle 9, 9 \rangle, \langle 10, 0 \rangle, \langle 10, 5 \rangle, \langle 10, 10 \rangle\}
\]

\[
\wedge \iff 9 \wedge \quad \langle 10, 10 \rangle \in \{\langle 0, 0 \rangle, \langle 0, 5 \rangle, \langle 0, 10 \rangle, \langle 1, 1 \rangle, \langle 1, 6 \rangle, \langle 2, 2 \rangle, \langle 2, 7 \rangle, \\
\langle 3, 3 \rangle, \langle 3, 8 \rangle, \langle 4, 4 \rangle, \langle 4, 9 \rangle, \langle 5, 0 \rangle, \langle 5, 5 \rangle, \langle 5, 10 \rangle, \langle 6, 1 \rangle, \langle 6, 6 \rangle, \\
\langle 7, 2 \rangle, \langle 7, 7 \rangle, \langle 8, 3 \rangle, \langle 8, 8 \rangle, \langle 9, 4 \rangle, \langle 9, 9 \rangle, \langle 10, 0 \rangle, \langle 10, 5 \rangle, \langle 10, 10 \rangle\}
\]
5.3 Computing ⊆ Proving

Goal: Make use of knowledge provided by the Theorema language semantics during proving!

**Proposition** "R reflexive on finite set",

reflexive_{i=0..10} \{(i, j) \mid (\text{Mod}[i, 5] = \text{Mod}[j, 5])\}

Prove:\[\text{Proposition}["R reflexive on finite set"],\]

using → Definition["reflexivity"]

Prove:

(Proposition (R refl)) \reflexive_{i=0..10} \{(i, j) \mid (\text{Mod}[i, 5] = \text{Mod}[j, 5])\},

under the assumption:

(Definition (reflexivity)) \forall_{\text{Rel}, S} (\reflexive_S[\text{Rel}] :\iff \forall_{x \in S} (\langle x, x \rangle \in \text{Rel})).

Using available computation rules we simplify (Proposition (R refl)):

1. \reflexive_{i=0,1,2,3,4,5,6,7,8,9,10} \{(0, 0), (0, 5), (0, 10), \text{\langle 19\rangle}, (10, 0), (10, 5), (10, 10)\}.

Formula (1), using (Definition (reflexivity)), is implied by:

(2) \forall_{x \in \{0,1,2,3,4,5,6,7,8,9,10\}} (\langle x, x \rangle \in \{0, 0, 5, 10, \langle 19\rangle, 10, 0, 10, 5, 10, 10\})

Using available computation rules we simplify (2):

(3) True.

Formula (3) is true because it is the constant True.
**Proposition** "R symmetric on finite set",

\[
\text{symmetric}_{\{i \mid i = 0, \cdots, 10\}} \left[ \{i, j \mid (\text{Mod}[i, 5] = \text{Mod}[j, 5]) \} \right]
\]

Prove [Proposition] "R symmetric on finite set",

using → Definition ["symmetry"],

built-in → \{\text{Built-in["Quantifiers"], Built-in["Tuples"]}\}

Prove:

(Proposition (R symm)) \text{symmetric}_{\{i \mid i = 0, \cdots, 10\}} \left[ \{i, j \mid \text{Mod}[i, 5] = \text{Mod}[j, 5]\} \right],

under the assumption:

(Definition (symmetry)) \forall_{\text{Rel}, S} \left( \text{symmetric}_S[\text{Rel}] :\Leftrightarrow \forall_{x \in S, y \in S} ((x, y) \in \text{Rel} \Rightarrow (y, x) \in \text{Rel}) \right).

Using available computation rules we simplify (Proposition (R symm)):

(1) \text{symmetric}_{\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}} \left[ \{0, 0\}, \{0, 5\}, \{0, 10\}, \langle 19 \rangle, \{10, 0\}, \{10, 5\}, \{10, 10\}\right].

Formula (1), using (Definition (symmetry)), is implied by:

(2) \forall_{x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}} ((x, y) \in \{0, 0\}, \{0, 5\}, \{0, 10\}, \langle 19 \rangle, \{10, 0\}, \{10, 5\}, \{10, 10\}) \Rightarrow.

Using available computation rules we simplify (2):

(3) True.

Formula (3) is true because it is the constant True.

### 5.4 Computing ⊂ Proving

The *Theorema* language does not provide semantics for infinite ranges in quantifiers, thus, when switching to expressions of that form we cannot anymore reduce proving to simple (finite) computations.

Compute[reflexive_\mathbb{N} \left[ \{i, j \mid (\text{Mod}[i, 5] = \text{Mod}[j, 5])\} \right],

using → \langle\text{Definition["reflexivity"]}\rangle]

\forall_{x \in \mathbb{N}} \left( (x, x) \in \{i, j \mid (i, j) \in \mathbb{N}, \text{Mod}[i, 5] = \text{Mod}[j, 5]\} \right)
5.5 Solving \( \subseteq \) Proving

Goal: Make use of available algebraic solution techniques during proving!

**Proposition** ["R reflexive on \( \mathbb{N} \),

reflexive\( \mathbb{N} \)\( \{(i, j) \mid (i, j) \in \mathbb{N} \land (\text{Mod}[i, 5] = \text{Mod}[j, 5]) \land j \in \mathbb{N}\}\)]]

Prove[Proposition["R reflexive on \( \mathbb{N} \)"],

using \( \rightarrow \) Definition["reflexivity"], built-in \( \rightarrow \) Built-in["Tuples"]]

Prove:

\[(\text{Proposition (R refl)}) \quad \text{reflexive}\( \mathbb{N} \)\( \{(i, j) \mid (i, j) \in \mathbb{N} \land (\text{Mod}[i, 5] = \text{Mod}[j, 5]) \land j \in \mathbb{N}\}\),

under the assumption:

\[(\text{Definition (reflexivity)}) \quad \forall_{\text{Rel}, S} \left( \text{reflexive}\_S[\text{Rel}] : \iff \forall_x (x \in S \Rightarrow \langle x, x \rangle \in \text{Rel}) \right)\].

Formula (Proposition (R refl)), using (Definition (reflexivity)), is implied by:

\[(1) \quad \forall_x \left( x \in \mathbb{N} \Rightarrow \langle x, x \rangle \in \left\{ \langle i, j \rangle \mid i \in \mathbb{N} \land j \in \mathbb{N} \land (\text{Mod}[i, 5] = \text{Mod}[j, 5]) \right\} \right) ;

We assume

\[(2) \quad x_0 \in \mathbb{N}\],

and show

\[(3) \quad \langle x_0, x_0 \rangle \in \left\{ \langle i, j \rangle \mid i \in \mathbb{N} \land j \in \mathbb{N} \land (\text{Mod}[i, 5] = \text{Mod}[j, 5]) \right\} \].

In order to prove (3) we have to show

\[(4) \quad \exists_{i, j} ((i, j) \in \mathbb{N} \land j \in \mathbb{N} \land (\text{Mod}[i, 5] = \text{Mod}[j, 5])) \land (\langle x_0, x_0 \rangle = \langle i, j \rangle).\]

Since \( i := x_0 \) and \( j := x_0 \) solves the equational part of (4) it suffices to show

\[(5) \quad x_0 \in \mathbb{N} \land x_0 \in \mathbb{N} \land (\text{Mod}[x_0, 5] = \text{Mod}[x_0, 5]).\]

We prove the individual conjunctive parts of (5):

Proof of (5.1) \( x_0 \in \mathbb{N} \):

Formula (5.1) is true because it is identical to (2).

Proof of (5.2) \( x_0 \in \mathbb{N} \):

Formula (5.2) is true because it is identical to (2).

Proof of (5.3) \( \text{Mod}[x_0, 5] = \text{Mod}[x_0, 5] \):

It can be easily verified that formula (5.3) always holds.
5.6 Computing \( \bigcup \) Solving \( \subseteq \) Proving

Due to space limitations, we only give one example of the interplay between proving, computing, and solving. For more examples we refer to the technical report [10], for a full exploration of equivalence relations, partitions, etc. with a proof also for the auxiliary proposition, see [5].

**Proposition**["induced by partition is transitive", any\(\{P\}\), with\([\text{is-partition}\{P\}]\), is-transitive\([\text{induced-relation}[\{P\}]\)]

One auxiliary property is needed in the proof:

**Proposition**["\(\cap =\)\", any\(\{P\}\), with\([\text{is-partition}\{P\}]\),
\(\forall_{X,Y\in P} (X \cap Y \neq \emptyset \Rightarrow (X = Y))\]  

Prove\([\text{Proposition}["induced by partition is transitive"],
using \(\rightarrow\)
(Definition["transitivity"], Definition["induced relation"], Proposition["\(\cap =\)"])]

Prove:

\((\text{Proposition (ind is trans)}) \quad \forall_P (\text{is-partition}[P] \Rightarrow \text{is-transitive}[\text{induced-relation}[P]])\),

under the assumptions:

\((\text{Definition (trans)}) \quad \forall_R \left(\text{is-transitive}[R] : \iff \forall_{x,y,z} ((x, y) \in R \land (y, z) \in R \Rightarrow (x, z) \in R)\right),\)

\((\text{Definition (ind relation)}) \quad \forall_S \left(\text{induced-relation}[S] : \iff \left\{ (x, y) \mid \exists_{x,y} (x \in M \land y \in M) \right\}\right),\)

\((\text{Proposition (\(\cap =\) )}) \quad \forall_P (\text{is-partition}[P] \Rightarrow \forall_{X,Y} (X \in P \land Y \in P \Rightarrow ((X \cap Y \neq \emptyset) \Rightarrow (X = Y)))\).

We assume

\((1) \quad \text{is-partition}[P_0],\)

and show

\((2) \quad \text{is-transitive}[\text{induced-relation}[P_0]].\)

Formula (2), using (Definition (trans)), is implied by:

\((3) \quad \forall_{x,y,z} ((x, y) \in \text{induced-relation}[P_0] \land (y, z) \in \text{induced-relation}[P_0] \Rightarrow (x, z) \in \text{induced-relation}[P_0])\)

We assume

\((4) \quad (x_0, y_0) \in \text{induced-relation}[P_0] \land (y_0, z_0) \in \text{induced-relation}[P_0],\)

and show

\((5) \quad (x_0, z_0) \in \text{induced-relation}[P_0].\)

From what we already know follows:

From (4.1) we can infer

\((6) \quad \text{induced-relation}[P_0] \neq \emptyset.\)

Formula (5), using (Definition (ind relation)), is implied by:
(8) \( \langle x_0, z_0 \rangle \in \{ \langle x, y \rangle \mid \exists_{M \in P_0} (x \in M \land y \in M) \} \).

In order to prove (8) we have to show

\[ \exists_{x,y,M} (\exists_M (M \in P_0 \land (x \in M \land y \in M)) \land (\langle x_0, z_0 \rangle = \langle x, y \rangle)) \]  

Since \( x := x_0 \) and \( y := z_0 \) solves the equational part of (9) it suffices to show

\[ \exists_{M} (M \in P_0 \land (x_0 \in M \land z_0 \in M)). \]

Formula (6), by (Definition (ind relation)), implies:

\[ \exists_{x,y,M} (x \in M \land y \in M) \neq \{\}. \]

We did not find any ground formula to match a part of (10).

Formula (4.1), by (Definition (ind relation)), implies:

\[ \langle x_0, y_0 \rangle \in \{ \langle x, y \rangle \mid \exists_{M \in P_0} (x \in M \land y \in M) \} \]

From what we already know follows:

From (12) we know by definition of \( \{ T_x \mid P \} \) that we can choose an appropriate value such that

\[ \exists_{M} (M \in P_0 \land (xI_0 \in M \land x2_0 \in M)) \]

\[ \langle x_0, y_0 \rangle = \langle xI_0, x2_0 \rangle. \]

Using available computation rules we can simplify the knowledge base:

Formula (13) simplifies to

\[ \exists_{M} (M \in P_0 \land xI_0 \in M \land x2_0 \in M) \]

Formula (14) simplifies to

\[ (x_0 = xI_0) \land (y_0 = x2_0), \]

By (15) we can take appropriate values such that:

\[ M_0 \in P_0 \land xI_0 \in M_0 \land x2_0 \in M_0. \]

From what we already know follows:

From (17.1) we can infer

\[ P_0 \neq \{\}. \]

From (17.2) we can infer

\[ M_0 \neq \{\}. \]

Because parts of the knowledge base match a part of (10), we try to find an instance of (10).

Now, let \( M := M_0 \). Thus, for proving (10) it is sufficient to prove:

\[ M_0 \in P_0 \land (x_0 \in M_0 \land z_0 \in M_0). \]

We prove the individual conjunctive parts of (21):

Proof of (21.1) \( M_0 \in P_0 \):

Formula (21.1) is true because it is identical to (17.1).

Proof of (21.2) \( x_0 \in M_0 \land z_0 \in M_0 \):

We prove the individual conjunctive parts of (21.2):

Proof of (21.2.1) \( x_0 \in M_0 \):

Formula (21.2.1), using (16.1), is implied by:

\[ xI_0 \in M_0. \]

Formula (22) is true because it is identical to (17.2).
Proof of (21.2.2) $z_0 \in M_0$:  
Formula (4.2), by (16.2), implies:
\[ \langle x_2, z_0 \rangle \in \text{induced-relation}[P_0], \]
which, by (Definition (ind relation)), implies:
\[ (23) \quad \langle x_2, z_0 \rangle \in \{ \langle x, y \rangle \mid \exists_{M \ni P_0} (x \in M \land y \in M) \}. \]
From what we already know follows:
From (23) we know by definition of $\{ T_x \mid P \}$ that we can choose an appropriate value such that
\[ (24) \quad \exists_{M} (M \in P_0 \land (x_3 \in M \land x_4 \in M)), \]
\[ (25) \quad \langle x_2, z_0 \rangle = \langle x_3, x_4 \rangle. \]
Using available computation rules we can simplify the knowledge base:
Formula (24) simplifies to
\[ (26) \quad \exists_{M} (M \in P_0 \land x_3 \in M \land x_4 \in M), \]
Formula (25) simplifies to
\[ (27) \quad (x_2 = x_3) \land (z_0 = x_4) \land (x_2 = x_3). \]
By (26) we can take appropriate values such that:
\[ (28) \quad M_1 \in P_0 \land x_3 \in M_1 \land x_4 \in M_1. \]
From what we already know follows:
From (28) we can infer
\[ (30) \quad M_1 \neq \emptyset. \]
Formula (21.2.2), using (27.2), is implied by:
\[ (32) \quad \exists_{x} x_4 \in M_0. \]
Formula (17.3), by (27.1), implies:
\[ (33) \quad x_3 \in M_0. \]
From what we already know follows:
From (28) together with (33) we know
\[ (35) \quad x_3 \in M_1 \cap M_0. \]
From what we already know follows:
From (35) we can infer
\[ (36) \quad M_1 \cap M_0 \neq \emptyset. \]
Formula (36), by (Proposition ($\cap = $)), implies:
\[ (37) \quad \forall_{P} (\text{is-partition}[P] \land M_0 \in P \land M_1 \in P \Rightarrow (M_1 = M_0)). \]
Formula (17.1), by (37), implies:
\[ (71) \quad \text{is-partition}[P_0] \Rightarrow (M_1 \in P_0 \Rightarrow (M_1 = M_0)). \]
From (1) and (71) we obtain by modus ponens
\[ (72) \quad M_1 \in P_0 \Rightarrow (M_1 = M_0). \]
From (28.1) and (72) we obtain by modus ponens
\[ (73) \quad M_1 = M_0. \]
Formula (32) is true because of (28.3) and (73).
Comments on the Method

The proof starts with a general "P" (prove) phase, which structures the proof according to the structure of the goal and the knowledge base. Natural deduction inference rules are used to eliminate universal quantifier(s) and to decompose compound formulae in the goal and in the knowledge base. As soon as one is left with atomic formulae (e.g. formula (2) and (5)) the "P" phase is terminated and a "C" (compute, simplify) phase is initiated, which uses particular formulae in the knowledge base in order to rewrite the goal or other formulae in the knowledge base. In this phase, definitions for function and predicate symbols are applied to goal and knowledge. After some cycles of "P" and "C" phases we arrive at formula (8), which then initiates a special "P" phase for handling formula whose outermost symbol is a set operator. In this example, the goal is of the form \( t \in \{ T_x \mid c \} \), which, "by definition", introduces an existential quantifier. Note however, that the definition of this variant of the set constructor is applied as an inference rule in the special set theory "P" phase, and it need not be given as a definition in the knowledge base.

Formula (9) is an existential goal, which in many situations describes a proof problem that can be handled by solving the quantified formula (or parts of it) with respect to the quantified variables. This is done now in an "S" (solve) phase. It of course depends on the structure of the formula, which solving technique is appropriate, but all available powerful black-box solving techniques (Gauss method, Gröbner bases method, Collins’ algorithm) are natural candidates to be applied in this step. In the example, one part of the formula is a tuple equation, which, after transforming it into a system of individual equations, can be solved by the standard Mathematica "Solve" command. We then continue to proof the remaining part of formula (9). Formula (10) is still an existential goal, but this time, due to the structure of the quantified formula, no solve technique is available, and we therefore suspend the "S" phase.

The steps from formula (12) until (17) are again cycles of "P" and "C", where (13) and (14) are derived by using definitions built into the prover, and, by application of Theorema language semantics of logical connectives and tuples, simplified to (15) and (16), respectively. Having new knowledge in the knowledge base, we now re-enter the "S" phase and we solve the existential goal (10) by basic predicate logic solving, i.e. we try to instanciate (10) guided by available formulae in the knowledge base. We now leave the "S" phase and continue in a "P"-"C" cycles. By a special inference technique for set theory, we infer formula (35): if we know \( x \in A \) and also \( x \in B \) then we can infer \( x \in A \cap B \). Formula (35) is now the key to make use of the auxiliary property about partitions given in the knowledge base and the proof quickly succeeds.

6 Conclusion

Two main theorems in the area of equivalence relations, induced relations, and partitions
together with all auxiliary propositions and lemmata starting from the definitions have been proved completely automatized using the Set Theory PCS Prover. After this case study we think that structuring proving into alternating cycles of P-C-S

- is a feasible approach for structuring automated theorem provers,
- gives naturally structured proofs, AND MOST IMPORTANTLY
- it opens the door for incorporating powerful known solving and computing techniques from the field of Computer-Algebra (Gröbner bases, Collins’ algorithm, Gauss method, Lazy Narrowing, etc.) into automated theorem provers.

References

The Tertiary Level in a Functional Cluster-Based Hierarchical VoD Server

Miguel Barreiro, Víctor M. Gulías, Juan J. Sánchez, and Santiago Jorge

LFCIA Lab, Department of Computer Science, Universidade da Coruña, Campus de Elviña, S/N, 15071, A Coruña, SPAIN
{enano,gulias,juanjo,sjorge}@lfcia.org
http://www.lfcia.org

Abstract. A Video-On-Demand (VoD) server provides video services to the end user, that can request a piece of video at any time, without any previously established timetable. The growing demand of such services suggests the design of flexible and scalable VoD servers, both in storage capacity and bandwidth. The tertiary level of a VoD server that is being implemented in top of a cheap Linux cluster, based on a hierarchical distributed architecture, using the functional programming language Erlang, is proposed in this paper.

1 Introduction: What Is a VoD System?

A Video-On-Demand (VoD) server provides video services to the end user, that can request a piece of video at any time, without any previously established timetable. Some examples of the use of such a system are: film-on-demand, remote learning, home shopping and interactive news.

The growing demand of such services suggests the design of flexible and scalable VoD servers, both in storage capacity and bandwidth.

The main requirements that must be taken into account at the analysis and design stage of such a system are:

- Fault tolerancy: the system, that usually is going to work 24x7, must be reliable. Changes in the code should be done without stopping the system.
- Large storage capacity: Thousands of video objects (news, movies or even pieces of video provided by the client)
- High bandwidth distributed among high number of concurrent users.
- Scalability: as the number of users grows, it should be possible to add new resources to the system in order to expand its performance.
- Low cost: the use of commodity hardware at the underlying architecture should give us a considerable reduction in the final price.
- Predictable (low) response time: the solutions should provide statistical estimations of the time that the client must wait until being served

* This work was supported by EC-CICYT Research Project 1FD97-1759 and XUGA PGIDT99COM1052
The paper is structured as follows. First, a brief introduction on the state-of-the-art of VoD servers is presented. In section 3, the proposed solution is shown, putting emphasis on the hierarchical layout of the system and the language used for implementing it. In section 4, the tertiary level of the hierarchical architecture is presented, giving an exhaustive explanation of its three subsystems: scheduling, input/output and supervision. As a subsection, a real scenario that illustrates how the tertiary level works when a user makes one video object request is presented. Finally, some conclusions are presented.

2 State of the Art

In the enterprise world there are different types of products, some more focused in LAN such as the Oracle Video Server [12] and IBM DB2 Digital Library Video Charger [3], other more focused to the Internet, such Real Networks RealVideo Server, Sun StorEdge Media Central or Apple Darwin Streaming Server (OpenSource solution). There are also companies that build ad hoc solutions, frequently using the above systems, such as: nCube, Diva TV, Sea Change or TeraCast.

All these applications have as common features that used to be expensive and non-scalable solutions. Besides, they use to be monolithic solutions whose adaptation to a complex network architecture, in order to manage, for example, the hierarchical bandwidth of a MAN cable network, is almost impossible.

In the academic world there are a lot of more theoretical works: cache optimisation, layered video delivery, video merging, etc. The approach we are going to present is partially inspired on [4]. There are some implementations such as Stony Brook Video Project [5,6,7] and Shared-Memory Server [8].

3 A Functional Cluster Based VoD System

A hierarchical storage system, based on a cheap Linux cluster [9], is proposed to meet the requirements.

For the high level system, the functional programming language Erlang [10] has been used as development language. Only in low level modules with special performance requirements C language has been used.

When designing the system, both the architectural parameters and operational procedures had to be taken into account. The former referees to the hardware design (bandwidth and storage capacity at each level) and the later to things like the admission policy (the criteria used to accept a user request), scheduling algorithm (used to decide how the video objects are moved across the levels of the hierarchy) or file replacement algorithm (which movie is removed in a given level of the system).

3.1 VoD Server Hierarchy: Hardware Layout

At the figure 1 the three level hierarchy of the system is shown. In spite of this static architecture, software design of the system lets us to create a very flexible
schema in which the number of levels can be variable. Three level standard architecture is presented here.

![VOD Server Hierarchy](image)

**Fig. 1. VOD Server Hierarchy**

- **Tertiary level (Massive Storage):** The massive storage level does not have the same requirements in terms of response time as the higher levels. The goal here is to store all the video objects available at the VoD server in a tape charger. The abstract view of this level is a mechanical arm in charge of loading the tapes in one of the reader units, as long as there are not enough readers for every tape in the storage. The performance of the server is constrained at this level by loading time, latency, throughput, and so on. Even though it is desirable to optimize these quantitative parameters, the next level alleviates this performance constraints because it acts as a cache for the video objects in the tertiary level.

- **Secondary level (Cache Stage):** It is composed of a group of nodes with disk storage large enough to host (at least) a complete title. Thus, a title read from the tertiary level is stored temporarily at this point before being striped at the primary level. An appropriate scheduling policy allows keeping the needed movies at this level when required, avoiding (most of) the accesses to the massive storage.

- **Primary level (Striping Stage):** It is composed of a group of nodes in charge of splitting the movie into frames and delivering them. As long as there is more than one node at this level, it is possible to recover from a failure in one of the striping computers by assigning its duties to another node. This level has important requirements in bandwidth and response time.
3.2 Target Architecture: Prototype Beowulf Cluster

The figure 2 shows the LFCIA’s Linux Beowulf cluster used for the video server implementation. It has a frontend and up to 22 nodes. The frontend is a Dual Pentium II 350MHz 384MB. Each node is an AMD K6 300-266MHz, 96MB. All the nodes are linked using a switched 100Mb Fast Ethernet.

![Fig. 2. The borg: LFCIA’s Linux Cluster](image)

3.3 Erlang: A Concurrent Functional Programming Language

**Erlang Basics** Erlang [10] is a concurrent functional programming language whose design features, such as code hot swap, makes it suitable for programming concurrent, soft real-time, distributed fault-tolerant systems. Functional language means that programming in Erlang is based on the notion of function and that values are immutable, that is, Erlang is a single assignment language. Iterative loops are replaced by explicit recursion and conditional execution by pattern-matching analysis. The language provides rich compound data structures such as (a) tuples (for example `{a, 12, b}`, `{}`, `{1, 2, 3}`, `{a, b, c, d, e}`) used for storing a fixed number of items and are written as sequences of items enclosed in curly brackets; (b) lists (for example `[]`, `[a,b, 12]`, `[22], [a, 'hello friend']`) used for storing a variable number of items and are written as sequences of items enclosed in square brackets; and (c) records (for example `-record{file, {file="a.out", size=198}}`), intended for storing a fixed number of data items, which main advantage is that its fields are accessed by name.

In order to control a set of parallel activities, Erlang has primitives for multiprocessing: `spawn` starts a parallel computation (called a process) and delivers the identity of the newly created process (its `pid`); `send` sends an asynchronous message to a process; and `receive` receives a message from a process. The syntax
Pid ! Msg is used to send a message Msg to the process with identity Pid. As most functional languages, complex data types as well as functions are handled as first-class values; that means that Erlang provides suitable marshaling for those structures with no extra programming effort. The selection of incoming messages is performed by pattern-matching analysis in each process mailbox. The following example shows the implementation of a process server that squares the input received from a client:

```erlang
-module(server).
-export(loop/0).

loop() ->
    receive {From, Message} ->
        From ! Message * Message,
        loop()
    end.
```

In order to create a server process at node `server@borg1.dc.fi.udc.es`, the following expression should be evaluated binding Server to the server identity.

```erlang
Server = spawn('server@borg1.dc.fi.udc.es', server, loop, []).
```

The interaction is initiated by sending a tuple with the local identity as well as the number to square to the server process: `Server ! {self(), 10}`. The response of the server is received at client’s mailbox.

**Erlang Main Features** This language has been used for programming the distributed system, because of the interesting features offered to write complex real-time fault-tolerant distributed processing systems using the message-passing paradigm. Erlang is a functional language developed by Ericsson, the large telecommunications company, and it has suitable features for the implementation of soft-real-time fault-tolerant distributed processing systems and has interesting features such as the functional approach (functions, single assignment), code replacement without stopping the system (Hot Swap), fault tolerance built-in primitives, message passing with primitives for multi-processing and behaviours (implementation patterns).

These features, and the Open Telecom Platform (OTP) -lots of useful libraries such as Mnesia (a distributed database) and operation and maintenance applications (EVA, MESH, ...)-, make the language very interesting for meeting the system requirements.

Erlang’s Behaviours are very similar to the GoF’s Design Patterns. They are implementation patterns in Erlang. For instance, the `gen_server` (generic server) is the implementation in Erlang of the Client-Server Pattern. Its use allows code hot swap in the server, stanitarized API and easy standard interaction with Erlang tools.

A module with `gen_server` behaviour has to implement a standard API: `start, start_link, stop...` and the required callback function `handle_call:`
behaviour associated with a call. When the generic server receives a request, it 
delegates the handle to the callback function implemented by the programmer 
in each case. The client uses \texttt{gen\_server:call} as the standard API to access 
the server.

4 The Tertiary Level

Due to the modular hierarchical system design, the tertiary level has low re-
quirements in terms of response time (alleviated by the next level). Its main 
goal is to store all the video objects in different protocols (tape charger, files, 
HTTP remote server, etc.) It is divided into three main subsystems: scheduling, 
input/output and supervision.

4.1 Scheduling Subsystem

The scheduling subsystem receives secondary level requests for video object 
charging and time estimation, which can be done with or without resource reser-
vation depending on a higher level decision. A unique dependency graph (implen-
tmented as a combination of queues and lists) for all the different resources (arms 
and heads) is constructed attending to different scheduling policies. Every time 
a new request is received, the scheduling algorithm decides where in the graph 
the subtasks needed to complete the new request should be carried out, always 
considering the dependencies with the rest of the graph.

A distributed client/server implementation based in Erlang/OTP is proposed 
for the scheduling subsystem. Especial attention is paid to the performance of 
this subsystem, for example, by implementing queues as an ADT which uses 
the well-known two lists implementation \[11\]. The \texttt{sched} module is an Erlang 
gen\_server behaviour (partially modified) with an API that must be known by 
the external modules, this module communicates with the \texttt{gen\_servers} of each of 
the physical resources. Plug-and-play independent modules can be used for the 
different scheduling algorithms (LRU, FIFO, Random,\ldots).

4.2 Input/Output Subsystem

A data movement abstraction: \textit{pipes} (similar to the UNIX ones), is proposed 
for the data communication. A \textit{pipe} has as its creation parameters the source 
and destination (both Erlang modules that implement the send and reception 
protocols plus some initialization parameters), and some general options about 
the way the transmission must be done. The Erlang modules implement three 
mandatory functions: \texttt{init} (protocol initialization), \texttt{proc} (read and write) and 
\texttt{done} (destructor).

The \textit{pipe} works as a supervisor: if there is any problem with the transmission, 
propagates the error. By using the \textit{pipe} abstraction, any two levels of the server 
can be interconnected. Indeed, a whole server could play the role of a source for 
the tertiary level of other server.
Generic Pipe Wrapper with options (very similar to Erlang gen_tcp) is proposed for the implementation of special transmission types. The generic pipe can create pipes of different kinds. CBRPipe (the implementation of CBR transmission) has an internal buffer and commits to a BPS rate taken as initialization parameter (\{bitrate,{BPS}\}).

**A Sample Scenario with Two Layers Hierarchy** On top of figure 3 the structure of the video server in a two level configuration can be seen: one streaming primary level and a tertiary (massive storage) level. Streaming level is composed by a local scheduler, where goes the system intelligence, and a composer that manages the different protocols the server uses to communicate with its clients (HTTP, H263, RTP). Storage level is composed by its local scheduler and the composer, that manages the different kinds of storage that are used for the server (file, tape or another hierarchical group of storages). Each level is created and supervised by a vodka_slave and can have associated monitors.

In the bottom part of the schema, the management application that works on top of the server is presented. A dynamic web server (XML/XSL+Servlets) with two databases (one with relevant video system data and the other with management information) works as the standard interface with the user. A mediator process is in charge of updating the database with system data gathered from the server.

When one user asks for the list of available videos using HTTP protocol, the system consults the database and builds the dynamic web page with all the URLs of the video objects. The user chooses the one he wants to watch using the browser and this new request is already redirected to the video server using the proper protocol. In the sample scenario the request is redirected to one of the HTTP ports in which the system is waiting.

The “lookup” command, with the VO (Video Object) identifier, is then propagated by the system until the local scheduler, that checks if the VO is available at its own level; as it isn’t, it delegates the responsibility to the massive storage level scheduler (this delegation pattern and the complete independency between the levels -only a common API- makes very easy the introduction of a new level, without the need of changing the algorithm). The local scheduler asks then to the protocol composer, that passes this question to all the subsystems that control the different storage types. All of them give one response, indicating both the VO availability and information about the quality, bandwidth, charge speed, etc. The composer puts all this information together and gives it back to the scheduler, that analyzes it and, based on its scheduling policies, makes the decision about what is going to be the used storage kind. Then, it answers to the streaming level scheduler, that propagates the answer towards the HTTP manager.

At this moment, the direct communication between the selected two processes (the HTTP selected at the streaming level by the user, and the file selected in this example by the scheduler at the massive storage level) can be started.
The HTTP manager initializes its *pipe*, that waits at the local port, and requests the file manager to send the VO to that port, adding to this information the protocol that should be used for the transmission. The *pipe* in charge of the communication between the file and the given protocol is created at this moment at the storage level. Finally, the VO transmission process between the two levels is done. As the video gets the streaming stage, it is sent to the end user by HTTP using the other *pipe*.

**Fig. 3.** Scenario with two layers (streaming and massive storage) hierarchy

### 4.3 Supervision Subsystem (Monet and Erlatron)

Supervision subsystem: Monet [4] is used for system supervision. Results of supervision tasks are generated in XML and transformed to many formats by using Erlatron, a distributed XSLT processor, also implemented in Erlang using a C++ library (Sablotron [5]).

**Monet** It is a simple and flexible monitoring tool, suitable for network monitoring, host performance tracking and for the instrumentation of complex systems. The whole foundation for Monet is the Erlang/OTP platform, including MESH,
EVA, SASL and Mnesia. Monet was initially intended as a replacement for simple monitoring tools as Mon, with additional support for variable tracking and graphing, as well as more complex instrumentation features.

Monet follows the MESH and EVA design, leveraging the infrastructure they provide and extending it. Figure 4 presents the modular structure of the sub-system. For each measurement an MO (measurement object) is created, supervised by an MRP (measurement responsible process). A master handler (main_handler) receives MESH and EVA events and alarms, and may invoke specific action handlers according to complex configured conditions. Event and alarm logging is performed through the standard MESH logs; additionally, custom loggers can be plugged into main_handler or directly into EVA. The user interface is implemented through a web adaptation, using Erlatron and INETS from erlets.

Fig. 4. Monet process structure

Instead of defining MESH Measurement Objects one by one, Monet creates a tree of monitored object classes, together with the measurement objects that will take objects of that class as monitored resources, like the one shown in figure 5. Each object class — a tree node — can be further specialized as desired.

Leaf nodes usually represent individual hosts or resources, but are not special in any way.

The tree is traversed from the root towards a node to determine the measurement classes suitable for that node.

Resources to be monitored are defined as a directed graph, reflecting the logical grouping as managers see them. This logical grouping is completely artificial, and can be based on their physical, topological or simply organizational structure. Note that, despite its name, this structure is a directed graph and not strictly a tree, because branches can merge at any point (see figure 6 for an
example). It is supposed to make sense to humans and has no other constraints, and does not even require all monitored resources to be present.

Each resource can have an attached list of classes; these classes position the resource in the classes tree, thus implicitly declaring the measurement objects that will monitor it.

Resources can also contain additional data in order to store their physical position, network connections, desired graphical representation, etc. This information can then be used by measurement objects to get additional configuration or by the user interfaces when representing the tree.

In order to ease the development of simple, common monitors, Monet provides two generic measurement types which act as bridges between MESH and
Erlang functions or Unix executables, respectively. Thus, existing monitor scripts and executables — such as those from Mon — or Erlang code can be used as monitors without additional coding.

Alarm handlers are called from main_handler when the appropriate alarms and conditions are triggered. Again, they can be either external executables or Erlang functions.

Whenever an event or alarm is received, Monet checks whether it should call a handler. Alarm destinations are currently defined as functional expressions in order to define complex conditions, such as:

```erlang
[
  % Call dumphandler whenever an alarm is received.
  % (ie., when stddeps:always/5 returns true)
  { always, dumphandler},

  % Call myHandler if the sender is enano2@borg
  { {sender, [enano2@borg]}, myHandler},

  % Call thisHandler if extmodule:extfunc/5 returns true (called as
  % extmodule:extfunc(Name, Sender,
  % Severity, Class, [LocalArgs]))
  { {extmodule,extfunc,[localargs]},thisHandler},

  % Call dumper if either always()
  % or never() are true:
  { {any, [{always,[]},{never,[]}]}, dumper },

  % Call selHandler if the two former are true:
  { {all, [{any, [{always,[]},{never,[]}]},
    {extmodule,extfunc,[localargs]} ]},
    selHandler}
]
```

This allows for composition of arbitrarily complex conditions as long as they can be expressed in Erlang. The use of higher-order functions to express complex configuration conditions is also employed in stylesheet selection.

As long as Monet information can be retrieved from different devices, it is advisable to provide a general mechanism to adapt this content to the specific features of such output devices. Contrary to when style information is hard-coded into the content, separation of style from content allows for the same data to be presented in different ways. This enables:

- Reuse of fragments of data,
- Multiple output formats,
- Styles tailored to the reader’s preference,
- Standardized styles,
- Freedom from style issues,
- Easy interface with third-party software.
In order to achieve such goals, Monet generates most of its results as XML documents which can be transformed at a later stage according to an XSL stylesheet.

Erlatron To perform the XSL transformation, Monet uses Erlatron, a distributed XSLT processor implemented in Erlang using a C++ library (sablotron).

Figure 7 presents the actors involved in the Erlatron subsystem. Erlatron adapts the Sablotron library by using a port which performs basic transformations of a couple of Erlang binaries representing the XSL stylesheet and the XML source. Sablotron is based on the Expat library [14] and has been designed for performing fast and compact transformations. The port is managed by a generic server that offers XSLT services to any client. This slave server constitutes the basic processing unit and, considering the CPU cost of performing XSL transformations, low additional overhead is expected when used from Erlang.

In order to exploit a distributed framework, such as the Beowulf cluster introduced in [9], a simple master/slave architecture is deployed. In this setup, a master server is used to distribute requests to different slave servers running on a pool of computer nodes. The state of the master server is a collection of pending transformations as well as the information about idle slaves. The dispatching of requests, carried out by the master scheduler, consists of pairing a pending transformation with an idle slave server. Figure 8 shows the interaction among actors when solving an XSLT service.

As the reader can guess, Erlatron architecture seems to be quite interesting for web sites with a high number of requests which involve many XSL independent transformations for dynamic content generation. We are going to present some preliminary results when using Erlatron on the cluster presented in section 3.2. The frontend runs INETS, serving a 64KB HTML generated using a 25KB XML document and a small XSL stylesheet; each node hosts an Erlatron slave.
Left side of figure 8 shows the requests per second achieved when running Apache Bench (ab), a tool for benchmarking an HTTP server, at the frontend. As new XSLT processors are added to the slave pool, the server is able to increase its service rate until the concurrency level \( C \), number of simultaneous requests, \(-c\) matches the pool size. Right side of figure 9 presents the time taken to attend a collection of requests, varying the concurrency level.

5 Conclusions and Future Work

The chosen hardware architecture has proved to be quite interesting both for its low cost and for its scalability when comparing with traditional proprietary systems. The hierarchical architecture seems to be very effective for managing the cost of the whole system while meeting the requirements. Tertiary level must have a great storage capacity, but does not need such a low latency and high bandwidth as the secondary one, for instance. Besides, this design helps the modular division of the global system, making it easier to create and maintain.

The use of Erlang as the development language has simplified the tertiary level implementation if compared with other alternatives for moving code across a cluster of PC like Java/RMI.
Design patterns and implementation patterns in Erlang simplify the maintenance of processes. The system is very scalable because of its modular hierarchical design, the low cost of the addition of a node, and the flexibility of the implemented software (design patterns, clean message passing).

As further work: final implementation of the complete hierarchy and testing of the system on a real environment (metropolitan cable network) should be done in the near future.

The functional approach, avoiding side effects, has improved the system reliability. The main contributions are the abstraction (high level of programming) and the referential transparency, which makes much more easier to deal with the intrinsic complexity of the concurrency.

The Erlang functional language has also contributed with some of its already described very interesting features: distributed message passing philosophy, behaviours. Similar to GoF’s Design Patterns, and lots of libraries, tools and applications.

References

Hidden Specification of a Functional System*

César Domínguez, Laureano Lambán, Vico Pascual, and Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja.
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain).
{cedomin,lalamban,mvico,jurubio}@dmc.unirioja.es

Abstract. This paper is devoted to the formal study of the data structures appearing in a symbolic computation system, namely the EAT system. One of the main features of the EAT system is that it intensively uses functional programming techniques. This implies that some formalisms for the algebraic specification of systems must be adapted to this functional setting. Specifically, this work deals with hidden and coalgebraic methodologies through an institutional framework. As a by-product, the new concept of coalgebraic institution associated to an institution is introduced. Then, the problem of modeling functorial relationships between data structures is tackled, giving a hidden specification for this aspect of the EAT system and proving the existence of final objects in convenient categories, which accurately model the EAT way of working.

1 Introduction

EAT (Effective Algebraic Topology) [15] is a symbolic computation system created by Sergeraert to calculate homology groups. Algorithms in Algebraic Topology are based on the handling of infinite data structures and then functional programming is necessary in order to obtain actual implementations. This feature, together with some efficiency requirements, led Sergeraert to use Common Lisp as programming language. When a system is both complex and relevant (providing knowledge which before was unreachable, neither by “hand” nor by other mechanical methods), software reliability becomes a main concern. Then some kind of formal analysis of the system is mandatory. In order to undertake this task, two alternative lines are possible. On the one hand, it is possible to map its actual programs in any well-known formalism (dealing with functional programming, some kind of λ-calculus is an obvious candidate) and, simultaneously or subsequently, to develop an adequate type system for the software package (this step would be unavoidable when working with the by-default Common Lisp implicit typing strategy). On the other hand, it is possible to analyze data structures in a more abstract way, generating an algebraic specification for the system and relying on the notion of abstract data type implementation, in order to fill the gap between the formalism and the concrete programs.

* Partially supported by DGES, project PB98-1621-C02-01 and by Universidad de La Rioja, project API-00/B28
Since EAT is a symbolic computation system based on algebraic structures (a typical operation is: given a simplicial set, constructing its corresponding chain complex; another simpler example would be: given a group, constructing its corresponding group ring), the second approach was chosen to start the formal analysis of the EAT data structures. A consequence of the results obtained up to now \[10, 11\] is that, in our case, the two above-mentioned lines seem to converge. The reason is that we have proved in \[11\] that the EAT data structures are intimately related to final objects in hidden algebraic specifications, objects which we can describe as sets of tuples of functions, as in Cardelli’s approach to object-oriented calculus \[1\]. In this paper we extend these previous results, in such a way that functors between categories can be \textit{internalized} as operations in hidden specifications. Since the final objects in this generalized context are always based on the tuples of functions suggested by EAT, the relationship with Cardelli’s models is reinforced.

The paper is organized as follows. In the following section, the problems are illustrated by means of examples expressed in some well-known programming languages. Section 3 is devoted to introducing some previous results by using the \textit{institutional framework}. The relationship with coalgebraic methods is also studied introducing \textit{coalgebraic institutions} (definition which, up to the authors’ knowledge, did not previously appear in the literature). The main new result, related to functorial constructions between specifications, is given in Section 4. The paper ends with a section devoted to conclusions and future work.

2 Examples

In a system like EAT, algebraic structures must be defined and handled at runtime. For instance, if we are interested in computing \textit{with} groups (and \textit{within} each group), the following signature should be considered:

\[
\text{GRP} : \begin{align*}
\text{prd} & : g \times g \rightarrow g \\
\text{inv} & : g \rightarrow g \\
\text{unt} & : \rightarrow g
\end{align*}
\]

and some \textit{GRP}-algebras should be represented into the computer memory. (Let us remark that if, a posteriori, an algebraic-based semantics analysis on such a system is undertaken, something similar to an \textit{auto-referential} situation will be obtained, since signatures \(\Sigma, \Sigma\)-algebras and the like will be both analysis tools and parts of the system under observation.) In order to implement such universal algebra structures, a candidate for representing signatures should be chosen. Note also that some kind of \textit{genericity} should be present in the system, if we want to compute with groups on \textit{different} sets (that is to say, the data type corresponding to the sort \(g\) should be, in a sense, \textit{variable}). For instance, a first attempt in Java to represent the signature \textit{GRP} could be:
public interface Group {
    Object prd (Object a, Object b);
    Object inv (Object a);           \text{And then a concrete group,}
    Object unt ();
}

the additive group on the integer numbers set $\mathbb{Z}$ for example, should be represented as a class implementing this interface:

public class GroupInt implements Group {
    Object prd ...
}

The use of the root class Object has two important consequences. Firstly, some verbose casting (explicit coercion) has to be included. Secondly, the operations have to be partial (in the example, only integer numbers can be operated by means of prd).

In Haskell [8], another paradigmatic programming language, a signature could be represented by a type class:

class Group a where
    prd: a $\rightarrow$ a $\rightarrow$ a
    inv: a $\rightarrow$ a
    unt: $\rightarrow$ a

and then a concrete group would be defined as:

instance Group Int where prd = ...\text{Let us remark that in the Haskell approach no casting is necessary (Group can be considered as an actual generic data type, while in Java genericity can be simulated through the class Object) and no partiality appears. Nevertheless, this approach is, although cleaner, more rigid than the previous one, because in a program fragment, only one group on the integer numbers set can be considered (since Group Int is the name of a unique data type in Haskell).}

If we imitate the data structures which appear in EAT, the signature would be represented by a Common Lisp structure (record) such as:

(defun struct group prd inv unt)

where each field is intended to store a functional object (a Common Lisp lexical closure [17]) implementing one of the operations of the group. For instance, we could define:

(setf GroupInt (make-group :prd #'(lambda (x y)

Since we are relying on the implicit, dynamic, typing strategy of Common Lisp, no casting is necessary, but, in order to obtain theoretical models, the maps will be partial. So, the EAT approach is closer, in this aspect, to Java.

Another feature of EAT is that algebraic structures must be constructed at runtime. For instance, the cartesian product (direct product) of two groups can be programmed:
Note the use of functional programming through the operators `lambda` and `funcall`. This can be achieved in EAT, because it is based on a mixture of (Common Lisp) records and functional programming, just as in Cardelli’s approach to object-oriented programming [1]. Even if the example is too simple to show this feature, this way of coding algebraic structures in EAT allows the programmer to deal with infinite data structures: the instances of a record such as `group` are purely behavioral objects, no information on the underlying set is explicitly stored (this could be more clearly illustrated by means of a functorial construction building, from a finite algebraic structure, an infinite one; the loop space functor is a typical construction of this kind, in the field of Algebraic Topology [14]).

It is quite clear that this approach cannot be directly translated into Java, with the representation pattern previously introduced, because Java does not allow to define classes at runtime.

In order to analyze the same question in Haskell, we will rely on the initial proposal of Wadler and Blott [19] for translating type classes into plain Standard ML. Their solution is based on the concept of dictionary. A dictionary acts as a repository for the functional components of each instantiation of a type class. In our example this runs as follows:

```
data GroupD a = GroupDict (a -> a -> a)
               (a -> a)
               a
```

Then a companion operator is defined for each component:

```
imp-prd :: GroupD a -> a -> a -> a
        ...
```

and a particular call would be something like:

```
imp-prd GroupInt 5 3
```

Two observations should be made at this point. Firstly, this specific implementation of type classes opens the door to operate with algebraic structures at runtime (but recall the limitation remarked above: we would not be able, with this na"ıf approach, to build the cartesian product of two different groups on the same set, of integer numbers, for instance). Secondly, and rather surprisingly, the Wadler-Blott’s proposal in 1988 is very close to the solution found by Segeraert around that date, in an independent way and in a very distant application field (Symbolic Computation versus Programming Language Theory). In effect, emulating the EAT style in our toy example, a family of functions for dealing with groups would be defined:

```
(defun imp-prd (group x y)
  (funcall (group-prd group) x y)
```
where the expression \((\text{group-prd } \text{group})\) directly corresponds to the extraction of a component from the dictionary in the Wadler-Blott approach.

In order to undertake the analysis of the EAT system, from the algebraic specification perspective, we introduced a general construction on Abstract Data Types \([10]\). From the syntactical point of view, given a signature \(\Sigma\) a new signature \(\Sigma_{\text{imp}}\) is constructed. For example, if we consider the signature for a group, we obtain:

\[
\begin{align*}
\text{GRP}_{\text{imp}} & : \text{imp-prd} : \text{imp-g} \times \text{g} \times \text{g} \rightarrow \text{g} \\
& : \text{imp-inv} : \text{imp-g} \times \text{g} \rightarrow \text{g} \\
& : \text{imp-unt} : \text{imp-g} \rightarrow \text{g}
\end{align*}
\]

This accurately models both the Haskell and the EAT approaches. The terminology “-imp” is ours, as it is not present in Wadler-Blott’s nor in Sergeraert’s papers. The reason for this notation is that, in \([10]\), we explained that this is a construction for modeling (the physical, computer-memory, parts of) implementations of algebraic structures, as groups in our current example. The general construction is presented in the following section in a new, institutional, way.

3 An Institutional Approach

As explained in the previous section, several years ago we started up the formal analysis of the EAT system. Through that work we have proved that the Common Lisp data structures used in EAT for storing algebraic and combinatorial structures (such as graded groups, simplicial sets and so on) can be considered (as a part of) final objects in certain categories of abstract data types implementations. In this section we present in a purely algebraic setting (without referring to implementation or programming language issues) and within the institutional framework, some of the results obtained in \([10], \[11]\). We start recalling the definitions of institution and institution morphism presented as in \([5]\).

**Definition 1 (Institution).** An institution \(\mathcal{I}\) consists of:

1. a category \(\text{SIG}\), whose objects are called signatures,
2. a functor \(\text{Sen} : \text{SIG} \rightarrow \text{Set}\), assigning to each signature \(\Sigma\) a set whose elements are called \(\Sigma\)-sentences,
3. a functor \(\text{Mod} : \text{SIG} \rightarrow \text{Cat}_{\text{op}}\), giving for each signature \(\Sigma\) a category whose objects are called \(\Sigma\)-models, and whose arrows are called \(\Sigma\)-morphisms, and
4. for each \(\Sigma \in \text{Obj}(\text{SIG})\), a relation \(\models_\Sigma \subseteq \text{Obj}(\text{Mod}(\Sigma)) \times \text{Sen}(\Sigma)\), called \(\Sigma\)-satisfaction,

such that for all the morphisms \(\phi : \Sigma \rightarrow \Sigma'\) in \(\text{SIG}\), the Satisfaction Condition

\[
m' \models_{\Sigma'} \text{Sen}(\phi)(e) \iff \text{Mod}(\phi)(m') \models_\Sigma e
\]

holds for each \(m' \in \text{Obj}(\text{Mod}(\Sigma'))\) and each \(e \in \text{Sen}(\Sigma)\).
**Definition 2 (Institution morphism).** Let \( \mathcal{I} \) and \( \mathcal{I}' \) be institutions. Then, an institution morphism \( \Phi : \mathcal{I} \to \mathcal{I}' \) consists of:

1. a functor \( \Phi : \text{SIG} \to \text{SIG}' \),
2. a natural transformation \( \alpha : \Phi; \text{Sen} \to \text{Sen} \), and
3. a natural transformation \( \beta : \text{Mod} \Rightarrow \Phi; \text{Mod}' \)

such that the following Satisfaction Condition holds:

\[
m \models_\Sigma \alpha_{\Sigma}(e') \iff \beta_{\Sigma}(m) \models_{\Phi(\Sigma)} e'
\]

for any \( \Sigma \)-model \( m \) from \( \mathcal{I} \) and any \( \Phi(\Sigma) \)-sentence \( e' \) from \( \mathcal{I}' \).

Let \( \mathcal{E} = (\text{SIG}, \text{Sen}, \text{Mod}, \models) \) be the well-known equational algebraic institution (see, for instance, [12], [4]). The above-mentioned results on EAT data structures can be modeled by means of the following endomorphism on the algebraic institution \( \mathcal{E} \):

- The functor \( \Phi : \text{SIG} \to \text{SIG} \) assigns to each signature \( \Sigma = (S, \Omega) \) the signature \( \Sigma_{\text{imp}} \) given by: \( \Sigma_{\text{imp}} = (S_{\text{imp}}, \Omega_{\text{imp}}) \) where \( S_{\text{imp}} = \{ \text{imp}_\Sigma \} \cup S \), \( \text{imp}_\Sigma \) being a fresh symbol (i.e. \( \text{imp}_\Sigma \notin S \)), and \( \Omega_{\text{imp}} = \{ \text{imp}_\omega : \text{imp}_\Sigma \ s_1 \ldots s_n \to s \ | \ \omega : s_1 \ldots s_n \to s \in \Omega \} \) (i.e. the new sort \( \text{imp}_\Sigma \) is added as first argument for the operations in \( \Omega_{\text{imp}} \)). For each morphism \( \mu : \Sigma \to \Sigma' \), we define \( \Phi(\mu) : \Sigma_{\text{imp}} \to \Sigma'_{\text{imp}} \), where \( \Phi(\mu)(s) := \mu(s) \) if \( s \in S \), \( \Phi(\mu)(\text{imp}_\Sigma) := \text{imp}_{\Sigma'} \), and \( \Phi(\mu)(\text{imp}_\omega) := \text{imp}_{\omega'} \) being \( \omega' = \mu(\omega) \).

- The natural transformation \( \alpha : \text{Sen} \circ \Phi \Rightarrow \text{Sen} \) is defined through the family of mappings \( \alpha_{\Sigma} : \text{Sen}(\Sigma_{\text{imp}}) \to \text{Sen}(\Sigma) \), one map \( \alpha_{\Sigma} \) for each \( \Sigma \) in \( \text{SIG} \), where \( \alpha_{\Sigma} \) acts on a sentence in \( \Sigma_{\text{imp}} \) by erasing all the variables of sort \( \text{imp}_\Sigma \) and replacing each occurrence of an operation \( \text{imp}_\omega \) by the corresponding operation \( \omega \) in \( \Sigma \). (In order to get a well-defined function it is necessary to exclude from the sets of sentences the equalities between variables of the distinguished sort; but it is clear that excluding these trivial sentences, the expressiveness of the institution \( \mathcal{E} \) is not impoverished.)

- The natural transformation \( \beta : \text{Mod} \Rightarrow \text{Mod} \circ \Phi \) is defined through the family of functors \( \beta_{\Sigma} : \text{Mod}(\Sigma) \to \text{Mod}(\Sigma_{\text{imp}}) \), one functor \( \beta_{\Sigma} \) for each \( \Sigma \) in \( \text{SIG} \). On objects in \( \text{Mod}(\Sigma) \), the image by \( \beta_{\Sigma} \) of a \( \Sigma \)-algebra \( A \) is the \( \Sigma_{\text{imp}} \)-algebra such that \( (\beta_{\Sigma}(A))_{\text{imp}_\Sigma} := \{ * \} \) (i.e. the carrier set for \( \text{imp}_\Sigma \) is the singleton \( \{ * \} \)) and \( (\beta_{\Sigma}(A))_s := A_s \). The interpretation of the \( \Sigma_{\text{imp}} \)-operations in \( \beta_{\Sigma}(A) \) is the natural one:

\[
\beta_{\Sigma}(A)(\text{imp}_\omega)(*, d_1, \ldots, d_n) = A(\omega)(d_1, \ldots, d_n).
\]

On morphisms in \( \text{Mod}(\Sigma) \), for each morphism of \( \Sigma \)-algebras \( f = (f_s)_{s \in S} : A \Rightarrow B \), we define \( \beta_{\Sigma}(f) : \beta_{\Sigma}(A) \Rightarrow \beta_{\Sigma}(B) \) by \( \beta_{\Sigma}(f) := (\text{id}_{\{ * \}}, (f_s)_{s \in S}) \).

The intuitive interpretation of this endomorphism on \( \mathcal{E} \) is to consider that from a signature \( \Sigma \) a new signature \( \Sigma_{\text{imp}} \) is constructed in such a way that models of \( \Sigma_{\text{imp}} \) correspond to families of models of \( \Sigma \). Nevertheless, from a
programming point of view (and, in particular, for our work on EAT) it is more natural to implement data structures (groups, for example) whose elements are represented following an equal pattern in the computer memory. In our context, this idea corresponds to fix, given a signature $\Sigma = (S, \Omega)$, a data domain $D = \{D_s\}_{s \in S}$ and then to consider only families of $\Sigma$-models with carrier sets in $D$. In order to formalize this situation we introduce the *equational algebraic institution* on a data universe $D$ as follows.

Let us consider a set $U$: the *sorts universe*. Then, for each $s \in U$, a non-empty set $D_s$ is also fixed. The family $D = \{D_s\}_{s \in U}$ is called *data universe*. The *equational algebraic institution* on a data universe $D$, denoted by $\mathcal{E}^D = (\text{SIG}_\mathcal{E}^D, \text{Sen}_\mathcal{E}^D, \text{Mod}_\mathcal{E}^D, \models_{\mathcal{E}^D})$ is just defined as the equational algebraic institution $\mathcal{E}$, except that for each signature $\Sigma = (S, \Omega), \text{Mod}_\mathcal{E}^D(\Sigma)$ is the category of $\Sigma$-algebras such that $A_s = D_s, \forall s \in S$ and the morphisms in $\text{Mod}_\mathcal{E}^D(\Sigma)$ are only the identity morphisms. Since it is evident that $\mathcal{E}^D$ is a subinstitution of $\mathcal{E}$ (in a sense easy to formalize), the previous endomorphism on $\mathcal{E}$ is particularized into a morphism of institutions from $\mathcal{E}^D$ to $\mathcal{E}$.

In this way, we have bounded the domain of our institutional morphism. In the same sense, it is quite clear that the range $\mathcal{E}$ is too wide: only those $\Sigma_{\text{imp}}$-algebras based on $D$ are relevant in our approach. The right institution in which our morphism ranges is the *hidden institution* (see [2], [6]). In a hidden *specification* [7], it is necessary to fix a data algebra (visible algebra) $D$ and then to consider algebras with a visible part on $D$ and another hidden, non-fixed, part. This kind of hidden signatures and hidden algebras on $D$ can be organized in a *hidden institution* on $D$, denoted by $\mathcal{H}^D$. In order to redefine our initial endomorphism on $\mathcal{E}$ to get a morphism from $\mathcal{E}^D$ to $\mathcal{H}^D$ it is enough to declare in a signature $\Sigma_{\text{imp}}$, the distinguished sort $\text{imp}_\Sigma$ (which does not belong to the universe sort $U$) as the unique hidden sort and then the rest of the construction is completed in a natural way. Hence, the objects in $\text{Mod}_{\mathcal{E}^D}(\Sigma)$ are algebraic structures defined on $D$ and the objects in $\text{Mod}_{\mathcal{H}^D}(\Sigma_{\text{imp}})$ are indexed families of such structures, where the elements of the carrier sets for the hidden sort $\text{imp}_\Sigma$ act as indexes for the algebraic structures of the family. This is exactly the framework needed to model the EAT system [10]. In addition, this parallelism can be even further exploited, because the EAT data structures take part of certain final objects which nicely correspond with final objects in categories $\text{Mod}_{\mathcal{H}^D}(\Sigma_{\text{imp}})$ (the existence of final objects in suitable hidden categories is well-known; see [7], for instance). More concretely, the final object in $\text{Mod}_{\mathcal{H}^D}(\Sigma_{\text{imp}})$ can be described, in our particular case, as a set of functional tuples, each tuple encoding the operations of one model in $\mathcal{E}^D$ [10]. This result was suggested by the way chosen by Sergeraert to develop EAT, using intensively functional programming (see [15], [10] and the examples in the previous section). With the help of our institutional morphism $(\Phi, \alpha, \beta) : \mathcal{E}^D \to \mathcal{H}^D$, the final object can be described in a new and instructive way:

**Theorem 1.** Given a signature $\Sigma$ in $\mathcal{E}^D$, the coproduct in $\text{Mod}_{\mathcal{H}^D}(\Sigma_{\text{imp}})$ of the objects in $\beta(\text{Mod}_{\mathcal{E}^D}(\Sigma))$ is the final object in $\text{Mod}_{\mathcal{H}^D}(\Sigma_{\text{imp}})$. 
A third formalism, intimately related to hidden specifications \[3\], allowing us to interpret the EAT data structures, is the theory of coalgebras (see, for instance, \[18\], \[9\]). In order to include coalgebras in our institutional framework we need to introduce the concept of coalgebraic institution associated to an institution, concept which, up to authors’ knowledge, is new in the literature.

Let \( \mathcal{I} = (\text{SIG}_\mathcal{I}, \text{Sen}_\mathcal{I}, \text{Mod}_\mathcal{I}, \models_\mathcal{I}) \) be an institution, with the property that for each \( \Sigma \in \text{Obj}(\text{SIG}_\mathcal{I}) \), \( \text{Mod}_\mathcal{I}(\Sigma) \) is a small category, that is to say, the class \( \text{Obj}(\text{Mod}_\mathcal{I}(\Sigma)) \) is a set. We define the coalgebraic institution associated to \( \mathcal{I} \), denoted by \( \text{CoAlg}(\mathcal{I}) \), as follows:

- The objects of \( \text{SIG}_{\text{CoAlg}(\mathcal{I})} \) are constant functors: for each signature \( \Sigma \) in \( \text{SIG}_\mathcal{I} \), the endofunctor \( F_\Sigma : \text{Set} \to \text{Set} \) which is constant on the set \( \text{Obj}(\text{Mod}_\mathcal{I}(\Sigma)) \)\(^1\) is an object of \( \text{SIG}_{\text{CoAlg}(\mathcal{I})} \). For each morphism \( \mu : \Sigma \to \Sigma' \) in \( \text{SIG}_\mathcal{I} \), the natural transformation \( F_\Sigma \Rightarrow \mu \) \( F_\Sigma \) induced by the functor \( \text{Mod}_\mathcal{I}(\mu) \) is a morphism in \( \text{SIG}_{\text{CoAlg}(\mathcal{I})} \) from \( F_\Sigma \) to \( F_{\Sigma'} \)\(^2\).
- The functor \( \text{Sen}_{\text{CoAlg}(\mathcal{I})} : \text{SIG}_{\text{CoAlg}(\mathcal{I})} \to \text{Set} \) is defined by \( \text{Sen}_{\text{CoAlg}(\mathcal{I})}(F_\Sigma) := \text{Sen}_\mathcal{I}(\Sigma) \) and extended in the natural way on the morphisms.
- The functor \( \text{Mod}_{\text{CoAlg}(\mathcal{I})} : \text{SIG}_{\text{CoAlg}(\mathcal{I})} \to \text{Cat}^{\text{op}} \) is defined by \( \text{Mod}_{\text{CoAlg}(\mathcal{I})}(F_\Sigma) := \text{CoAlg}(F_\Sigma) \) (here \( \text{CoAlg}(F) \) denotes the category of coalgebras on a functor \( F \); see \[18\]) and extended in the natural way on the morphisms.
- The satisfaction condition \( \models^{\text{CoAlg}(\mathcal{I})}_{F_\Sigma} \) with \( F_\Sigma \in \text{Obj}(\text{SIG}_{\text{CoAlg}(\mathcal{I})}) \) is defined by \( A \models^{\text{CoAlg}(\mathcal{I})}_{F_\Sigma} e \) iff \( \alpha(x) \models^{\mathcal{I}} e, \ \forall x \in X, \) where \( A = (X, \alpha : X \to F_\Sigma(X)) \in \text{Obj}(\text{Mod}_{\text{CoAlg}(\mathcal{I})}(F_\Sigma)) \) and \( e \in \text{Sen}_{\text{CoAlg}(\mathcal{I})}(F_\Sigma) \).

If this construction is particularized to \( \mathcal{I} := \mathcal{E}^D \), the equational algebraic institution on \( D \) (this is possible because all the categories of algebras in \( \mathcal{E}^D \) are small, since \( D \) is fixed), it is easy to establish a morphism of institutions from \( \mathcal{E}^D \) to \( \text{CoAlg}(\mathcal{E}^D) \). (The existence of a similar morphism \( \mathcal{I} \to \text{CoAlg}(\mathcal{I}) \) in general is also true if all the morphisms in the corresponding categories of \( \mathcal{I} \)-models are endomorphisms, condition which is trivially satisfied in \( \mathcal{E}^D \).) The very definition of \( \text{CoAlg}(\mathcal{E}^D) \) explains how the canonical morphism \( \mathcal{E}^D \to \text{CoAlg}(\mathcal{E}^D) \) is defined on signatures and sentences. For a given signature \( \Sigma \), we associate to each \( \Sigma \)-algebra \( A \in \text{Obj}(\text{Mod}_{\mathcal{E}^D}(\Sigma)) \) the coalgebra \( \{\ast\}, i_A: \{\ast\} \to \text{Mod}_{\mathcal{E}^D}(\Sigma) \) where \( i_A(\ast) = A \). As in the case of hidden specifications, we can recover the final object of the category of coalgebras \( \text{CoAlg}(F_\Sigma) \) (see \[18\], for instance, for general results on finality in coalgebraic settings), through the coproduct of the objects which belong to the image of this morphism of institutions.

\(^1\) Note that \( F_\Sigma : \text{Set} \to \text{Set} \) is not constant on a particular object of \( \text{Mod}_\mathcal{I}(\Sigma) \), but it is constant on the whole \( \text{Obj}(\text{Mod}_\mathcal{I}(\Sigma)) \), since, \( \text{Mod}_\mathcal{I}(\Sigma) \) being a small category, \( \text{Obj}(\text{Mod}_\mathcal{I}(\Sigma)) \) is simply a set.

\(^2\) The contravariant flavour of the definition is introduced in order to maintain the variancy which is usual when dealing with institutions (that is to say, the targets of \( \text{Sen} \) and \( \text{Mod} \) are \( \text{Set} \) and \( \text{Cat}^{\text{op}} \), respectively).
The three morphisms of institutions previously introduced show three equivalent ways of explaining the formal specification of the EAT system. In addition, the relations between them can be expressed in the following commutative diagram:

\[
\begin{array}{ccc}
\mathcal{E}^D & \xrightarrow{\text{CoAlg}} & \mathcal{H}^D \\
\downarrow & & \downarrow \\
\mathcal{E} & \xrightarrow{\text{forgetful morphism}} & \mathcal{H}^D
\end{array}
\]

where the vertical morphisms are the institutional framework formalization of the relationship between coalgebras and hidden algebras \[3\], and the well-known \[2\] forgetful morphism from \( \mathcal{H}^D \) to \( \mathcal{E} \).

4 Putting Together Different Algebraic Structures

Even if the institutional approach described in the previous section gives a mathematically sound and quite elegant framework for our previous results \[11\], it is too rigid at least for three different reasons, if the aim is to accurately model an actual software system such as EAT:

1. The models are usually defined on quotient sets of the data domain \( D \) (and not on \( D \) itself). To illustrate this point, let us consider the case of groups. With the previous approach, if \( D = \mathbb{Z} \), the set of integer numbers, we are dealing with groups on \( \mathbb{Z} \). But a finite group such as \( \mathbb{Z}/n\mathbb{Z} \) cannot be represented on \( D \). On the contrary, it is quite clear that we can model these finite groups on \( D \) if equalities (i.e. equivalence relations) on \( \mathbb{Z} \) are allowed.

2. The models have partial operations. This is due to the fact that, in the practice, programs are rarely well-defined for each syntactically correct input. In particular, in the field of Algebraic Topology, structures are based on graded sets (usually, the degree is related to some notion of geometric dimension), and this implies that operations are only defined for elements with the same degree; see later the particular cases of simplicial sets and chain complexes (we refer to \[13\] for the mathematical definition of these structures).

3. Different algebraic categories are linked by functors, functors which should be also formally modeled. (The first two points were studied in \[10\] in a different context; this third one is new in this paper.) In this section, we will develop as an example the particular case of the EAT function constructing the chain complex canonically associated to a simplicial set.

Let us stress that we are not claiming that simplicial sets or chain complexes cannot be specified in other ways, for instance by using total algebras. When an algebraic structure is rich enough, the definition of a signature for it requires numerous design decisions (see in \[13\] the very different presentations which are possible for the simplicial categories, for example). But our goal is not to analyze different alternatives to specify some data structures. Our goal is to reflect, as
First of all, we generalize our results in order to deal with the aspects 1 and 2. The initial datum is a category, or rather a class of objects \( \mathcal{T} \), that we want to model (the class of groups, the class of simplicial sets and so on). Then a signature \( \Sigma \) is designed with the aim of obtaining \( \Sigma \)-algebras which represent (in some sense) the objects in \( \mathcal{T} \). Nevertheless it is not sensible to think of representing any object in \( \mathcal{T} \), both from the computability point of view (usually, \( \mathcal{T} \) is a non-numerable class) and from the practical one (it is very useful to identify the ground elements of some sort with a unique programming type); hence, we choose a data domain \( D \). Then, the signature \( \Sigma = (S, \Omega) \) is enriched with a boolean test operator \( eq_s : s \times s \to \text{bool} \) for each \( s \in S \), giving a new signature denoted by \( \Sigma^{eq} \). The idea now is that a \( \Sigma^{eq} \)-algebra \( A \) on \( D \), such that each \( A(eq_s) \) is an equivalence relation on \( D \), defines a \( \Sigma \)-algebra \( A/\sim \) on the quotient domain \( D/A(eq) \). Hopefully the quotient \( \Sigma \)-algebra \( A/\sim \) is (a representation of) an object of \( \mathcal{T} \). Finally, for each \( \Omega \)-operation \( \omega : s_1 \ldots s_n \to s \) we consider a set \( dom_\omega \subseteq D_{s_1} \times \ldots \times D_{s_n} \), the definition domain of \( \omega \), and we write \( dom = \{ dom_\omega \}_{\omega \in \Omega} \). The ambient category will be \( P\text{Alg}^{D,dom}(\Sigma^{eq}) \), that is to say, the category of partial \( \Sigma^{eq} \)-algebras on \( D \) with definition domain \( dom \) and endowed with weak morphisms of partial algebras \([12]\). Then the category of models \( \mathcal{C} \) will be a full subcategory of \( P\text{Alg}^{D,dom}(\Sigma^{eq}) \) such that its \( \Sigma^{eq} \)-algebras \( A \) satisfy that \( A(eq_s) \) are equivalence relations and \( A/\sim \) is an object of \( \mathcal{T} \).

In this extended context, we can repeat the construction \( \mathcal{C}_{imp} \) introduced in the previous section, obtaining for each signature \( \Sigma \) a category \( \mathcal{C}_{imp} \), subcategory of \( P\text{Alg}^{D,dom}(\Sigma^{eq}_{imp}) \), the category of partial hidden algebras on \( D \) (with the distinguished sort \( imp_\Sigma \) as the unique hidden sort), where \( \Sigma^{eq}_{imp} \) denotes \( (\Sigma^{eq})_{imp} \). For each algebra \( A \) in \( \mathcal{C}_{imp} \) the operation \( imp_\omega : imp_\Sigma s_1 \ldots s_n \to s \) will have as definition domain: \( A_{imp_\omega} \times dom_\omega \) (in other words, it will be total on the hidden argument). In addition, each \( A \) in \( \mathcal{C}_{imp} \) must represent a family of \( \mathcal{C} \)-objects (so, of \( \mathcal{T} \)-objects). Then, the category \( \mathcal{C}_{imp} \) has a final object similar to the one established in Section 3 (in particular, based on tuples of methods as in Cardelli’s approach \([11]\)).

In order to tackle the above mentioned point 3, let \( \mathcal{T}^1 \) and \( \mathcal{T}^2 \) be two categories and \( F : \mathcal{T}^1 \to \mathcal{T}^2 \) a functor, the mathematical construction to be modeled (for example, \( F \) could be the group-ring construction or the functor giving the chain complex associated with a simplicial set \([15]\)). Let us assume that the modeling process explained above can be accomplished for each \( \mathcal{T}^i \). So, we dispose of \( \Sigma^i, D^i, \text{dom}^i \) and \( \mathcal{C}^i \) subcategory of \( P\text{Alg}^{D^i,\text{dom}^i}(\Sigma^i, eq) \) for \( i = 1, 2 \). In addition, it is necessary to impose that for each \( \Sigma^1, eq \)-algebra \( A \) of \( \mathcal{C}^1 \), the \( \mathcal{T}^2 \)-object \( F(A/\sim) \) can be modeled as \( B/\sim \) for some \( \Sigma^2, eq \)-algebra \( B \) of \( \mathcal{C}^2 \) (in fact, this is usually a constraint on the definition of the data domain \( D^2 \); see examples later).

\[ \text{It is implicitly stated in this situation that the operations } A(\omega) \text{ respect the equalities } A(eq_s). \]
In other words, we are assuming that the functor $F : T^1 \to T^2$ can be translated into the representation level by means of a map $\bar{F} : \text{Obj}(C^1) \to \text{Obj}(C^2)$.

Let then $\Sigma$ be the union of $\Sigma_1^{\text{eq}}$ and $\Sigma_2^{\text{eq}}$, signatures constructed from $\Sigma_1^{\text{imp}}$ and $\Sigma_2^{\text{imp}}$, respectively, together with a new operation $f : \text{imp}_{\Sigma_1} \to \text{imp}_{\Sigma_2}$ between the distinguished sorts. In such a signature, there are two hidden sorts $\text{imp}_{\Sigma_1}$ and $\text{imp}_{\Sigma_2}$. This situation is new with respect to Section 3 and to our previous work [11] (but this kind of signatures is, of course, included in the general setting of hidden specifications [7]). Let us consider the hidden category $\text{PHA}l\text{g}^{D,\text{dom}}(\Sigma)$ with data domain $D := D^1 \cup D^2$ and definition domain $\text{dom} := \text{dom}^1 \cup \text{dom}^2$ (the operation $f$ is considered total). Then a subcategory $C$ of $\text{PHA}l\text{g}^{D,\text{dom}}(\Sigma)$ is defined as follows: an algebra belongs to $C$ if its restriction to $\Sigma_i^{\text{eq}}$ (its reduct on $\Sigma_i^{\text{eq}}$ following the algebraic specification terminology) is in $C_i^{\text{imp}}$, for $i = 1, 2$, and the interpretation of the operation $f$ is exactly that deduced from $\bar{F}$. More concretely, let $A$ be such a $\Sigma$-algebra and let us fix an element $I \in A_{\text{imp}_{\Sigma_1}}$. Then the element $A(f)(I) \in A_{\text{imp}_{\Sigma_2}}$ is (the representation of) the object of $C^2$ obtained from the application of $\bar{F}$ to the object of $C^1$ represented (or indexed) by $I$.

We can define a canonical algebra $A^{\text{can}}$ in $C$ from the union of the final objects in $C_i^{\text{imp}}$. To be precise, let us denote by $A_i^{\text{can}}$ the final object of $C_i^{\text{imp}}$ (i.e., the final object of $C_i^{\text{imp}}$ which consists of functional tuples defining, together with $D_i$, objects of $C_i$, as previously explained). Then, we define $A^{\text{can}}(f)$ mapping the tuple of functions representing a $C_1$-object $O$, into the tuple of functions which represents the $C_2$-object $\bar{F}(O)$. From this definition, the next result easily follows.

**Theorem 2.** The $\Sigma$-algebra $A^{\text{can}}$ is the final object in $C$. In addition, the reducts of $A^{\text{can}}$ on $\Sigma_1^{\text{eq}}$ and $\Sigma_2^{\text{eq}}$ correspond with the canonical final objects in $C_1^{\text{imp}}$ and $C_2^{\text{imp}}$, respectively.

In order to illustrate the power of these constructions, the section finishes with three increasingly complex (and interesting) examples: in the first one, we deal with the group-ring construction in a “rigid” setting (with total algebras and without quotient sets); in the second one, the same functor is considered with equalities different from the literal ones; the third example is devoted to the chain complex of a simplicial set in a way very close to the EAT way of working, proving that our techniques are powerful enough to model real functional symbolic computation systems.

**Example 1.** Let $T^1$ be $\text{GRP}$, the category of groups, let $T^2$ be $\text{RNG}$, the category of rings, and let $F : \text{GRP} \to \text{RNG}$ be the group-ring functor. As usual, $\Sigma^1 := \Sigma^\text{grp}$, the signature with a unique sort $\{g\}$ and three operations $\{\text{prd} : g \times g \to g, \text{inv} : g \to g, \text{unt} : \to g\}$. In this case, since equalities are the literal ones on the data domain, it is not necessary to construct the signature $\Sigma_1^{\text{eq}}$. In the same way, since all the operations are considered total, we do not need to define a definition domain $\text{dom}^1$. As data domain, it is enough to define $D_g := X$, a fixed set. The category $C^1 := C^{\text{grp}}$ is the subcategory of $\text{Alg}(\Sigma^\text{grp})$ such that the carrier sets are equal to $X$ and the algebras are objects in $T^1$. 


Similarly, the signature \( \Sigma^2 := \Sigma^{rng} \) has as sorts set \{a\} and as operations \{+ : a \times a \to a, - : a \to a, e : \to a, * : a \times a \to a\}. In order to be able to model the functor \( F \), we must define the data domain \( D_a := \mathbb{Z}[X] \), the free abelian group on the set \( X \). This decision implies that \( F \) can be modeled by means of \( \tilde{F} := F|_{C^{grp} : C^{grp} \to C^{rng}} \), where \( C^{rng} \) is the subcategory of \( Alg(\Sigma^{rng}) \) such that the carrier sets are equal to \( \mathbb{Z}[X] \) and the algebras are objects in \( T^2 \). Hence, our theorem 2 applies and we have a “universal” way of representing the group-ring construction (on a set \( X \)). For example, if \( X = \mathbb{Z} \), the set of integer numbers, we can represent any numerable group (endowed with its corresponding group-ring). Nevertheless, it is not possible, in this “rigid” context, to encode finite groups in the same way, keeping \( X = \mathbb{Z} \). The next example deals with this case.

**Example 2.** The construction and the first definitions are as in the previous example. We now consider the signature \( \Sigma^{grp,eq} \) with sorts \{g, bool\} and operations \{\( prd : g \times g \to g, inv : g \to g, un\) : \( g \to g, eqg : g \times g \to bool\)\} (we do not include an equality \( eq_{bool} : bool \times bool \to bool\), because the equality on \( bool \) will be the literal one). To define \( C^{grp} \) we choose \( \Sigma^{grp,eq}\)-algebras \( A \) from \( Alg(\Sigma^{grp,eq}) \) such that \( A_g = \mathbb{Z} \), \( A_{bool} = \{true, false\} \), \( A(eq_g) \) defines an equivalence relation on \( \mathbb{Z} \), and \( \sim \) is a group on \( \mathbb{Z}/A(eq_g) \). In this way, any finite or numerable group can be modeled on \( C^{grp} \) (and this covers any interesting group, from a computability or symbolic computation point of view). For the rings, we define \( \Sigma^{rng,eq} \) with sorts \{a, bool\} and operations as above but including a new operation \( eq_a : a \times a \to bool \). In order to define \( \tilde{F} \), we choose \( D_a := \mathbb{Z}[\mathbb{Z}] \) and given an object \( A \) in \( C^{grp} \) we consider an object \( B \) in \( C^{rng} \) such that \( B(eq_a) : \mathbb{Z}[\mathbb{Z}] \times \mathbb{Z}[\mathbb{Z}] \to \{true, false\} \) is the equivalence relation induced by \( A(eq_g) : \mathbb{Z} \times \mathbb{Z} \to \{true, false\} \) and we complete the definition of \( \tilde{F}(A) = B \) in the natural way. In this example, all the operations are total; in the following one, partiality must be considered.

**Example 3.** Let \( T^1 \) be \( SS \), the category of simplicial sets, let \( T^2 \) be \( CC \), the category of chain complexes, and let \( F : SS \to CC \) be the functor constructing the chain complex associated to a simplicial set. In the rest of the example, we use the terminology and constructions of the EAT system. Thus, we refer to [15] for further details and to [13] for the corresponding mathematical definitions.

Let \( \Sigma^{ss} \) be the signature with sorts \{nat, gsm, asm, bool\} and with operations \{\( dgn : nat \times nat \times asm \to asm, face : nat \times nat \times asm \to asm, eq_{gsm} : gsm \times gsm \to bool\)\}. The sort \( nat \) represents the natural numbers set \( \mathbb{N} \), \( bool \) the boolean values, \( gsm \) the geometrical simplexes and \( asm \) the abstract simplexes of the simplicial set. The operations \( dgn \) and \( face \) correspond to the degeneracy and face operators, respectively, in the simplicial set. This signature, being close to EAT, is placed somewhere between \( \Sigma^1 \) and \( \Sigma^{1,eq} \); only an equality test \( eq_{gsm} \) has been included. This is because in the data domain: \( D_{nat}^{ss} := \mathbb{N} \) and \( D_{bool}^{ss} := \{true, false\} \), with the literal equalities, \( D_{gsm}^{ss} := B \equiv \{B_p\}_{p \in \mathbb{N}} \) endowed in each particular simplicial set with an explicit equality, and \( D_{asm}^{ss} := \{< (j_k, \ldots, j_1), a > \mid a \in B_n, \text{ for some } n \in \mathbb{N}, k \in \mathbb{N}\} \).
$\mathbb{Z}$, $k \geq 0$, $j_i \in \mathbb{N}$, $\forall i = 1, \ldots, k$, $j_k < n + k$ and $j_i > j_{i-1}$, $\forall i = 2, \ldots, k$} with the equality induced by the equality on $D^{ss}_{gsm}$. The set $D^{ss}_{gsm}$ of geometrical simplexes is an indexed set to encode the concept of dimension (or degree) of a geometrical simplex. This induces a degree operation on the set of abstract simplexes: the degree of an abstract simplex $x = < (j_k, \ldots, j_1), a >$ is $\text{degree}(x) := k + n$, if $a \in B_n$. Then the operations of degeneracy and face must be partial because they depend on the degree of the abstract simplexes (see [13], for instance). This implies that a suitable definition domain $\text{dom}^{ss}$ must be introduced. So $C^{ss}$ is the subcategory of $PAlg^{D^{ss}, dom^{ss}}(\Sigma^{ss})$ whose objects $A$ satisfy the following properties: $A(eq_{gsm})$ is an equivalence relation on $D^{ss}_{gsm}$ and then the quotient algebra $A/\sim$ (where in $D^{ss}_{asm}$ the equality induced by $A(eq_{gsm})$ is considered) is a simplicial set, an object of $SS$.

For chain complexes, the signature $\Sigma^{cc}$ consists of the sorts \{int, gnr, cmb, bool\} and the operations \{add-cmb-to-cmb : cmb $\times$ cmb $\rightarrow$ cmb, zero-cmb : int $\rightarrow$ cmb, mns-cmb : cmb $\rightarrow$ cmb, mlt-n-cmb : int $\times$ cmb $\rightarrow$ cmb, d : cmb $\rightarrow$ cmb, eq_{gnr} : gnr $\times$ gnr $\rightarrow$ bool\} with the following interpretation: int represents the integer numbers set $\mathbb{Z}$, bool the boolean values, gnr a (graded) set of generators and cmb the elements, usually called combinations, of the (graded) free abelian group on the set denoted by gnr. Then the four first operations explain the $\mathbb{Z}$-module structure on cmb, d is intended to represent the differential operators (see [13]) and $eq_{gnr}$ will define an equivalence relation on the generators set (inducing another equivalence relation on the corresponding combinations set). In order to draw on $\Sigma^{cc}$ the chain complexes coming from the simplicial sets in $C^{ss}$, the data domain is designed as follows: $D^{cc}_{int} := \mathbb{Z}$, $D^{cc}_{bool} := \{\text{true, false}\}$, $D^{cc}_{gnr} := B \equiv \{B_p\}_{p \in \mathbb{Z}}$ with $B_p = \emptyset$ if $p < 0$ (in other words, $D^{cc}_{gnr}$ is defined from $D^{ss}_{gsm}$, in the data domain of $C^{ss}$), and $D^{cc}_{cmb} := \{< p, [(t_1, a_1), (t_2, a_2), \ldots, (t_m, a_m)] > | p \in \mathbb{Z}$, $m \in \mathbb{Z}$, $m \geq 0$, $t_i \in \mathbb{Z}$, $t_i \neq 0$ and $a_i \in B_p$, $\forall i = 1, \ldots, m\}$ (this is an explicit representation of the elements of the free abelian group on $D^{cc}_{gnr}$, being $p$ the degree of the combination). To finish, it is enough to define $\text{dom}^{cc}$: all the operations are total, except add-cmb-to-cmb and $eq_{gnr}$ which only work on arguments of the same degree (for the combinations and the generators, respectively). With these definitions it is very easy to determine the category $C^{cc}$ and the map $\tilde{F} : Obj(C^{ss}) \rightarrow Obj(C^{cc})$ translating the chain complex functor $F : SS \rightarrow CC$.

Thus, theorem 2 applies and we obtain a final object which perfectly matches the functional data structures used in EAT for encoding the chain complex associated to a simplicial set.

5 Conclusions and Future Work

In order to analyze a symbolic computation system essentially based on functional programming, several formalisms designed rather for object-oriented systems have been explored (namely, hidden specifications, coalgebras and Cardelli’s metaphor of “objects as records of methods”). A first conclusion of our results is that the functional programming case (objects without local state), being easier,
allows a more manageable work with the formalisms for the analyst, in particular
with the different final objects in the literature (see [5, 18]). Another conclusion
is that the direct relation between these particular, functional, final objects and
the EAT data structures shows the “more general”, “universal” nature of the
EAT system, in the field of Algebraic Topology.

In the paper, some of the results have been presented in an institutional
framework. More research will be necessary to elucidate if the new results related
to functorial constructions can be also expressed by means of institutions. It
should also be explored the relationship with the work [4], where the inheritance
between data structures in the Sergeraert’s Kenzo system [16] (successor of EAT)
is studied.

Finally, some investigation should be undertaken in order to compare our
approach with others coming from the type system community (in this paper,
only a little example described in Haskell has shown that this comparison is
feasible and, in fact, quite interesting).

References

2. R. Burstall, R. Diaconescu, Hiding and behaviour: an institutional approach, in A
3. C. Cîrstea, Coalgebra semantics for hidden algebra: parameterised objects and in-
4. C. Domínguez, J. Rubio, Modeling inheritance as coercion in a symbolic computation
5. J. Goguen, R. Burstall, Institutions: Abstract model theory for specification and
programming, Journal of the Association for Computing Machinery 39(1) (1992)
95-146.
6. J. Goguen, R. Diaconescu, Towards an algebraic semantics for the object paradigm,
Lecture Notes in Computer Science 785 (1994) 1-29.
55-101.
Haskell: a non-strict, purely functional language, Version 1.2., SIGPLAN Notices
9. B. Jacobs, J. Rutten, A tutorial on (co)algebras and (co)induction, EATCS Bulletin
10. L. Lambán, V. Pascual, J. Rubio, Specifying implementations, in Proceedings IS-
11. L. Lambán, V. Pascual, J. Rubio, An object-oriented interpretation of the EAT
system. Preprint.
Teubner, 1996.
14. J. Rubio, F. Sergeraert, Locally effective objects and Algebraic Topology, in Com-
15. J. Rubio, F. Sergeraert, Y. Siret, EAT: Symbolic Software for Effective Homology
Computation, ftp://fourier.ujf-grenoble.fr/pub/EAT, Institut Fourier, Greno-
ble, 1997.


Distributed Execution of Functional Programs Using the JVM

André Rauber Du Bois\textsuperscript{1} and Antônio Carlos da Rocha Costa\textsuperscript{2}

\textsuperscript{1} UFRGS-Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação (PPGC)
Caixa Postal 15064, 91501-900, Porto Alegre, RS, Brazil
dubois@inf.ufrgs.br

\textsuperscript{2} UCPel-Universidade Católica de Pelotas
Escola de Informática
Rua Félix da Cunha 412, 96010-000, Pelotas, RS, Brazil
rocha@atlas.ucpel.tche.br

Abstract. We present in this paper the implementation, in the Java language, of a distributed environment for running functional programs. The idea is to join Java Virtual Machines (JVMs) running on different computers into a single virtual machine for running functional programs. To test this virtual machine we have implemented a small Haskell like functional language in which parallelism is expressed by some simple combinators.

1 Introduction

The “Write once, run anywhere” philosophy of the Java Language had attracted many developers of sequential declarative languages (some examples are \cite{3,6,7,8,16}). Besides portability, another important feature of the JVM for developers of declarative languages is its built-in garbage collector.

The Java language is well known for its facilities for distributed programming. It contains a large number of standard classes that implement some high level abstractions for network programming such as sockets, streams, object serialization and remote method invocation.

In this paper we explore the distributed programming facilities of the Java language to implement a parallel/distributed functional language. The idea is to join Java Virtual Machines running on different computers into a single distributed virtual machine for running functional programs.

First we present how we have implemented a functional language on top of the JVM. In the second part of the paper we explain how parallelism is introduced in this functional language and how the distributed environment was implemented.

2 Running Functional Programs on the JVM

We have implemented a small functional programming language, the Fun Language \cite{3}, by compiling it to Java. It is a powerful but simple Haskell like func-
tional language based on Peyton Jones’ Core language [10]. Although simple, it can be used as target code for more powerful ones.

Functional languages are usually implemented using graph reduction, by compiling them to a graph reduction machine language. Thus, our first step towards running functional programs on the JVM was to implement a graph reduction machine (based on the classic G-Machine [1]), in Java. With a graph reduction machine implemented on top of the JVM, we can easily translate any pure functional program into Java byte-code.

The G-Machine is a fast graph reduction machine based on the compilation of expressions. The main idea is to translate each function in the functional program into a sequence of instructions which, when executed, will construct an instance of the function body [10]. As the G-Machine represents expressions as graphs, its instructions allows to construct the graph representing function application and each expression in the functional program is used as a reduction rule for the graph.

The G-Machine we have implemented is simply a Java class, the \texttt{GM} class (figure 1), which has some static methods that are the G-Machine instructions. To run functional programs on the JVM we compile each function in the functional program to one Java Class. These classes call the G-Machine instructions (by calling the static methods of the \texttt{GM} class) necessary to instantiate the function in the G-Machine. For example, the function

\[
fx = \text{id} \times \quad
\]

will be compiled to the following Java class:

\begin{verbatim}
class f extends Nsuperc{
    f (){    
        narg = 1;
        name = new String ("f");}

    public void code(){
        GM.push (0);
        GM.pushglobal (new id());
        GM.mkap();
        GM.update(1);
        GM.pop(1);
    }
}
\end{verbatim}

The \texttt{Nsuperc} is an abstract class that encompasses all functions defined in the functional program being compiled. The constructor of the class (in this case \texttt{f()}) always has to define two values: \texttt{narg}, which is the number of arguments for the function and the string \texttt{name}, which gives the name of the function (for debugging reasons).
All classes implementing the Nsuperclass must have a method called `code()` which has in its body the G-Machine instructions necessary to instantiate the body of the function.

The `GM.push(0)` instruction pushes onto the stack the argument of the function. Next, `GM.pushglobal (new id())`, pushes a pointer to the pre-defined function id onto the stack. The `GM.mkap()` instruction takes the first two pointers from the stack leaving a pointer to an application node from them. The `GM.update(1)` instruction overwrites the root of the original redex with an indirection node pointing to the newly constructed instance.

Our approach of compiling each function in the functional program to one Java class gives us a simple interface that makes it easier to reuse the G-Machine implemented. Any programmer who knows how the G-Machine works can use our implementation of the G-Machine as a back-end for the implementation of some other lazy functional language. Having one Java class for each function we can also take advantage of the load on demand strategy of the JVM for loading classes during the execution of programs. This design option will also be important for the distributed implementation of the runtime system as will be shown in the next section.

More about the G-Machine and the behaviour of its instructions can be found in [9,10,1].

```java
class GM {

    private static Gstack stack = new Gstack();

    static void push(int n) {
        (...) }

    static void pop(int n) {
        (...) }

    static void mkap() {
        (...) }

    static void eval() {
        (...) }

    (...) 
}
```

Fig. 1. The GM class
3 Distributed Execution of Functional Programs on the JVM

This section presents the implementation of a distributed G-Machine in Java. First we show how parallelism is expressed in the Fun Language and then we discuss the implementation issues.

3.1 Parallel Combinators

Parallelism is introduced in the Fun language by the par combinator. It works like the par operator of the Glasgow Parallel Haskell Language (GpH) [12]. The expression

\[ \text{par } p \ e \]

has the same value as \( e \). Its dynamic behaviour is to indicate that \( p \) could be evaluated in another machine or PE (Processing Element).

The par combinator is implemented using a new G-Machine instruction called GM.par. This instruction simply puts the node on top of the stack into a task pool (a pool of nodes that can be evaluated in parallel with the main program) and leaves on top of the stack an indirection node pointing to the new position of the node in this task pool. The GM.par instruction will be explained in details later.

The par combinator can easily be used to implement parallel functional programs. A simple example is the parallel implementation of the fib function:

\[
\begin{align*}
\text{pfib } n & = \text{let fib1 = pfib } (n-1); \text{ fib2 = pfib } (n-2) \\
\text{ in } \text{if } (n<1) 1 \text{ (seq (par fib2 fib1) (fib1 + fib2 +1))};
\end{align*}
\]

Here we used the seq combinator that forces the evaluation of its first argument to weak head normal form (WHNF) and returns its second argument. Parallel Functional programming using par and seq is explained in [12].

3.2 How Does the Distributed Execution Works?

The main idea of Fun’s distributed runtime system is to combine JVMs running on different machines, in a network of workstations, into a single distributed virtual machine for running parallel functional programs. This virtual machine is called Distributed Functional Environment (see figure 2).

In the Distributed Functional Environment there is a main PE that starts the evaluation and coordinates the distribution of tasks. This PE runs the Main G-Machine. The first step to start the distributed evaluation of a functional program (after its compilation) is to initialize the distributed functional environment. We start the main G-Machine and it will wait for connections of other
JVMs running the client G-Machine. Once all clients have connected to the main PE, it can start the evaluation of the program.

Each time the main G-Machine encounters a par combinator it sends one part of the graph being evaluated to a task pool and leaves in the graph a node (we call it Npool node) that points to an address in the task pool where the sub-graph is now located.

When a node is sent to the task pool, the runtime system checks if there is any idle client; if there is one, it sends the nodes in the task pool to the client G-Machines using a FIFO scheduling. This approach of sending nodes only when there are idle clients is called lazy task creation [5].

The client G-Machine receives the node, evaluates it to WHNF and sends it back to its original address in the task pool.

When the main G-Machine encounters an indirection node (Npool node) pointing to a task pool address three things may happen:

- If the node in the task pool is in WHNF (this means that some client had already evaluated it), it grabs the node and continues to evaluate the program;
- If the node is not in WHNF and is not being evaluated by any other client, it grabs the node and continues to evaluate the program;
- If the node is being evaluated by some client, the main G-machine saves its state, grabs any other available node in the task pool and begins its evaluation. When the evaluation is finished, it returns the result to its location in the task pool and checks if the first node is in WHNF. In this case it restores its old state and continues the main evaluation. If not, it repeats the process.

3.3 Implementing the Parallel Combinator

As explained before, the par combinator is implemented using a new G-Machine instruction, the GM.par instruction. This instruction puts the node on top of the
public class par extends NsuperC{

    public par(){
        nargs = 2;
        name= new String ("par");
    }

    public void code() {
        GM.push(0);
        GM.par();
        GM.update(0);
        GM.push(1);
        GM.update(2);
        GM.pop(2);
    }
}

Fig. 3. Implementation of the par combinator

stack into a task pool and leaves on the stack an indirection node pointing to
the address in the task pool where this node is now located. The Java code for
this instruction is:

public static void par()
{
    Node node = GM.pop(1);
    int add = pool.putnode(node);
    GM.push (new Npool(add));
}

It first pops out of the stack its first element and sends it to the task pool
using the putnode method of the pool. This method returns the address where
the node is now located. Finally the GM.par instruction pushes onto the stack
an Npool node, which is a G-Machine’s graph node that contains only the task
pool address. The task pool is implemented as a linked list of G-Machine’s nodes
with some address control. When the main G-Machine, during the evaluation of
the program, encounters an Npool, it will search for the real node in the task
pool using the address in the Npool object.

As can be seen in figure 3, it is straightforward to implement the par combi-
nator using the GM.par instruction combined with the other G-Machine instruc-
tions.

The par implementation works as follows: first the GM.push(0) instruction
pushes onto the stack the first argument of the par combinator which is the
argument that we want to send to the task pool. Then, the next instruction to
be executed is GM.par, which sends this node to the task pool and leaves on top
of the stack the \texttt{Npool} object which contains the address of the node in the pool. Next, the \texttt{GM.update} instruction updates the first argument so it now points to the task pool address. In this way, when the G-Machine, while evaluating the program, encounters this \texttt{Npool} node, it will look for it in the specified address in the task pool. Finally we update the root of the redex with the second argument of \texttt{par} (instructions \texttt{GM.push(1)} and \texttt{GM.update(2)}) which is the result of the evaluation of the combinator.

egin{lstlisting}[language=Java]
public class server implements Runnable {
    public final static int port = 1515;
    ServerSocket ss;

    public void run(){
        try{
            ss = new ServerSocket(port);
            while (true)
            {
                connect fs = new connect (ss.accept());
                fs.start();
            }
        }catch (IOException e)
        {
            (...) 
        }
    }
}

Fig. 4. The server class

3.4 Starting the Distributed Functional Environment

To start the Distributed Functional Environment we first run the Main G-Machine. When it starts running, the program is divided in two threads, the Main thread (that contains the sequential G-Machine explained before) and the server thread that deals with the connections of the clients.

The server thread (figure 4) creates a socket object in a port and keeps waiting for the clients’ connection. When a client connects to that port, it creates a connect thread for that client. This thread deals with sending and receiving G-Machine’s nodes and with accessing the task pool.

Once all clients are connected to the main G-Machine, it can start the evaluation of the program. Initially, all connect threads are with a low priority. When the task pool starts receiving nodes, those threads receive a higher priority and start sending nodes to clients.
3.5 Implementing the Distributed Execution

To send nodes from one G-Machine to other G-machines located on different JVMs, we use some important features of the Java Language such as sockets, object streams and serialization.

The Socket class performs networking on top of a stream-based network connection. To connect two machines we just have to create the client and the server sockets and connect them using a stream. Anything that you write on the server side can be read on the client side. Java has a special kind of stream that is an object stream. If you connect a client and a server socket using an object stream, sending objects over the network is straightforward. All objects that are written to an object stream must implement the serializable interface. One of the most important features of the Java.io package is the ability to serialize objects: to convert an object into a stream of bytes that can later be deserialized back into a copy of the original object.

The Node class is the superclass for all G-Machine’s nodes. Thus, it must implement the serializable interface, so the main G-Machine can send its nodes to clients using an Object stream:

abstract public class Node implements Serializable { }

All other types of nodes inherit the serialization capability from the Node class (as can be seen figure 5).

![Fig. 5. G-Machine’s Nodes](image)

When the runtime system sends a node to a client machine, this node must be packed together with its task pool address using the packet class (figure 6). It’s important to notice that the packet class must also implement the Serializable interface and that all Java’s basic types are already serializable.
public class packet implements Serializable{

    public Node value;
    public int add;

    public packet (Node value, int add)
    {
        this.value = value;
        this.add = add;
    }
}

Fig. 6. The packet class

When the task pool is not empty, the connect threads will send nodes to the clients using the Socket created in the server thread. The connect thread takes one node from the task pool and sends it in a packet to its respective client using an object stream:

    (...)
    packet pack = GM.pool.fgetnode();
    streamout.writeObject(pack);
    streamout.flush();
    packet resp = (packet) streamin.readObject();
    (...)

The fgetnode method of the pool returns a non-evaluated node in a packet. The connect thread sends the node to its client using the writeObject method of the stream and reads the response using the readObject method. This thread will stay blocked waiting for the client’s response.

The client machine receives this packet, extracts the node, evaluates it to WHNF and sends it back to the main G-Machine together with its task pool address:

    (...)
    packet clientpacket = (packet) strin.readObject();
    Node resp = (Node) GM.reduce (clientpacket.node);
    strout.writeObject (new packet (resp, clientpacket.add));
    (...)

The GM.reduce is the main method of the client G-Machine. It receives the node and evaluates to WHNF:

    public static Node reduce (Node main)
    {
        GM.push_onto_stack(main);
GM.eval();
return (GM.first_stack_elem());
}

The GM.eval() method is the G-Machine’s instruction that evaluates the element on top of the stack to WHNF. This instruction is described in [9,10]. The reduce method returns the evaluated node to the client and it sends it back to the main G-Machine.

When the main G-Machine receives the node back, it unpacks it and puts it back into its task pool address.

4 Benchmarks

As in other implementations of sequential declarative languages (for example [6,14,7,8]) on the JVM, our language is not very fast, but with the distributed evaluation of the programs we have achieved a good speedup in some programs, as can be seen in table 1, using Linux/Pentium200MMX machines connected with a 10Mbps Ethernet network.

<table>
<thead>
<tr>
<th>Programs</th>
<th>1 Proc. t(s)</th>
<th>1 Proc. Speedup</th>
<th>2 Proc. t(s)</th>
<th>2 Proc. Speedup</th>
<th>3 Proc. t(s)</th>
<th>3 Proc. Speedup</th>
<th>4 Proc. t(s)</th>
<th>4 Proc. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfib 25</td>
<td>55s</td>
<td>1.0</td>
<td>34s</td>
<td>1.6</td>
<td>20s</td>
<td>2.75</td>
<td>20s</td>
<td>2.75</td>
</tr>
<tr>
<td>pcoins</td>
<td>28s</td>
<td>1.0</td>
<td>20s</td>
<td>1.4</td>
<td>17s</td>
<td>1.64</td>
<td>16s</td>
<td>1.75</td>
</tr>
<tr>
<td>peuler 200</td>
<td>1m10s</td>
<td>1.0</td>
<td>48s</td>
<td>1.45</td>
<td>33s</td>
<td>2.12</td>
<td>24s</td>
<td>2.91</td>
</tr>
<tr>
<td>minmax</td>
<td>20s</td>
<td>1.0</td>
<td>11s</td>
<td>1.81</td>
<td>9s</td>
<td>2.22</td>
<td>7s</td>
<td>2.85</td>
</tr>
<tr>
<td>ptak</td>
<td>42s</td>
<td>1.0</td>
<td>19s</td>
<td>2.21</td>
<td>19s</td>
<td>2.21</td>
<td>19s</td>
<td>2.21</td>
</tr>
<tr>
<td>listOffibs</td>
<td>34s</td>
<td>1.0</td>
<td>20s</td>
<td>1.7</td>
<td>20s</td>
<td>1.7</td>
<td>20s</td>
<td>1.7</td>
</tr>
</tbody>
</table>

We used the System.currentTimeMillis() method to measure the run time and we did not count the time wasted to start the parallel machine.

4.1 Discussion

Although we have achieved a good speedup compared to running the programs in a single machine, we can see in table 1 that some programs that should run faster with more machines, do not modify their execution time with more than three machines.

This happens because we have not implemented yet any load balance algorithm. Currently, when a par combinator is called in the client machine it is ignored. The par combinator than behaves as a function that returns as result its second argument. Thus, sometimes we have the situation in which the task
pool is empty and there are also some idle clients, while a working client, could be generating work. The problem is that usually a client machine would generate a large number of small tasks and it would cost too much to send those tasks to the task pool that is located in the main machine. As it is difficult to know if it is worth sending a task created in a client to the task pool in the main machine, we have decided to ignore all tasks generated by clients.

Another source of overhead in the distributed execution of functional programs in our system is shared graphs. For example:

\[
\text{let } x = 4*5 \text{ in } \\
\text{par (f x) (g x);} \\
\]

In this case if \(x\) is sent to the client machine before its evaluation in the main machine, it will be evaluated twice. The problem is that the implementation and management, using the JVM, of a distributed heap would be too expensive.

It is important to notice that, as said before, an implementation of a functional language using Java and its virtual machine, not a JIT (just in time) compiler, is very slow. Thus we can find parallelism where a fast implementation of a parallel functional programming language would not.

5 Related Works

Many compilers for lazy functional languages to Java byte-code were already developed. For example, in Wakeling’s work [14,15,16], he implements Haskell compilers to Java byte-code, one based on the G-Machine [14] (using an approach similar to ours) and another based on the \((\nu, G)\)-Machine [15]. He shows that his implementation using the JVM is very slow but he achieved some good results using a JIT compiler [16]. Erik Meijer [8], in his work with the functional scripting language Mondrian, implements a compiler to Java using a class to specify each function definition, but uses a different evaluation method. Meehan and Joy [7] implemented a compiler for their functional Language Ginger based on the G-Machine, but they avoid generating one Java class for each function. There is also a paper that describes an implementation in Java of a small functional language based using the Spineless Tagless G-Machine [2]. Finally, there is the work on the multi-paradigm Curry language [3]. In that paper, Hanus presents an abstract machine and its implementation in the Java language, for the functional / logic language Curry.

We did not find, as far as we could investigate, any work that uses the distributed programming facilities of the Java language to implement distributed execution of functional programs.

Our model of distributed execution and parallel graph rewriting is based on the work on the GpH language [13,12] and the models presented in the books of Peyton Jones [9,10].
6 Conclusions

We have presented our first experiences on the implementation of Parallel / Distributed functional languages. The main idea of this paper was to show how we used the distributed and network features of the Java language to implement a distributed abstract machine for running functional programs.

This paper doesn’t intend to be a final discussion on the subject but just to present some experiments.

We are now working on some modifications to the model in order to have a better load balancing between clients. We plan to add a task pool for each client and then to use a message passing algorithm to send tasks to the main task pool only when they are necessary.

The source code of the Fun language and the programs used for the benchmarks can be downloaded from the web page of the project: 

References

2. Choi, K., Lim, H., Han, T. Compiling Lazy Functional Programs Based on the Spineless Tagless G-machine for the Java Virtual Machine. In Fifth International Symposium on Functional and Logic Programming FLOPS, Waseda University, Tokyo, Japan March (2001)


Fusion in Coq

José L. Freire Nistal¹, José E. Freire Brañas², Antonio Blanco Ferro¹, and Juan J. Sánchez Penas¹*

¹ LFCIA, Dpt. of Computer Science
University of A Coruña
Facultad de Informática
Campus de Elviña
15071 A Coruña
Spain
{freire,blanco,juanjo}@dc.fi.udc.es

² IES Carballo Calero
Avda. Castelao, 15406 Ferrol
Spain
rike@inicia.es

Abstract. Fusion theorem is a classical result that allows the simplification of the morphisms among homogeneous structures [10]. We present this theorem and some generalizations in the context of the constructive proof assistant tool Coq [2] where we have dependent types and parametric polymorphism.

The work is organised as follows: after the classical interpretation of the fusion law for catamorphisms in a categoric context, examples of fusion for programs defined with recursive types in Coq are analysed and the theorems of corresponding optimisation are shown. Finally, a generalisation of fusion law for inductive types is presented which is applied to a specific case.

1 Introduction

The logical framework Coq is an implementation of the Calculus of Inductive Constructions (CIC) of G. Huet, T. Coquand and C. Paulin–Mohring, performed at INRIA [2], [11]. This system is a theorem prover goal-directed and tactic-driven in the style of LCF [7]. Types can be defined inductively and procedures on them can be obtained automatically. This allows programming directly from the specifications via program constructors associated to the inductive type [8], an approach to generic programming philosophy. Also, Coq provides an extraction mechanism of programs from proofs.

Our aim is to extend the law of fusion which is valid for catamorphisms to more general programs constructed in Coq with the recursive schemes of the inductive types. It involves, therefore, a process of extension of laws of non-dependent elimination schemes to their dependent homonyms. [12].

* This work was supported by EC-CICYT Research Project 1FD97-1759 and XUGA PGIDT99COM1052
CIC is a type theory that results from the combination of the intuitionistic
type theory of types of Martin-Löf and the \( \lambda \)-polymorphic calculation of Girard’s \( F_\omega \).
The theorems to prove are represented as types, and their proofs are terms with
these types. This uses the well known Curry-Howard isomorphism \cite{9} based on
the ”formulas as types” paradigm; this correspondence has been the principal
tool for the correct interpretation of the relation between the intuitionistic
logic and the typed \( \lambda \) calculus. The basic idea consists of thinking that logical formulas
can be interpreted as types in an adequate type theory; thus, the formula to
prove is associated to a corresponding \( \lambda \)-type term and the reduction of a proof
by cut-elimination corresponds to a normalisation of the associated \( \lambda \)-term. So,
if a formula is derivable in a particular logical system, the corresponding type
contains terms in the associated type theory.

There are two basic types in Coq: Set for the definition of objects, and Prop
for the declaration of properties and relations with regard to these types. Simi-
larly, the establishment of the truth of one proposition consists in the construc-
tion of a term which inhabits that proposition as a type. That is the only way
to prove things: truth is inhabitation.

The notation \( a:A \) (\( a \) is of type \( A \)) is interpreted as ”\( a \) is a proof of \( A \)”, when
\( A \) is of type Prop, or ”\( a \) is an element of \( A \)”, when \( A \) is a Set (constructive view).

The constructions permitted are: \( x \mid (M \ N) \mid [x:T]M \mid (x:T)P \), where \( x \) denotes
variables as well as constants; \( (M \ N) \) is the application; the third expresion repre-
sents the program (\( \lambda \)-expression) of parameter \( x \) and body \( M \) (the abstraction
of variable \( x \) of type \( T \) in \( M \)). Lastly, the fourth is the program type that admits
an entry of type \( T \) and returns a result of type \( P \). This type is referred to as
product type and, in type theory, is represented as \( \prod_{x:T} P \) or also as \( \forall x:T. P \).
If \( x \) is not free in \( P \) then, this is simply written \( T \rightarrow P \), the type of the functions
between these two types, or non-dependent product.

2 Categorical Interpretacion of Inductive Types

Given a category \( A \), a functor \( F : A \rightarrow A \) and an object \( A \in A \), a \( F \)-algebra on
\( A \) is a pair \((A, \xi)\), where \( \xi : F(A) \rightarrow A \) is a morphism in \( A \).

If \((A, \xi)\) and \((B, \theta)\) are \( F \)-algebras, an \( F \)-algebras homomorphism is a mor-
phism \( h : A \rightarrow B \) in \( A \) so that the following diagram commutes:

\[
\begin{array}{ccc}
F(A) & \xrightarrow{F(h)} & F(B) \\
\downarrow{\xi} & & \downarrow{\theta} \\
A & \xrightarrow{h} & B
\end{array}
\]

If \( F \) is a cocontinuous functor, the category of \( F \)-algebras has an initial object
\((\mu F, \text{in}_F)\). Then \( \text{in}_F \) is an isomorphism (\( \mu F \) is a fixed point of \( F \)) and for any
other \( F \)-algebra \((A, \xi)\), there exists a unique \( F \)-algebras homomorphism called
catamorphism \((\text{cata } \xi) : (\mu F, \text{in}_F) \to (A, \xi)\). This is the unique homomorphism making the following diagram commutative:

\[
\begin{array}{ccc}
F(\mu F) & \xrightarrow{F(\text{cata } \xi)} & F(A) \\
\downarrow \text{in}_F & & \downarrow \xi \\
\mu F & \xrightarrow{(\text{cata } \xi)} & A
\end{array}
\]

The inductive type corresponds to the initial algebra, \((\mu F, \text{in}_F)\), of a cocontinuous functor \(F\), where \(\text{in}_F\) encloses the type constructors. Dually, the coinductive type is the final coalgebra of a continuous functor.

If \((B, \theta)\) is another \(F\)-algebra, then, the universality of \((\mu F, \text{in}_F)\) establishes that for all \(h : (A, \xi) \to (B, \theta)\) we have the following commutative diagram:

\[
\begin{array}{ccc}
F(B) & \xrightarrow{F(h)} & F(h) \\
F(\mu F) & \xrightarrow{F(\text{cata } \xi)} & F(A) \\
\downarrow \text{in}_F & & \downarrow \xi \\
\mu F & \xrightarrow{(\text{cata } \xi)} & A \\
& \downarrow h & \\
& B
\end{array}
\]

and, in particular

\((\text{cata } \xi); h = (\text{cata } \theta)\)

which expresses the well known fusion law.

### 2.1 The Type \texttt{nat}

Given the functor \(F(X) = 1 + X\), the algebra \((\text{nat}, [O, S] : 1 + \text{nat} \to \text{nat})\) defines the inductive type of the naturals. Here, 1 denotes the terminal object (one element set), and \([O, S]\) is the function defined by the constructors \(O : 1 \to \text{nat}\) and \(S : \text{nat} \to \text{nat}\).

For any other \(F\)-algebra \((C, [c : C, t : C \to C])\), the homomorphism \((\text{cata } (c, t))\) is the only one which makes the following diagram commutative:
In Coq:

```
Inductive nat : Set :=  O : nat | S : nat->nat
```

the system provides the following schemes:

```
nat_ind:(P:(nat->Prop))(P O)->((n:nat)(P n)->(P (S n)))->
(n:nat)(P n)
nat_rec:(P:(nat->Set))(P O)->((n:nat)(P n)->(P (S n)))->
(n:nat)(P n)
```

and the recursive scheme verifies:

```
(nat_rec P init step 0)=init.
(nat_rec P init step (S n))=(step n (nat_rec P init step n)).
```

The catamorphism \( (\text{cata} (c,t)) : (C:\text{Set})C->(C->C)->\text{nat}->C \) is defined by:

```
Definition cata_nat:=[C:\text{Set};c:C;t:C->C](nat_rec[_:nat]Cc[_:nat]t).
```

verifying the two conditions:

```
(cata_nat C c t O )=c
(cata_nat C c t (S n)) = (t (cata_nat C c t n))
```

Addition of an \( n:\text{nat} \) is a catamorphism:

```
Definition sum_n := (cata_nat nat->nat [n:nat]n
[f:nat->nat;n:nat](S (f n))).
```

also, the Ackermann function:

```
Definition iter:=(cata_nat((nat->nat)->nat)([f:nat->nat](f (S O)))
([h:(nat->nat)->nat][k:nat->nat](k (h k)))).
Definition iterate:=([f:nat->nat][n:nat](iter n f).
Definition Ackermann:=(cata_nat nat->nat S iterate).
```

But, even the primitive recursive factorial function

```
Definition factorial := (nat_rec [_:nat]nat (S O)
[n:nat][m:nat](mult (S n) m)).
```

with type \( \text{nat} \) to \( \text{nat} \), can not be defined as a catamorphism.
2.2 The Type list

Let us now consider the functor \( F_A(X) = 1 + A \times X \). The algebra

\[
((\text{list } A), [(\text{Nil } A), (\text{cons } A)] : 1 + A \times (\text{list } A) \to (\text{list } A)),
\]
defines the inductive type of type \( A \) lists.

If \((B, [b : B, f : A \times B \to B])\) is another \( F_A \)-algebra, the homomorphism \((\text{cata } (b, f)) : (\text{list } A) \to B\) is the only one which makes the following diagram commutative:

\[
\begin{array}{ccc}
1 + A \times (\text{list } A) & \xrightarrow{1 + A \times (\text{cata } (b, f))} & 1 + A \times B \\
((\text{Nil } A), (\text{cons } A)) & \downarrow & \downarrow [b : B, f : A \times B \to B] \\
(\text{list } A) & \xrightarrow{(\text{cata } (b, f))} & B
\end{array}
\]

In Coq:

```
Inductive list [A:Set]:Set := Nil : (list A)
  | Cons : A -> (list A) -> (list A).
```

the system provides the following schemes:

```
list_ind : (A:Set; P:((list A)->Prop)) (P (Nil A)) ->((y:A; l:(list A))(P l)->(P (Cons A y l)))
  ->(l:(list A))(P l)
list_rec : (A:Set; P:((list A)->Set)) (P (Nil A)) ->((y:A; l:(list A))(P l)->(P (Cons A y l)))
  ->(l:(list A))(P l)
```

and the recursive scheme verifies:

\[
\begin{align*}
(\text{list_rec } A \ P \ \text{vnil} \ \text{vstep} \ (\text{Nil } A)) &= \text{vnil.} \\
(\text{list_rec } A \ P \ \text{vnil} \ \text{vstep} \ (\text{Cons } A \ y \ l)) &= \\
& (\text{vstep } y \ l \ (\text{list_rec } A \ P \ \text{vnil} \ \text{vstep } l)).
\end{align*}
\]

The catamorphism \((\text{cata } (b, f)) : (A,B:Set)B\to(A*B\to B)\to(\text{list } A)\to B\)

```
Definition cata_list :=[A,B:Set;b:B;f:A*B->B]
(list_rec A [_.:(list A)] B b [a:A][_.:(list A)][x:B](f(a,x))).
```

verifies the reductions

\[
\begin{align*}
(\text{cata_list } A \ B \ b \ f \ (\text{Nil } A)) &= b \\
(\text{cata_list } A \ B \ b \ f \ (\text{cons } A \ (a,l))) &= (f \ (a, \ (\text{cata_list } A \ B \ b \ f \ l))).
\end{align*}
\]

where \(\text{cons}\) is the uncurryfied version of \(\text{Cons}\)\(^1\)

\(^1\) Definition curry :=[A,B,C:Set; f:A*B->C; a:A; b:B](f (a,b)).

Definition uncurry :=[A,B,C:Set; f:(A->B->C); l:(A*B)](f (Fst l) (Snd l)).

Definition cons :=[A:Set](uncurry A (list A) (list A) (Cons A)).
This catamorphism \texttt{cata_list} is usually referenced as \texttt{foldr}. As an example we have the catamorphism \texttt{maplist:(A,B:Set)(A->B)->(list A)->(list B)} showing the functorial character of the constructor \texttt{list}:

\begin{verbatim}
Definition maplist:=[A,B:Set;f:A->B](cata_list A (list B) (Nil B) [ym:A*(list B)]let (y,m)=ym in (Cons B (f y) m)).
\end{verbatim}

Another useful catamorphism
\begin{verbatim}
Ext_list:(A,B,C:Set)B->(C->A)->(A*B->B)->(list C)->B
\end{verbatim} is given by:

\begin{verbatim}
Definition Ext_list :=[A,B,C:Set][b:B][f:C->A][e:A*B->B](cata_list C B b [ym:C*B]let (y,m)=ym in (e ((f y),m))).
\end{verbatim}

and, we can obtain \((\texttt{cata_list A B b e})\) as \((\texttt{Ext_list A B A b i d e})\)

Similarly, as with the factorial function on \texttt{nat}, we can define with \texttt{list_rec}, functions that are beyond catamorphisms.

For example, given \(f : A \rightarrow B, \oplus : A \times B \rightarrow B\), the leftwards fold \([6]\):

\begin{equation}
(f,\oplus)([a_1,\ldots,a_n]) = ((a_1 \oplus (a_2 \oplus \cdots \oplus (a_{n-1} \oplus (f a_n))))
\end{equation}

\begin{verbatim}
Definition leftwards_fold :=[A,B:Set][f:A->B][b[A:B][h:A*B->B]let h'=(curry A B B h) in
(list_rec A [l:(list A)]B b ((y:A)[l:(list A)][x:B] Cases l of Nil=>(f y) |(Cons a l')=>(h' y x) end)).
\end{verbatim}

\texttt{can not be defined as a catamorphism for every \(f : A \rightarrow B\). Of course, if \(f\) is the identity then \texttt{leftwards_fold} = (\texttt{cata_list A A b h'}).}

The fusion law on polymorphic lists can also be proved directly in Coq:

\begin{verbatim}
Goal : (A,B,C:Set; b:B; f:A*B -> B; c:C;g:A*C -> C;h: B->C)((h b)=c) -> ((a:A;x:B)(h (f (a,x)))=(g (a, (h x))))

->(l : (list A))((h (cata_list A B b f l))=(cata_list A C c g l)).
\end{verbatim}

\texttt{Proof.}
\texttt{Intros.}
\texttt{Elim 1.}
\texttt{Simpl; Auto.Intros.Simpl.Rewrite (H0 y (cata_list A B b f l0)).}
\texttt{Rewrite H1.Trivial.Qed.}

Let us now consider the classical example of making the sum of the squares of the elements of a list of naturals.

Let \(h : (\text{list nat}) \rightarrow \text{nat} \) the function \([l:(list nat)](\text{cata_list nat nat (0) Plus l})\) and \(g : \text{nat \times (list nat)} \rightarrow (\text{list nat})\), and \(g' : \text{nat \times nat} \rightarrow \text{nat}\) the auxiliar functions \(g(x,[a_1,\ldots,a_n]) = [x^2, a_1, \ldots, a_n]\), which in Coq will be: \([ym: \text{nat\times(list nat)}] \text{let (y,m)=ym in (Cons nat (square y) m)})\) and \(g'(x,y) = x^2 + y\).
Then, as in the general case, the commutativity of this diagram implies

\[(\text{cata }((\text{Nil nat}), g)) \cdot h = (\text{cata } (O, g'))\]

which expresses the optimization obtained on employing \((\text{cata } (O, g'))\) instead of \((\text{cata } ((\text{Nil nat}), g)) \cdot h\).

In Coq, we can extract the functional programs of \(F = (\text{cata } ((\text{Nil nat}), g)); h\) and \(sF = (\text{cata } (O, g'))\) in, for example, Caml:

```plaintext
Require Extraction.
Write Caml File "F" [F].
Write Caml File "sF" [sF]
Then, we found the known equivalence of the programs:
```

<table>
<thead>
<tr>
<th>F</th>
<th>sF</th>
</tr>
</thead>
</table>
| let rec sum_of_list = function
  [] -> 0
  |(y::l1) ->
  (y+(sum_of_list l1));; |
  let f l = sum_of_list
  (let rec f2 = function
   [] -> []
   |(y::l1) -> (square y)::(f2 l1)
   in f2 l);;               |
|                            | let rec sF = function
  [] -> 0
  |(y::l1) ->
  (square y)+(sF l1);;     |
```

This process of optimization which involves the elimination of intermediate values produced by catamorphisms, can be extended to the procedures obtained with the recursive schemes associated with inductive types.
3 Fusion on Lists

It can now be seen how the fusion theorem can be extended to whatever program obtained with \texttt{list-rec}, although what results is not a catamorphism.

Let us consider \( f : A \rightarrow B \), and two families of sets \( P : (\text{list } A) \rightarrow \text{Set} \), \( Q : (\text{list } B) \rightarrow \text{Set} \). and, for each \( x : A \), \( y : B \) and \( l : (\text{list } A) \), \( m : (\text{list } B) \) two functions \( t_1, t_2 \) such that \( (t_1 \, x \, l) : (P \, l) \rightarrow (P \, (\text{Cons} \, x \, l)) \), and \( (t_2 \, y \, m) : (Q \, m) \rightarrow (Q \, (\text{Cons} \, y \, m)) \), and two elements \( a : (P \, \text{nil}) \), \( b : (Q \, \text{nil}) \). Finally, for every list \( l : (\text{list } A) \), a function \( h \) with \( (h \, l) : (P \, l) \rightarrow (Q \, (\text{maplist} \, f \, l)) \), so that:

\[
(h \, \text{nil} \, a) = b \quad \text{and} \quad \forall x : A \quad (t_1 \, x \, l) \, ; \, (h \, x \, l) = (h \, l) \, ; \, (t_2 \, f \, x) \, (\text{maplist} \, f \, l).
\]

Then

\[
(h \, l \, (\text{list-rec} \, A \, P \, \text{a} \, t_1 \, l)) =
((\text{list-rec} \, B \, Q \, b \, t_2) \, (\text{maplist} \, A \, B \, f \, l)).
\]

in Coq:

\[
\text{Theorem Fusion_list_Gen:}(A,B:\text{Set})(P:(\text{list } A)\rightarrow\text{Set})
\]

\[
(Q:(\text{list } B)\rightarrow\text{Set})
\]

\[
(a:(P \, \text{nil} \, A)) \quad (b:(Q \, \text{nil} \, B)) \quad (f:A\rightarrow B)
\]

\[
(h:\lambda : (\text{list } A) \, ((P \, l) \rightarrow (Q \, (\text{maplist} \, A \, B \, f \, l))))
\]

\[
(t1 : (x:A) \, (l : (\text{list } A)) \, ((P \, l) \rightarrow (P \, (\text{Cons} \, A \, x \, l))))
\]

\[
(t2 : (y:B) \, (m : (\text{list } B)) \, ((Q \, m) \rightarrow (Q \, (\text{Cons} \, B \, y \, m))))
\]

\[
(((h \, \text{nil} \, A) = b) \rightarrow
(((x:A) \, (l : (\text{list } A)) \, ((z : (P \, l)) \, (t2 \, f \, x) \, (\text{maplist} \, A \, B \, f \, l) \, (h \, l \, z)) =
(h \, (\text{Cons} \, A \, x \, l) \, (t1 \, x \, l \, z) ))) \rightarrow
(((l : (\text{list } A)) \, (h \, l \, ((\text{list-rec} \, A \, P \, \text{a} \, t1 \, l)) =
((\text{list-rec} \, B \, Q \, b \, t2) \, (\text{maplist} \, A \, B \, f \, l))))).
\]

Proof.

Intros; Elim \( 1 \); Simpl; Trivial. Intros; Simpl. Replace \( (h \, (\text{Cons} \, A \, y \, 10) \, (t1 \, y \, 10 \, (\text{list-rec} \, A \, P \, \text{a} \, t1 \, 10))) \) with
\( (t2 \, f \, y) \, (\text{maplist} \, A \, B \, f \, 10) \, (h \, 10 \, (\text{list-rec} \, A \, P \, \text{a} \, t1 \, 10))) \).

Replace \( (\text{list-rec} \, B \, Q \, b \, t2) \, (\text{maplist} \, A \, B \, f \, 10) \) with
\( (h \, 10 \, (\text{list-rec} \, A \, P \, \text{a} \, t1 \, 10)) \).

Trivial.

Apply \( (H0 \, y \, 10 \, (\text{list-rec} \, A \, P \, \text{a} \, t1 \, 10)) \). Qed.

4 Generalised Fusion Theorem

We are attempting to demonstrate a fusion theorem for whatever type defined inductively. To be able to understand the significance of the theorem, we will present a short summary of inductive types and the terms which we are going to need in the presentation. For a detailed description of inductive types see [11].
We begin with a type $I$ inductive, with $n$ parameters $A_1, \ldots, A_n$. In addition, to include the case of dependent types, we will consider that type $I$ is defined with a dependency on natural numbers. The treatment is analogous to whatever other circumstance. Let $N_1, \ldots, N_r, R_1, \ldots, R_s$, its $k$ constructors. Of these the first $r$ are non recursive and the following $s$ are recursive. This means that each type $N_i$ is of the form $(p : \text{nat}) A_1^i \rightarrow A_2^i \rightarrow \cdots \rightarrow A_n^i \rightarrow (I A_1 \ldots A_n p)$ and each type $R_j$ is of the form $(p : \text{nat}) A_1^j \rightarrow A_2^j \rightarrow \cdots \rightarrow (I A_1 \ldots A_n p) \rightarrow \cdots \rightarrow A_{n_j}^j \rightarrow (I A_1 \ldots A_n (F p))$, where $F$ is a function of the naturals. Note that $I$ has exactly the same arguments as those with which it was previously defined, which excludes the possibility of generating nested types.

On defining the inductive type, the Coq system automatically generates the functional $I_{\text{rec}}$, which represents the expansion of primitive recursion over the type $I$. Given a family of sets $P : (p : \text{nat})(I A_1 \ldots A_n p) \rightarrow \text{Set}$ and morphisms $f_i$ of type $\Delta(T_i, P)$, one for each constructor $T_i$, where these types are the following, depending on whether the arguments of the constructors are recursive or not:

1. if the arguments are not recursive, $\Delta(N_i, P) \equiv (p : \text{nat})(x_1^i : A_1^i) \ldots (x_{n_i}^i : A_{n_i}^i)(P p (N_i A_1 \ldots A_n p x_1^i x_2^i \ldots x_{n_i}^i))$.
2. if the $k$ argument of constructor $R_j$ is recursive (or $(I A_1 \ldots A_n p)$ for a particular natural $p$), then $\Delta(R_j, P) \equiv (p : \text{nat})(x_1^j : A_1^j) \ldots (x_{k-1}^j : A_{k-1}^j)(x : (I A_1 \ldots A_n p))(P p x) \rightarrow \cdots \rightarrow (x_{n_j}^j : A_{n_j}^j) \rightarrow (P (F p) (R_j A_1 \ldots A_n p x_1^j \ldots x_{k-1}^j x \ldots x_{n_j}^j))$.

Therefore, denoting $\triangleright$ the reduction of terms, the semantic of $I_{\text{rec}}$ is as follows:

1. $I_{\text{rec}} A_1 \ldots A_n P f_1 \ldots f_k p (N_i p u_1 \ldots u_{n_i}) \triangleright f_i p u_1 \ldots u_{n_i}$
2. $I_{\text{rec}} A_1 \ldots A_n P f_1 \ldots f_k p (R_j p u_1 \ldots x \ldots u_{n_i}) \triangleright f_j p u_1 \ldots x (I_{\text{rec}} A_1 \ldots A_n P f_1 \ldots f_k p x) \ldots u_{n_i}$.

Let us now consider a family of sets $B_1, \ldots, B_n$ and morphisms $t_i : A_i \rightarrow B_i$, $1 \leq i \leq n$, and a natural $p$. The morphism $\text{mapI} : (I A_1 \ldots A_n p) \rightarrow (I B_1 \ldots B_n p)$, changes through $t_i$, in each constructor of $(I A_1 \ldots A_n p)$, the values of type $A_i$ for their corresponding values in $B_i$.

Let us take, in addition $Q : (p : \text{nat})(I B_1 \ldots B_n p) \rightarrow \text{Set}$ another family of sets and $\{g_i\}$ a family of morphisms similar to the $\{f_i\}$ changing the family $P$ with $Q$. Finally, let $h : (p : \text{nat})(x : (I A_1 \ldots A_n p))(P p x) \rightarrow (Q p (\text{mapI} A_1 \ldots A_n B_1 \ldots B_n t_1 \ldots t_n x))$, satisfying:

\[h_1\]
\[
(p : \text{nat})(x_1^i : A_1^i) \ldots (x_{n_i}^i : A_{n_i}^i)(h p (N_i A_1 \ldots A_n p x_1^i \ldots x_{n_i}^i))(f_i p x_1^i \ldots x_{n_i}^i) = (g_i p (t_1^i x_1^i) \ldots (t_{n_i}^i x_{n_i}^i)).
\]

\[h_2\]
\[
(p : \text{nat})(x_1^i : A_1^i) \ldots (x_{n_i}^i : A_{n_i}^i) (l : I A_1^i \ldots A_{n_i}^i) (y : (P l))(h (F p) (R_j A_1 \ldots A_n p x_1^j \ldots y \ldots x_{n_j}^j))(f_j p x_1^j \ldots y \ldots x_{n_j}^j) = (g_j p (t_{n_j}^j x_{n_j}^j) \ldots (\text{mapI} A_1 \ldots A_n B_1 \ldots B_n t_1 \ldots t_n p l)(h p l y) \ldots (t_{n_j}^j x_{n_j}^j)).
\]

we are avoiding inconsistences that might occur in the general case
Then, the extended theorem of fusion establishes that for each natural \( p \) and each \( x : (I \ A_1 \ldots A_n) \)

\[
(h \ p \ x \ (I_{rec} \ P \ f_1 \ldots f_k \ p \ x))
\]

\[
= (I_{rec} \ Q \ g_1 \ldots g_k \ p \ (mapI \ A_1 \ldots A_n \ B_1 \ldots B_n \ t_1 \ldots t_n \ p \ x)).
\]

To demonstrate this, let us suppose that, in the first place, we are dealing with a non recursive constructor. So

\[
(p : nat)(x_1^i : A_1^i)\ldots(x_n^i : A_n^i)(h \ p \ (N_i \ A_1^i \ldots A_n^i \ p \ x_1^i \ldots x_n^i))
\]

\[
= (h \ p \ (N_i \ A_1^i \ldots A_n^i \ p \ x_1^i \ldots x_n^i))
\]

= \((g_i \ p \ (t_1^i \ x_1^i)\ldots(t_n^i \ x_n^i))\) from the hypothesis \( h1 \) assumed on the function \( h \). But this last expression coincides with \((I_{rec} \ B_1 \ldots B_n \ Q \ g_1 \ldots g_k \ p \ (mapI \ A_1 \ldots A_n \ B_1 \ldots B_n \ t_1 \ldots t_n \ p \ (N_i \ A_1^i \ldots A_n^i \ p \ x_1^i \ldots x_n^i))\)).

Now, with the recursive constructor \( R_j : A_1^j \rightarrow \ldots \rightarrow (I \ A_1 \ldots A_n \ p) \rightarrow \ldots \rightarrow A_n^j \rightarrow (I \ A_1 \ldots A_n \ p) \) we want to see that

\[
(p : nat)(x_1^j : A_1^j)\ldots(x : (I \ A_1 \ldots A_n))\ldots(x_n^j : A_n^j)
\]

\[
= (h \ (F \ p) \ ((R_j \ A_1 \ldots A_n \ Ax_1^j \ldots x \ldots x_n^j)) \ (I_{rec} \ A_1 \ldots A_n \ P \ f_1 \ldots f_k \ p)
\]

\[
= (g_j \ p \ (t_1^j \ x_1^j)\ldots(mapI \ A_1 \ldots A_n \ B_1 \ldots B_n \ h_1 \ldots h_n \ p \ x) \ldots
\]

\[
= (t_n^j \ x_n^j)).
\]

But, employing the hypothesis of induction assumed about these constructors, when we applied them to the element \( x \), we can replace the term \((I_{rec} \ B_1 \ldots B_n \ Q \ g_1 \ldots g_n \ p \ (mapI \ A_1 \ldots A_n \ B_1 \ldots B_n \ h_1 \ldots h_n \ p \ x))\) for \((h \ p \ x \ (I_{rec} \ A_1 \ldots A_n \ P \ f_1 \ldots f_n \ p \ x))\), leaving the equality looked for in the following form:

\[
(h \ (F \ p) \ ((R_j \ A_1 \ldots A_n \ Ax_1^j \ldots x \ldots x_n^j)) \ (f_j \ p \ x_1^j \ldots x)
\]

\[
= (g_j \ p \ (t_1^j \ x_1^j)\ldots(mapI \ A_1 \ldots A_n \ B_1 \ldots B_n \ t_1 \ldots t_n \ p \ x)
\]

\[
= (t_n^j \ x_n^j))\) which is the condition we have imposed on hypothesis \( h2 \) assumed about the function \( h \), therefore, demonstrating the proposition.

### 4.1 An Example

Now we describe an example using dependent types. We require the length function on lists and a parity test of naturals.

**Definition**

\[\text{length := } [\text{A : Set}]
\]

\[\text{list}_\text{rec} \text{ A} \ [1 : (\text{list A})] \text{nat} \ 0 \ [\_ : \text{A}][\_ : (\text{list A})][\_ : \text{nat}](S \ n)). \]
Definition iseven := (nat_rec [::_:nat]bool true [::_:nat][b:bool](if b then false else true)).

Now we define the dependent type \texttt{ListN} where \texttt{(ListN A n)} specifies a list of elements of \texttt{A} with length \texttt{n + 1}.

\begin{verbatim}
Inductive ListN [A:Set]:nat->Set:=
 ListN_0:A->(ListN A 0)
 | ListN_S:(n:nat)A->(ListN A n)->(ListN A (S n)).
\end{verbatim}

We can forget the constraint of the length implicit in \texttt{de definition of ListN} by means of \texttt{trad:(A:Set; n:nat)(ListN A n)->(list A)}

\begin{verbatim}
Definition trad:=[A:Set]
 (ListN_rec A [n:nat][::_:(ListN A n)](list A)
  [y:A](Cons A y (Nil A))
 [n:nat][y:A][l:(ListN A n)][lis:(list A)](Cons A y lis)).
\end{verbatim}

and then give a proof of the specification:

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

Note that, if we employ the extraction mechanism of Coq we obtain

\begin{verbatim}
Extraction ListN.
 ListN ==>
 Inductive ListN [A:Set] : Set :=
 ListN_0 : A->(ListN A)
 | ListN_S : nat->A->(ListN A)->(ListN A)
\end{verbatim}

and, naturally, extracting to, for example Caml, we obtain:

\begin{verbatim}
type 'a listN =
 ListN_0 of 'a
 | ListN_S of nat * 'a * ('a listN)
\end{verbatim}

given the lack of dependent types in this language.

We now define the two families of sets indexed by lists of naturals \texttt{P,Q : (list nat) \rightarrow Set}.

\begin{verbatim}
Definition P := [xs:(list nat)](ListN nat (length nat xs)).
 Definition Q := [xs:(list nat)](ListN bool (length nat xs)).
\end{verbatim}

and the mapping function of \texttt{ListN}

\begin{verbatim}
Definition mapListN := [A,B:Set][f:A->B]
 (ListN_rec A [n:nat][::_:(ListN A n)](ListN B n)
  [y:A](ListN_0 B (f y))
 [n:nat][y:A][::_:(ListN A n)][x:(ListN B n)]
 (ListN_S B n (f y) x)).
\end{verbatim}
The function $f$

**Definition** $f := \[x:\text{nat}][l:\text{(list nat)}][xs:\text{(P l)}] (\text{ListN}_S \text{ nat} (\text{length nat} \ 1) (\text{plus} \ x (\text{length nat} \ 1)) \ xs)$.

adds $x + \text{length}(l)$ to $xs:\text{(P l)}$, and the function $g$

**Definition** $g := \[x:\text{nat}][l:\text{(list nat)}][ys:\text{(Q l)}] (\text{if} \ (\text{iseven} \ (\text{plus} \ x (\text{length nat} \ 1))) \ \text{then} \ (\text{ListN}_S \text{ bool} (\text{length nat} \ 1) \text{ true} \ ys) \ \text{else} \ (\text{ListN}_S \text{ bool} (\text{length nat} \ 1) \text{ false} \ ys))$.

adds true or false to $ys:\text{(Q l)}$ depending on whether $x + \text{length}(l)$ has an even or odd value.

Now,

**Definition** $h :=[l:\text{(list nat)}]$ 
(\text{mapListN} \text{ nat bool} \text{ iseven} \ (\text{length nat} \ 1))$.

verifies the hypothesis $h_1$ and $h_2$. For example:

**Goal** $(x:\text{nat})(l:\text{(list nat)})(xs:\text{(P l)}) ((h \ (\text{Cons nat} \ x l) \ (f x l xs))=(g x l (h l xs)))$.

Proof

Intros.
Unfold $h$; Unfold $f$; Unfold $g$.
Simpl.
Case (iseven (plus x (length nat l))).
Trivial.
Trivial.
Save alpha.

Therefore, as a consequence of the general fusion theorem, the two programs:

**Definition** $\text{FN} :=[l:\text{(list nat)}]$ 
(\text{h l (list_rec nat P (ListN}_0 \text{ nat 0) f l)}$).

and

**Definition** $\text{FB} :=[l:\text{(list nat)}]$ 
(\text{list_rec nat Q (ListN}_0 \text{ bool true) g l)}$).

must be equivalent.

Note that to evaluate $\text{FN}$ we firstly construct the corresponding list of naturals and then apply the function $f$, but when we use $\text{FB}$ we work directly with each element of the list.

This is easy to check also in Coq:

**Goal** $(l:\text{(list nat)})(\text{FN} \ l)=(\text{FB} \ l)$.

Proof.
Induction $l$.
Trivial.
5 Conclusions and Future Work

We have shown how the logical framework Coq, based on the Calculus of Inductive Constructions, can be used to prove concrete applications of program transformation rules, using its expressivity and also its powerful Extraction mechanism.

In the general case, with dependent types, fusion theorem is proved using Coq for the type list. Also a proof of the fusion theorem for the recursion scheme of any inductive type is given.

For future work it would be interesting to build a new Extraction tool for Coq to a language with dependent types as Augustsson’s Cayenne, Cardelli’s Quest, Boehm’s Russell, Xi’s deCaml or others.

References

Abstract. We present a Language Prototyping System that facilitates the modular development of interpreters from semantic specifications. The theoretical basis of our system is the integration of ideas from generic programming and modular monadic semantics. The system is implemented as a domain-specific language embedded in Haskell and contains an interactive framework for language prototyping.

In the monadic approach, the semantic specification of a programming language is captured as a function $\Sigma \rightarrow M V$ where $\Sigma$ represents the abstract syntax, $M$ the computational monad, and $V$ the domain value. In order to obtain more extensibility, we use folds or catamorphisms over the fixpoint of non-recursive pattern functors that capture the structure of the abstract syntax. For each pattern functor $F$, the semantic specifications are defined as independent $F$-Algebras whose carrier is $M V$, where $M$ is the computational monad and $V$ models the domain value.

The computational monad $M$ can itself be obtained from the composition of several monad transformers applied to a base monad, and the domain value $V$ can be defined using extensible union types.

In this paper, we also show that when the abstract syntax contains several categories, it is possible to define many-sorted algebras obtaining the same modularity.

1 Introduction

E. Moggi [39] applied monads to denotational semantics in order to capture the notion of computation and the intuitive idea of separating computations from values.

After his work, there was some interest in the development of modular interpreters using monads [49, 43, 10]. The problem was that, in general, it is not possible to compose two monads to obtain a new monad [25]. A proposed solution was the use of monad transformers [63, 82] which transform a given monad into a new one adding new operations. This approach was called modular monadic semantics.

In a different context, the definition of recursive datatypes as least fixpoints of pattern functors and the calculating properties that can be obtained be means
of folds or catamorphisms led to a complete discipline which could be named as
generic programming [3, 34, 35].

In [30], L. Duponcheel proposed the combined use of folds or catamorphisms with modular monadic semantics allowing the independent specification of the abstract syntax, the computational monad and the domain value.

Following [36], we applied monadic folds to modular monadic semantics allowing the separation between recursive evaluation and semantic specification [28, 29, 30].

In practice, the abstract syntax is usually formed from $n$ mutually recursive categories. In this paper we show how we can extend our previous work to handle many-sorted algebras.

The paper is organized as follows. In section 2 we give an informal presentation of modular monadic semantics defining some monad transformers. Section 3 presents the basic concepts from generic programming extending previous work to handle many-sorted algebras. In section 4 we specify the semantics of a simple imperative programming language from reusable components.

Along the paper, we use Haskell syntax with some freedom in the use of mathematical operators and datatype declarations. As an example, the predefined datatype

\[ \text{data } \text{Either a b } = \text{Left a } | \text{ Right b } \]

could be defined with our notation as

\[ \alpha \parallel \beta \triangleq L\alpha \mid R\beta \]

We also omit the type constructors in some definitions for brevity. The notions we use from category theory are defined in the paper, so it is not a prerequisite.

## 2 Modular Monadic Semantics

A monad $M$ captures the intuitive notion of computation. In this way, the type $M\alpha$ represents a computation the returns a value of type $\alpha$.

In functional programming, a monad can be defined as a type constructor $M$ with 2 operations

\[ \text{return} : \alpha \rightarrow M\alpha \]
\[ (\gg\gg) : M\alpha \rightarrow (\alpha \rightarrow M\beta) \rightarrow M\beta \]

which satisfy a number of laws (see [17, 19, 38]).

**Example 1.** The simplest monad is the identity monad

\[ \text{Id } \alpha \triangleq \alpha \]
\[ \text{return} = \lambda x \rightarrow x \]
\[ m \gg\gg f = f \ x \]
In the rest of the paper, we will use the do-notation defined as:

\[
\begin{align*}
\text{do} \{ m; e \} & \equiv m \gg\gg = \lambda m \rightarrow \text{do} \{ e \} \\
\text{do} \{ x \leftarrow m; e \} & \equiv m \gg\gg = \lambda x \rightarrow \text{do} \{ e \} \\
\text{do} \{ \text{let} \ exp; e \} & \equiv \text{let} \ exp \in \text{do} \{ e \} \\
\text{do} \{ e \} & \equiv e
\end{align*}
\]

It is possible to define monads that capture different kinds of computations, like partiality, nondeterminism, side-effects, exceptions, continuations, interactions, etc. [39, 40, 5]. Table 1 presents two classes of monads that will be used in the rest of the paper.

<table>
<thead>
<tr>
<th>Name</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Access</td>
<td>( rdEnv : \mathcal{M} \mathcal{E}nv )</td>
</tr>
<tr>
<td></td>
<td>( inEnv : \mathcal{E}nv \rightarrow \mathcal{M} \alpha \rightarrow \mathcal{M} \alpha )</td>
</tr>
<tr>
<td>State transformer</td>
<td>( update : (\mathcal{S}tate \rightarrow \mathcal{S}tate) \rightarrow \mathcal{M} \mathcal{S}tate )</td>
</tr>
<tr>
<td></td>
<td>( fetch : \mathcal{M} \mathcal{S}tate )</td>
</tr>
<tr>
<td></td>
<td>( set : \mathcal{S}tate \rightarrow \mathcal{M} \mathcal{S}tate )</td>
</tr>
</tbody>
</table>

When describing the semantics of a programming language using monads, the main problem is the combination of different classes of monads. It is not possible to compose two monads to obtain a new monad in general [25]. Nevertheless, a monad transformer \( T \) can transform a given monad \( \mathcal{M} \) into a new monad \( T \mathcal{M} \) that has new operations and maintains the operations of \( \mathcal{M} \). The idea of monad transformer is based on the notion of monad morphism that appeared in Moggi’s work [39] and was later proposed in [33]. The definition of a monad transformer is not straightforward because there can be some interactions between the intervening operations of the different monads. These interactions are considered in more detail in [31, 32, 33] and in [17] it is shown how to derive a backtracking monad transformer from its specification.

Our system contains a library of predefined monad transformers corresponding to each class of monad and the user can also define new monad transformers. When defining a monad transformer \( T \) over a monad \( \mathcal{M} \), it is necessary to specify the new \textit{return} and \((\gg\gg=)\), the \textit{lift} \( \mathcal{M} \alpha \rightarrow T \mathcal{M} \alpha \) operation that transforms any operation in \( \mathcal{M} \) into an operation in the new monad \( T \mathcal{M} \), and the new operations provided by the new monad. Table 2 presents the definitions of the two monad transformers that will be used in the paper.

### 2.1 Extensible Domains

[33] defines extensible union types using multi-parameter type classes. Although we are not going to give the full details, we can assume that if \( \alpha \) is a subtype
Table 2. Some monad transformers with their definitions

<table>
<thead>
<tr>
<th>Environment reader</th>
<th>State transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{Env}} \ M \alpha \triangleq \text{Env} \to M \alpha$</td>
<td>$T_{\text{State}} \ M \alpha \triangleq \text{State} \to M(\alpha, \text{State})$</td>
</tr>
<tr>
<td>$\text{return } x = \lambda \rho \to \text{return}\ x$</td>
<td>$\text{return } x = \lambda \varsigma \to \text{return}(x, \varsigma)$</td>
</tr>
<tr>
<td>$x \triangleright\triangleright f = \lambda \rho \to (x \triangleright\triangleright (\lambda a \to f\ a\ \rho))$</td>
<td>$x \triangleright\triangleright f = \lambda \varsigma \to (x \triangleright\triangleright (\lambda x \to \text{return}(x, \varsigma)))$</td>
</tr>
<tr>
<td>$\text{lift } x = \lambda \rho \to x \triangleright\triangleright \text{return}$</td>
<td>$\text{update } f = \lambda \varsigma \to \text{return}(\varsigma, f\varsigma)$</td>
</tr>
<tr>
<td>$\text{rdEnv} = \lambda \rho \to \text{return}\ \rho$</td>
<td>$\text{fetch} = \text{update}(\lambda \varsigma \to \varsigma)$</td>
</tr>
<tr>
<td>$\text{inEnv } \rho \ x = \lambda \to x\ \rho$</td>
<td>$\text{set } \varsigma = \text{update}(\lambda \to \varsigma)$</td>
</tr>
</tbody>
</table>

of $\beta$, which will be denoted as $\alpha \in \beta$, then we have the functions $\uparrow: \alpha \to \beta$ and $\downarrow: \beta \to \alpha$. We also assume that $\alpha \in (\alpha \parallel \beta)$ and that $\beta \in (\alpha \parallel \beta)$.

As an example, if we define a domain of integers and booleans as $\text{Int} \parallel \text{Bool}$, then $(\uparrow 3)$ belongs to that domain and to further extensions of it.

3 Generic Programming Concepts

3.1 Functors, Algebras, and Catamorphisms

As in the case of monads, functors also come from category theory but can easily be defined in a functional programming setting. A functor $F$ can be defined as a type constructor that transforms values of type $\alpha$ into values of type $F\alpha$ and a function $\text{map}_F: (\alpha \to \beta) \to F\alpha \to F\beta$.

The fixpoint of a functor $F$ can be defined as

$$\mu F \triangleq \text{In}(F(\mu F))$$

In the above definition, we explicitly write the type constructor $\text{In}$ because we will refer to it later.

A recursive datatype can be defined as the fixpoint of a non-recursive functor that captures its shape.

**Example 2.** The following inductive datatype for arithmetic expressions $\text{Term}$

$$\text{Term} \triangleq N\ \text{Int} \mid \text{Term} + \text{Term} \mid \text{Term} - \text{Term}$$
can be defined as the fixpoint of the functor \( A \)

\[
T \ x \triangleq \ N\ \text{Int} \mid x + x \mid x - x
\]

where the \( \text{map}_T \) is:

\[
\begin{align*}
\text{map}_T &: (\alpha \to \beta) \to (T \alpha \to T \beta) \\
\text{map}_T f (N \ n) &= n \\
\text{map}_T f (x_1 + x_2) &= f \ x_1 + f \ x_2 \\
\text{map}_T f (x_1 - x_2) &= f \ x_1 - f \ x_2
\end{align*}
\]

Once we have the shape functor \( T \), we can obtain the recursive datatype as the fixpoint of \( T \)

\[
\text{Term} \triangleq \mu T
\]

In this way, the expression \( 2 + 3 \) can be represented as

\[
\text{In } ((\text{In } (N \ 2)) + (\text{In } (N \ 3))) : \text{Term}
\]

The sum of two functors \( F \) and \( G \), denoted by \( F \oplus G \) can be defined as

\[
(F \oplus G) \ x \triangleq F \ x \parallel G \ x
\]

where \( \text{map}_{F \oplus G} \) is

\[
\begin{align*}
\text{map}_{F \oplus G} &: (\alpha \to \beta) \to (F \oplus G) \alpha \to (F \oplus G) \beta \\
\text{map}_{F \oplus G} f (L \ x) &= L \ (\text{map}_F f \ x) \\
\text{map}_{F \oplus G} f (R \ x) &= R \ (\text{map}_G f \ x)
\end{align*}
\]

Using the sum of two functors, it is possible to extend recursive datatypes.

**Example 3.** We can define a new pattern functor for boolean expressions

\[
B \ x = B\ \text{Bool} \mid x == x \mid x < x
\]

and the composed recursive datatype of arithmetic and boolean expressions can easily be defined as

\[
\text{Expr} \triangleq \mu (T \oplus B)
\]

Given a functor \( F \), an \( F \)-algebra is a function \( \varphi_F : F \alpha \to \alpha \) where \( \alpha \) is called the carrier. An homomorphism between two \( F \)-algebras \( \varphi : F \alpha \to \alpha \) and \( \psi : F \beta \to \beta \) is a function \( h : \alpha \to \beta \) which satisfies

\[
h \cdot \varphi = \psi \cdot \text{map}_F h
\]

\footnote{In the rest of the paper we omit the definition of \( \text{map} \) functions as they can be automatically derived from the shape of the functor.}
We consider a new category with $F$-algebras as objects and homomorphisms between $F$-algebras as morphisms. In this category, $In : F(\mu F) \rightarrow \mu F$ is an initial object, i.e. for any $F$-algebra $\varphi : F \alpha \rightarrow \alpha$ there is a unique homomorphism $(\langle \varphi \rangle) : \mu F \rightarrow \alpha$ satisfying the above equation.

$(\langle \varphi \rangle)$ is called fold or catamorphism and satisfies a number of calculational properties [3, 6, 35, 42]. It can be defined as:

$$(\langle \varphi \rangle) : (F \alpha \rightarrow \alpha) \rightarrow (\mu F \rightarrow \alpha)$$

$$(\langle \varphi \rangle)(In x) = \varphi(map_F(\langle \varphi \rangle)x)$$

**Example 4.** We can obtain a simple evaluator for arithmetic expressions defining an $T$-algebra whose carrier is the type $m v$, where $m$ is, in this case, any kind of monad, and $Int$ is a subtype of $v$.

$\varphi_T : (Monad m, Int \in v) \Rightarrow T(m v) \rightarrow m v$

$\varphi_T(Num n) = return(↑ n)$

$\varphi_T(e_1 + e_2) =$ do

$v_1 \leftarrow e_1$
$v_2 \leftarrow e_2$
return(↑ (↓ $v_1 +$ $v_2$))

$\varphi_T(e_1 - e_2) =$ do

$v_1 \leftarrow e_1$
$v_2 \leftarrow e_2$
return(↑ (↓ $v_1 -$ $v_2$))

Applying a catamorphism over $\varphi_T$ we obtain the evaluation function for terms:

$eval_{Term} : (Monad m, Int \in v) \Rightarrow Term \rightarrow m v$

$eval_{Term} = (\langle \varphi_T \rangle)$

The operator $\oplus$ allows to obtain a $(F \oplus G)$-algebra from an $F$-algebra $\varphi$ and a $G$-algebra $\psi$

$\oplus : (F \alpha \rightarrow \alpha) \rightarrow (G \alpha \rightarrow \alpha) \rightarrow (F \oplus G)\alpha \rightarrow \alpha$

$(\varphi \oplus \psi)(L x) = \varphi x$

$(\varphi \oplus \psi)(R x) = \psi x$

**Example 5.** The above definition allows to extend the evaluator of example $\text{II}$ to arithmetic and boolean expressions.

We can specify the semantics of boolean expressions with the following $B$-algebra

$\varphi_B : (Monad m, Bool \in v) \Rightarrow B(m v) \rightarrow m v$

$\varphi_B(B b) =$ return(↑ $b$)
A Language Prototyping Tool Based on Semantic Building Blocks

\[ \varphi_B (e_1 == e_2) = \text{do} \]
\[ v_1 \leftarrow e_1 \]
\[ v_2 \leftarrow e_2 \]
\[ \text{return}(\uparrow (\downarrow v_1 == \downarrow v_2)) \]

\[ \varphi_B (e_1 < e_2) = \text{do} \]
\[ v_1 \leftarrow e_1 \]
\[ v_2 \leftarrow e_2 \]
\[ \text{return}(\uparrow (\downarrow v_1 < \downarrow v_2)) \]

Now, the new evaluator of boolean and arithmetic expressions is automatically obtained as a catamorphism over the \((T \oplus B)\)-algebra.

\[ \text{eval}_{Expr} : (\text{Monad } m, \text{Int } \in v, \text{Bool } \in v) \Rightarrow Expr \rightarrow m v \]
\[ \text{eval}_{Expr} = (\varphi_T \oplus \varphi_B) \]

The theory of catamorphisms can be extended to monadic catamorphisms as described in [12, 15, 28, 30].

### 3.2 Many-Sorted Algebras and Catamorphisms

The abstract syntax of a programming language is usually divided in several mutually recursive categories. It is possible to extend the previous definitions to handle many-sorted algebras. In this section, we present the theory for \(n = 2\), but it can be defined for any number of sorts [11, 37, 21, 41].

A bifunctor \(F\) is a type constructor that assigns a type \(F \alpha \beta\) to a pair of types \(\alpha\) and \(\beta\) and an operation

\[ \text{bimap}_F : (\alpha \rightarrow \gamma) \rightarrow (\beta \rightarrow \delta) \rightarrow (F \alpha \beta \rightarrow F \gamma \delta) \]

The fixpoint of two bifunctors \(F\) and \(G\) is a pair of values \((\mu_1 FG, \mu_2 FG)\) that can be defined as:

\[ \mu_1 FG \triangleq \text{In}_1 (F (\mu_1 FG) (\mu_2 FG)) \]
\[ \mu_2 FG \triangleq \text{In}_2 (G (\mu_1 FG) (\mu_2 FG)) \]

Given two bifunctors \(F\) and \(G\), a two-sorted \(F, G\)-algebra is a pair of functions \((\varphi, \psi)\) such that:

\[ \varphi : F \alpha \beta \rightarrow \alpha \]
\[ \psi : G \alpha \beta \rightarrow \beta \]

where \(\alpha, \beta\) are called the carriers of the two-sorted algebra.

It is possible to define \(F, G\)-homomorphisms and a new category where \((\text{In}_1, \text{In}_2)\) form the initial object. This allows the definition of bicatamorphisms as:

\[ (\text{[-]}_1) : (F \alpha \beta \rightarrow \alpha) \rightarrow (G \alpha \beta \rightarrow \beta) \rightarrow (\mu_1 FG \rightarrow \alpha) \]
\[(\varphi, \psi)_1 (In_1 x) = \varphi (\text{bimap}_F (\varphi, \psi)_1 (\varphi, \psi)_2 x)\]

\[(\varphi, \psi)_2 (In_2 x) = \psi (\text{bimap}_G (\varphi, \psi)_1 (\varphi, \psi)_2 x)\]

The sum of two bifunctors \(F\) and \(G\) is a new bifunctor \(F \boxplus G\) and can be defined as:

\[(F \boxplus G) \alpha \beta \triangleq F \alpha \beta || G \alpha \beta\]

where the \textit{bimap} operator is

\[\text{bimap}_{F \boxplus G} : (\alpha \to \gamma) \to (\beta \to \delta) \to ((F \boxplus G) \alpha \beta \to ((F \boxplus G) \gamma \delta)\]

\[\text{bimap}_{F \boxplus G} f g (L x) = L (\text{bimap}_{F \boxplus G} f g x)\]

\[\text{bimap}_{F \boxplus G} f g (R x) = R (\text{bimap}_{F \boxplus G} f g x)\]

In order to extend two-sorted algebras, we define the operators \(\boxplus_1\) and \(\boxplus_2\) as:

\[(\boxplus_1) : (F \alpha \beta \to \alpha) \to (G \alpha \beta \to \alpha) \to (F \boxplus G) \alpha \beta \to \alpha\]

\[(\phi_1 \boxplus_1 \phi_2) (L x) = \phi_1 x\]

\[(\phi_1 \boxplus_1 \phi_2) (R x) = \phi_2 x\]

\[(\boxplus_2) : (F \alpha \beta \to \beta) \to (G \alpha \beta \to \beta) \to (F \boxplus G) \alpha \beta \to \beta\]

\[(\psi_1 \boxplus_2 \psi_2) (L x) = \psi_1 x\]

\[(\psi_1 \boxplus_2 \psi_2) (R x) = \psi_2 x\]

### 3.3 From Functors to Bifunctors

When specifying several programming languages, it is very important to be able to share common blocks and to reuse the corresponding specifications. For example, arithmetic expressions should be specified in one place and their specification should be reused between different languages.

In order to reuse specifications made using single-sorted algebras in a two-sorted framework, it is necessary to extend functors to bifunctors.

Given a functor \(F\), we define the bifunctors \(F^2_1\) and \(F^2_2\) as:

\[F^2_1 \alpha \beta \triangleq F \alpha\]

\[F^2_2 \alpha \beta \triangleq F \beta\]

where the \textit{bimap} operations are defined as

\[\text{bimap}_{F^2_1} f g x = f x\]

\[\text{bimap}_{F^2_2} f g x = g x\]

Given an \(F\)-algebra, the operators \(\epsilon^2_1\) and \(\epsilon^2_2\) obtain the corresponding two-sorted algebras.
\[ \epsilon_1^2 : (F \alpha \rightarrow \alpha) \rightarrow F_1^2 \alpha \beta \rightarrow \alpha \]
\[ \epsilon_1^2 \phi x = \phi x \]

\[ \epsilon_2^2 : (F \beta \rightarrow \beta) \rightarrow F_2^2 \alpha \beta \rightarrow \beta \]
\[ \epsilon_2^2 \phi x = \phi x \]

4 Specification of a Simple Imperative Language

4.1 Abstract Syntax

A typical imperative programming language can be divided in two different worlds: expressions and commands. In our example, the expressions will be arithmetic, boolean and variables. The abstract syntax of arithmetic and boolean expressions are captured by the functors T and B defined in examples 2 and 3.

Variables are defined using the functor V

\[ V x \triangleq V \text{Name} \]

We will define commands in two steps. Firstly, sequence and assignments are defined using the bifunctor S

\[ S e c \triangleq c ; c \mid \text{String} := e \]

Secondly, control structures (conditional and loops) are defined using the bifunctor R

\[ R e c \triangleq \text{If } e c c \mid \text{While } e c \]

In order to define the imperative language, we need a bifunctor that represents the shape of expressions and another one representing commands. The bifunctor of expressions can be defined as an extension of the functor obtained as the sum of T, B and V

\[ E \triangleq (T \oplus B \oplus V)_1^2 \]

The bifunctor of commands is defined as the sum of the bifunctors S and R

\[ C \triangleq S \oplus R \]

Finally, the imperative language is the fixpoint of E and C

\[ \text{Imp} \triangleq \mu_{C} E \]

4.2 Computational Structure

In this simple language, the computational structure needs to access the environment and to transform a global state. We will use the monad Comp which is obtained by transforming the identity monad using the monad transformers T_{State} and T_{Env} defined in table 2.

\[ \text{Comp} \triangleq (T_{State} \cdot T_{Env}) \text{Id} \]
The domain value of expressions consist of integer and boolean values

\[ \text{Value} \triangleq \text{Int} \parallel \text{Bool} \]

and the domain value of commands is the null type \( (\text{\textit{}}) \) indicating that commands do not return any value. The state and environment are defined as:

\[ \text{Env} \triangleq \text{Name} \rightarrow \text{Loc} \]
\[ \text{State} \triangleq \text{Loc} \rightarrow \text{Value} \]

where \( \text{Loc} \) represent memory locations. We will also use the notation \( \varsigma \triangleright \{x/v\} \) to represent the updated state \( \varsigma \) which assigns \( v \) to \( x \).

### 4.3 Semantic Functions

The semantic specification of arithmetic and boolean expressions were defined in the examples 4 and 5. We will reuse those specifications in the imperative language. With regard to variables, the \( \text{V} \)-algebra is

\[ \varphi_V : \text{V} (\text{Comp Value}) \rightarrow \text{Comp Value} \]
\[ \varphi_V (\text{Var } x) = \text{do} \]
\[ \quad \rho \leftarrow \text{rdEnv} \]
\[ \quad \varsigma \leftarrow \text{fetch} \]
\[ \quad \text{return}(\varsigma (\rho x)) \]

The specification of sequence and assignment is

\[ \psi_S : \text{S} (\text{Comp Value}) (\text{Comp (}) \rightarrow \text{Comp (}) \]
\[ \psi_S (c_1 ; c_2) = \text{do} \]
\[ \quad c_1 \]
\[ \quad c_2 \]
\[ \psi_S (x := e) = \text{do} \]
\[ \quad v \leftarrow e \]
\[ \quad \rho \leftarrow \text{rdEnv} \]
\[ \quad \varsigma \leftarrow \text{fetch} \]
\[ \quad \text{set} (\varsigma \triangleright \{\rho x / v\}) \]
\[ \quad \text{return (}) \]

In the same way, the specification of conditional and repetitive commands is:

\[ \psi_R : \text{R} (\text{Comp Value}) (\text{Comp (}) \rightarrow \text{Comp (}) \]
\[ \psi_R (\text{If } e \ c_1 \ c_2) = \text{do} \]
\[ \quad v \leftarrow e \]
\[ \quad \text{if } v \text{ then } c_1 \]
\[ \quad \text{else } c_2 \]

\(^2 (\text{}) \) is a predefined Haskell datatype that only contains the value (\( \text{}) \)
\[ \psi_R \ (\text{While} \ e \ c) = \text{loop} \]

\[
\begin{align*}
\text{where} \\
\text{loop} &= \text{do} \\
& \quad v \leftarrow e \\
& \quad \text{if } v \text{ then} \\
& \quad \text{do } \{ e ; \text{loop} \} \\
& \quad \text{else} \\
& \quad \text{return}()
\end{align*}
\]

Finally, the interpreter is automatically obtained as a bicatamorphism

\[
\text{Inter}_{\text{Imp}} : \ \text{Imp} \to \text{Comp} ()
\]

\[
\text{Inter}_{\text{Imp}} = (\langle \epsilon_T^2 (\varphi_T \oplus \varphi_B \oplus \varphi_V), \varphi_S \boxplus_2 \varphi_R \rangle)_2
\]

Although in the above definition we have explicitly written the particular algebras, it is not necessary to do so in the implementation because the overloading mechanism of Haskell allows to detect which is the corresponding algebra.

## 5 Conclusions and Future Work

We have presented an integration of modular monadic semantics and generic programming concepts that allows the definition of programming languages from reusable semantic specifications.

This approach has been implemented in a Language Prototyping System which allows to share semantic building blocks and provides an interactive framework for language testing. The system can be considered as another example of a domain-specific language embedded in Haskell \[40, 20\]. This approach has some advantages: The development is easier as we can rely on the fairly good type system of Haskell, it is possible to obtain direct access to Haskell libraries and tools, and we do not need to define a new language with its syntax, semantics, type system, etc. At the same time, the main disadvantages are the mixture of error messages from the domain-specific language and the host language, Haskell type system limitations and the Haskell dependency which impedes the development of interpreters implemented in different languages. It would be interesting to define an independent domain specific meta-language for semantic specifications following \[5, 7, 38\].

On the theoretical side, \[17\] shows how to derive a backtracking monad transformer from its specification. That approach should be applied to other types of monad transformers and it would be interesting to define a general framework for the combination many-sorted algebras and monadic catamorphisms. It would also be fruitful to study the combination of algebras, coalgebras, monads and comonads in order to provide the semantics of interactive and object-oriented features \[4, 23, 22, 27, 45\].

Another line of research is the automatic derivation of compilers from the interpreters built. This line has already been started in \[14, 15\].
With regard to the implementation, we have also made a simple version of the system using first-class polymorphism [24] and extensible records [13]. This allows the definition of monads as first class values and monad transformers as functions between monads without the need of type classes. However, this feature is still not fully implemented in current Haskell systems. Recent advances in generic programming would also improve the implementation [18, 16].

At this moment, we have specified simple imperative, functional, object-oriented and logic programming languages. The specifications have been made in a modular way reusing common components of the different languages.

The original goal of our research was to develop prototypes for the abstract machines underlying the integral object-oriented operating System Oviedo3 [2] whith the aim to test new features as security, concurrency, reflectiveness and distribution [8, 44].

More information on the Language Prototyping System can be obtained at [1].

References


Verifying an Applicative ATP Using Multiset Relations

Francisco J. Martín-Mateos, Jose A. Alonso, María J. Hidalgo, and José L. Ruiz-Reina

Departamento de Ciencias de la Computación e Inteligencia Artificial.
Facultad de Informática y Estadística, Universidad de Sevilla
Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain
{fjesus,jalonso,mjoseh,jruiz}@cica.es

Abstract. We present in this paper a formalization of multiset relations in the ACL2 theorem prover, and we show how multisets can be used to mechanically prove non-trivial termination properties. Every relation on a set induces a relation on finite multisets over that relation is also well-founded. We have carried out a mechanical proof of this property in the ACL2 logic. This allows us to provide well-founded multiset relations in order to prove termination of recursive functions. Once termination is proved, the function definition is admitted as an axiom in the logic and formal mechanized reasoning about it is possible. As a major application of this tool, we show how multisets can be used to prove termination of a tableaux based theorem prover for propositional logic.

Introduction

We present in this paper a formalization of multiset relations in the ACL2 system, and we show how these relations can be used to prove non-trivial termination properties, providing a tool for defining relations on finite multisets and showing that, under certain conditions, these relations are well-founded. Such well-founded relations allows the user to provide a particular multiset measure in order to prove termination of a recursively defined function. Termination proofs are required by ACL2 to admit function definitions as axioms in the logic, as a mean to avoid inconsistencies. Once a function definition is admitted, formal mechanized reasoning about it is possible. We illustrate the use of this tool, presenting the termination proof of a Common Lisp definition of a tableaux based theorem prover for propositional logic. This allows us to verify soundness and completeness of this prover.

ACL2 is a programming language, an applicative subset of Common Lisp. ACL2 is also a logic designed to reason about the programs defined in the language. And, finally, ACL2 is a mechanical theorem proving system, supporting

* This work has been supported by DGES/MEC: Projects TIC2000-1368-C03-02 and PB96-1345
formal reasoning in the logic. The system evolved from the Boyer-Moore theorem prover, also known as Nqthm. For an introduction to ACL2, see the tutorials in the ACL2 web page [6]. To obtain more background on ACL2, see [5].

The ACL2 logic is a quantifier-free, first-order logic with equality, describing an applicative subset of Common Lisp. The syntax of terms is that of Common Lisp (we will assume the reader familiar with this language). The logic includes axioms for propositional logic and for a number of Lisp functions and data types. Rules of inference include those for propositional calculus, equality, instantiation and induction. By the principle of definition, new function definitions (using defun) are admitted as axioms only if there exists a measure function taking values on a well-founded set, in which the arguments of each recursive call decrease, ensuring in this way that no inconsistencies are introduced by new definitions. The primitive well-founded set in the logic is the ordinal $\varepsilon_0$. The theory has a constructive definition of the ordinals up to $\varepsilon_0$, in terms of lists and natural numbers, given by the predicate e0-ordinalp and the order e0-ord-<. For every function definition introduced by the user, ACL2 starts a proof attempt of its termination. In some non trivial cases, the system is not able to prove it by its own and needs help from the user. Thus, it allows the user to provide a particular measure and a well-founded relation.

Multisets provide a powerful way to prove termination in some of these non trivial cases. Multisets are usually defined in an informal way as “sets with repeated elements”. Dershowitz and Manna [3] proved that every well-founded relation on a set $A$ induces a well-founded relation on the set of finite multisets of elements taken from $A$. In the first section of this paper, we present how we have formalized and proved this theorem using ACL2, and stated it in an abstract way. This allows to instantiate the theorem to show well-foundedness of concrete multiset relations. We have also developed a macro defmul in order to easily make definitions of induced multiset relations. Besides defining the multiset relation induced by a given relation, this macro performs a mechanical proof, by functional instantiation, of well-foundedness of the defined multiset relation, provided that the given relation is well-founded.

We illustrate our multiset tool, showing how it is used as part of the verification process of a Common Lisp definition of a tableaux based theorem prover for propositional logic. This prover is defined in the second section. In the third section we show that the use of a well founded multiset relation is specially well suited in the termination proof of that definition, and how our defmul tool can assist in the automation of the proof. Once termination is proved, one can use the ACL2 logic to reason about the prover and mechanically prove its soundness and completeness. This case study is part of our current work on formalizing properties of deduction systems using ACL2.

Due to the lack of space we will skip details of the mechanical proofs. The complete files with definitions and theorems are available on the web in http://www-cs.us.es/~fmartin/acl2-tab-prop/.
1 Formalization of Multiset Relations in ACL2

A multiset $M$ over a set $A$ is a function from $A$ to the set of natural numbers. This is a formal way to define “sets with repeated elements”. Intuitively, $M(x)$ is the number of copies of $x \in A$ in $M$. This multiset is finite if there are finitely many $x$ such that $M(x) > 0$. The set of all finite multisets over $A$ is denoted as $\mathcal{M}(A)$.

Basic operations on multisets are defined to generalize the same operations on sets, taking into account multiple occurrences of elements: $x \in M$ means $M(x) > 0$, $M \subseteq N$ means $M(x) \leq N(x)$, for all $x \in A$, $M \cup N$ is the function $M + N$ and $M \setminus N$ is the function $M \div N$ (where $x \div y$ is $x - y$ if $x \geq y$ and 0 otherwise).

Any ordering defined on a set $A$ induces an ordering on multisets over $A$: given a multiset, a smaller multiset can be obtained by removing a non-empty subset $X$ and adding elements which are smaller than some element in $X$. This construction can be generalized to binary relations in general, not only for partial orderings. This is the formal definition:

**Definition 1.** Given a relation $<$ on a set $A$, the multiset relation induced by $<$ on $\mathcal{M}(A)$, denoted as $<_{mul}$, is defined as $N <_{mul} M$ iff there exist $X, Y \in \mathcal{M}(A)$ such that $\emptyset \neq X \subset M$, $N = (M \setminus X) \cup Y$ and $\forall y \in Y \exists x \in X$, $y < x$. It can be easily shown that if $<$ is a strict ordering, then so is $<_{mul}$. In such case we talk about multiset orderings.

A relation $<$ on a set $A$ is terminating if there is no infinite decreasing sequence $x_0 > x_1 > x_2 \ldots$. An important property of multiset relations on finite multisets is that they are terminating when the original relation is terminating, as stated by the following theorem:

**Theorem 1.** (Dershowitz and Manna, [3]). Let $<$ be a terminating relation on a set $A$, and $<_{mul}$ the multiset relation induced by $<$ on $\mathcal{M}(A)$. Then $<_{mul}$ is terminating.

The above theorem provides a tool for showing termination of recursive function definitions, by using multisets: show that some multiset measure decreases in each recursive call, comparing multisets with respect to the relation induced by a given terminating relation. In the following subsection, we explain how we formalized theorem 1 in the ACL2 logic.

1.1 Formalization of Well-Founded Multiset Relations in ACL2

Let us deal with formalization of terminating relations in ACL2. A restricted notion of terminating relations is built into ACL2 based on the following meta-theorem: a relation $<$ on a set $A$ is terminating iff there exists a function $F$:

---

1 Although not explicitly, we will suppose that the relations given here represent some kind of “smaller than” relation.
A → Ord such that \( x < y \Rightarrow F(x) < F(y) \), where Ord is the class of all ordinals. In this case, we also say that the relation is well-founded. Note that we are denoting the relation on A and the ordering between ordinals using the same symbol \(<\). Thus, an arbitrary well-founded relation \textsf{rel} defined on a set of objects satisfying a property \textsf{mp} (measure property) can be defined in ACL2 as shown below:

(\text{encapsulate}
 ((\textsf{mp} (x) \text{booleanp}) (\textsf{rel} (x y) \text{booleanp}) (\textsf{fn} (x) \text{e0-ordinalp}))
 ...
 (defthm \textsf{rel-well-founded-relation-on-mp}
   (and (implies (\textsf{mp} x) (\text{e0-ordinalp} (\textsf{fn} x)))
    (implies (and (\textsf{mp} x) (\textsf{mp} y) (\textsf{rel} x y))
     (\text{e0-ord-<} (\textsf{fn} x) (\textsf{fn} y))))
 :rule-classes :well-founded-relation))

By the encapsulation mechanism (using \texttt{encapsulate}), the user can introduce new function symbols by axioms constraining them to have certain properties (to ensure consistency, a witness local function having the same properties has to be exhibited). Inside an \texttt{encapsulate}, properties stated with \texttt{defthm} need to be proved for the local witnesses, and outside, those theorems work as assumed axioms. The functions partially defined with \texttt{encapsulate} can be seen as second order variables, representing functions with those properties. A derived rule of inference, functional instantiation, allows some kind of second-order reasoning: theorems about constrained functions can be instantiated with function symbols known to have the same properties.

In this case, we partially define three functions \textsf{mp}, \textsf{fn} and \textsf{rel}, defining a general well-founded relation in ACL2 (dots are used to omit the irrelevant local definitions). The predicate \textsf{mp} recognizes the kind of objects (called \textit{measures}) that are ordered in a well-founded way by \textsf{rel}. The \textit{embedding} function \textsf{fn} is an order-preserving function mapping every measure to an ordinal. Once a relation is proved to satisfy these properties and the theorem is stored as a well-founded relation rule, it can be used in the admissibility test for recursive functions. We call the theorem \textsf{rel-well-founded-relation-on-mp} above the well-foundedness theorem for \textsf{rel}, \textsf{mp} and \textsf{fn}. In ACL2, every particular well-founded relation (except the primitive relation on \(\varepsilon_0\) ordinals) has to be given by means of three functions (a binary relation, a measure predicate and an embedding function), and the proof of the corresponding well-foundedness theorem for such functions.

Let us now deal with formalization of multisets relations. We represent multisets in ACL2 as true lists. Given a predicate \texttt{(mp x)} describing a set \(A\), finite multisets over \(A\) are described by the following function:

(defun \textsf{mp-true-listp} (l)
  (if (\text{atom} l)
    (\text{equal} l \text{nil})
    (and (\textsf{mp} (\text{car} l)) (\textsf{mp-true-listp} (\text{cdr} l))))))
Note that this function depends on the particular definition of the predicate \( mp \). With this representation, different true lists can represent the same multiset: two true lists represent the same multiset iff one is a permutation of the other. Thus, the order in which the elements appear in a list is not relevant, but the number of occurrences of an element is important. This must be taken into account, for example, when defining multiset difference in ACL2 (the function `remove-one`, omitted here, deletes one occurrence of an element from a list, whenever possible):

\[
\text{(defun multiset-diff (m n)}
\]
\[
\text{\quad \quad (if (atom n) m (multiset-diff (remove-one (car n) m) (cdr n))))}
\]

The definition of \( <_{\text{mul}} \) given in the preceding subsection is quite intuitive but, due to its many quantifiers, computationally complex. Instead, we will use a somewhat restricted definition, based on the following theorem:

**Theorem 2.** Let \( < \) be a strict ordering on a set \( A \), and \( M, N \) two finite multisets over \( A \). Then \( N <_{\text{mul}} M \) iff \( M \setminus N \neq \emptyset \) and \( \forall n \in N \setminus M, \exists m \in M \setminus N, \text{such that } n < m \).

From the computational point of view, the main advantage of this alternative definition is that we do not have to search the multisets \( X \) and \( Y \) of the original definition because we can take \( M \setminus N \) and \( N \setminus M \), respectively. It should be remarked that this equivalence is true only when \( < \) is a strict partial ordering. Anyway, this is not a severe restriction. Moreover, well-foundedness of \( <_{\text{mul}} \) also holds when this restricted definition is used, even if the relation \( < \) is not transitive, as we will see. Thus, given a defined (or constrained) binary relation \( \text{rel} \), we define the induced relation on multisets based on this alternative definition:

\[
\text{(defun exists-rel-bigger (x l)}
\]
\[
\text{\quad \quad (cond ((atom l) nil)}
\]
\[
\text{\quad \quad \quad \quad ((rel x (car l))) t)}
\]
\[
\text{\quad \quad \quad \quad (t (exists-rel-bigger x (cdr l))))}
\]

\[
\text{(defun forall-exists-rel-bigger (l m)}
\]
\[
\text{\quad \quad (if (atom l)}
\]
\[
\text{\quad \quad \quad \quad t)}
\]
\[
\text{\quad \quad \quad \quad (and (exists-rel-bigger (car l) m)}
\]
\[
\text{\quad \quad \quad \quad \quad (forall-exists-rel-bigger (cdr l) m))})
\]

\[
\text{(defun mul-rel (n m)}
\]
\[
\text{\quad \quad (let ((m-n (multiset-diff m n))}
\]
\[
\text{\quad \quad \quad \quad \quad (n-m (multiset-diff n m)))}
\]
\[
\text{\quad \quad \quad \quad (and (consp m-n) (forall-exists-rel-bigger n-m m-n))))}
\]

Finally, let us see how we can formalize in the ACL2 logic the theorem 1 above, which states well-foundedness of the relation \( \text{mul-rel} \). As said before, in
order to establish well-foundedness of a relation in ACL2, in addition to the
relation (mul-rel in this case), we have to provide the measure predicate and the
embedding function, and then prove the corresponding well-foundedness theo-
rem. Since mul-rel is intended to be defined on multisets of elements satisfying
mp, then mp-true-listp is the measure predicate in this case. Let us suppose we
have defined a suitable embedding function called map-fn-e0-ord. Then theo-
rem 1 is formalized as follows:

(deffthm multiset-extension-of-rel-well-founded
  (and (implies (mp-true-listp x)
            (e0-ordinalp (map-fn-e0-ord x)))
      (implies (and (mp-true-listp x)
                   (mp-true-listp y)
                   (mul-rel x y))
            (e0-ord-< (map-fn-e0-ord x) (map-fn-e0-ord y))))
:rule-classes :well-founded-relation)

The command defthm starts a proof attempt in ACL2. The theorem prover
is automatic in the sense that once defthm is invoked, the user can no longer
interact with the system. However, the user can guide the prover by adding
previous lemmas and definitions, in order to carry out a formal proof based on a
preconceived hand proof. In the following, we present a suitable definition for the
embedding function map-fn-e0-ord and the proof sketch we followed to obtain
a mechanical proof of the above theorem.

1.2 A Proof of Well-Foundedness of the Multiset Relation

In the literature (for example [3]) Theorem 1 is usually proved using König’s
lemma: every infinite and finitely branched tree has an infinite path. Neverthe-
less, we have to find a different proof (and more constructive) in ACL2, defining
an order-preserving embedding function map-fn-e0-ord from mp-true-listp
objects to e0-ordinalp objects. Thus, our proof is based on the following re-
result from ordinal theory: given an ordinal \( \gamma \), the set \( \mathcal{M}(\gamma) \) of finite multisets of
elements of \( \gamma \), ordered by the multiset relation induced by the order between
ordinals, is order-isomorphic to the ordinal \( \omega^\gamma \) and the isomorphism is given by
the function \( H \) where \( H(\{\beta_1, \ldots, \beta_n\}) = \omega^{\beta_1} + \ldots + \omega^{\beta_n} \). This result can be
proved using Cantor’s normal form of ordinals and its properties.

The isomorphism \( H \) above suggests the following definition of the embedding
function map-fn-e0-ord: given a multiset of elements satisfying mp, apply fn
to every element to obtain a multiset of ordinals. Then apply \( H \) to obtain an
ordinal less than \( \varepsilon_0 \). If ordinals are represented in ACL2 notation (see [5]), then
the function \( H \) can be easily defined, provided that the function fn returns
always a non-zero ordinal: the function \( H \) simply has to sort the ordinals in
the multiset and add 0 as the final cdr. These considerations lead us to the
following definition of the embedding function map-fn-e0-ord. Note that the
non-zero restriction on fn is easily overcome, defining (the macro) fn1 equal
to \( \text{fn} \) except for integers, where 1 is added. In this way \( \text{fn1} \) returns non-zero ordinals for every measure object and it is order-preserving if and only if \( \text{fn} \) is.

\[
(\text{defun insert-e0-ord-<} \ (x \ l) \\
(\text{cond} \ ((\text{atom} \ l) \ (\text{cons} \ x \ l)) \ ((\text{not} \ ((\text{e0-ord-<} \ x \ (\text{car} \ l))) \ (\text{cons} \ x \ l)) \ ((\text{t} \ (\text{cons} \ (\text{car} \ l) \ (\text{insert-e0-ord-<} \ x \ (\text{cdr} \ l))))))))
\]

\[
(\text{defun add1-if-integer} \ (x) \ (\text{if} \ (\text{integerp} \ x) \ (1+ \ x) \ x))
\]

\[
(\text{defmacro} \ \text{fn1} \ (x) \ \text{'(add1-if-integer} \ (\text{fn} ,x)))
\]

\[
(\text{defun map-fn-e0-ord} \ (l) \\
(\text{if} \ (\text{consp} \ l) \ (\text{insert-e0-ord-<} \ (\text{fn1} \ (\text{car} \ l)) \ (\text{map-fn-e0-ord} \ (\text{cdr} \ l))) \ 0))
\]

Once \( \text{map-fn-e0-ord} \) has been defined, let us now deal with the ACL2 mechanical proof of the well-foundedness theorem for \( \text{mul-rel} \), \( \text{mp-true-listp} \) and \( \text{map-fn-e0-ord} \) as stated at the end of subsection 1.1. The first part of the theorem, which establishes that \( \text{map-fn-e0-ord} \ x \) is an ordinal when \( \text{mp-true-listp} \ x \), it is not difficult, and can be proved in ACL2 with minor help form the user. The hard part of the theorem is to show that \( \text{map-fn-e0-ord} \) is order-preserving. Here is an informal proof sketch:

**Proof sketch:**

Let us denote, for simplicity, the functions \( \text{fn1} \) and \( \text{map-fn-e0-ord} \), as \( f \) and \( f_{\text{mul}} \), and the relation \( \text{rel} \), \( \text{mul-rel} \) and \( \text{e0-ord-<} \) as \( \text{<rel} \), \( \text{<mul} \) and \( \text{<} \), respectively. Let \( M \) and \( N \) be two multisets of \( \text{mp} \) elements such that \( N \text{<mul} M \). We have to prove that \( f_{\text{mul}}(N) \text{<mul} f_{\text{mul}}(M) \). We can apply induction on the number of elements of \( N \). Note that \( M \) can not be empty, and if \( N \) is empty the result trivially holds. So let us suppose that \( M \) and \( N \) are not empty. Let \( f(x) \), \( f(y) \) be the biggest elements of \( f[N] \) and \( f[M] \), respectively. Note that \( f(x) \) and \( f(y) \) are the \text{car} elements of \( f_{\text{mul}}(N) \) and \( f_{\text{mul}}(M) \), respectively. Since \( f(x) \) and \( f(y) \) are ordinals, three cases may arise:

1. \( f(x) \text{<} f(y) \). Then, by definition of \( \text{<} \), we have \( f_{\text{mul}}(N) \text{<mul} f_{\text{mul}}(M) \).
2. \( f(x) \text{>} f(y) \). This is not possible: in that case \( x \) is in \( N \setminus M \) and by the multiset relation definition, exists \( z \) in \( M \setminus N \) such that \( x \text{<rel} z \). Consequently \( f(z) \text{>} f(x) \text{>} f(y) \). This contradicts the fact that \( f(y) \) is the biggest element of \( f[M] \).
3. \( f(x) = f(y) \). In that case, \( x \in M \), since otherwise it would exist \( z \in M \setminus N \) such that \( x \text{<rel} z \) and the same contradiction as in the previous case appears. Let \( M' = M \setminus \{x\} \) and \( N' = N \setminus \{x\} \). We have \( N' \text{<mul} M' \) and, in addition, \( f_{\text{mul}}(N') \) and \( f_{\text{mul}}(M') \) are the \text{cdr} of \( f_{\text{mul}}(N) \) and \( f_{\text{mul}}(M) \), respectively. Induction hypothesis can be applied here to conclude that \( f_{\text{mul}}(N') \text{<mul} f_{\text{mul}}(M') \) and therefore \( f_{\text{mul}}(N) \text{<mul} f_{\text{mul}}(M) \). \( \Box \)
We carried out this proof in ACL2. The proof effort was not trivial: lemmas to handle each of the cases generated by the above induction scheme have to be proved, obtaining a mechanical proof very close to the previous proof sketch. See the book `multiset.lisp` in the web page for details about the mechanical proof.

Well-foundedness of `mul-rel` has been proved in an abstract framework, without assuming any particular properties of `rel`, `mp` and `fn`, except those concerning well-foundedness. This allows us to functionally instantiate the theorem in order to establish well-foundedness of the multiset relation induced by any given well-founded ACL2 relation. We defined a macro `defmul` in order to mechanize this process of functional instantiation, providing a convenient way to define the multiset relation induced by a given well-founded relation and to declare the corresponding well-founded relation rule. The following section describes the `defmul` macro.

### 1.3 The `defmul` Macro

Let us suppose we have a previously defined relation `my-rel`, which is known to be well-founded on a set of objects satisfying the measure property `my-mp` and justified by the embedding function `my-fn`. That is to say, the following theorem, using variables `x` and `y`, has been proved (and stored as a well-founded relation rule):

```
(defthm theorem-name
  (and (implies (my-mp x) (e0-ordinalp (my-fn x)))
   (implies (and (my-mp x) (my-mp y) (my-rel x y))
      (e0-ord-< (my-fn x) (my-fn y))))
:rule-classes :well-founded-relation))
```

In order to define the (well-founded) multiset relation induced by `my-rel`, we simply write the following macro call:

```
(defmul (my-rel theorem-name my-mp my-fn x y))
```

The expansion of this macro generate a number of ACL2 events. After the above call to `defmul`, the function `mul-my-rel` is defined as a well-founded relation on multisets of elements satisfying the property `my-mp`, induced by the well-founded relation `my-rel`, and a proof of the corresponding well-foundedness theorem is carried out, without assistance from the user. From this moment on, `mul-my-rel` can be used in the admissibility test for recursive functions to show that the recursion terminates.

### 2 An Applicative ATP for Propositional Logic

We illustrate the use of the `defmul` tool with a case study: the formal verification of an applicative Common Lisp definition of a tableaux based theorem prover for propositional logic. In this section, we present an ACL2 function implementing
the prover; as we will see, termination of this function is not trivial. In the next section we sketch a termination proof in ACL2 using well-founded multisets relations. To build the theorem prover, we closely follow the approach given by M. Fitting in [4].

2.1 Formalization of Propositional Logic and Uniform Notation

We explain now how we have represented propositional formulas in ACL2. Any ACL2 symbol (recognized by the ACL2 function symbolp) will represent a propositional symbol. We represent propositional formulas in prefix notation, using lists. The propositional connectives considered are the usual: negation (~), conjunction (\&), disjunction (v), implication (\rightarrow) and equivalence (\leftrightarrow). If a list represents a propositional formula, its first element is a logic connective, and the rest are the arguments. The following function propositional-p recognizes those ACL2 objects representing propositional formulas. The functions arg1 and arg2 obtain, respectively, the first and the second argument of a formula, if they exist. There are three kinds of propositional formulas: atomic, monary and binary formulas. The functions atomic-p, monary-p and binary-p to identify these formulas. We omit here all these auxiliary functions.

(defun propositional-p (x)
  (cond ((monary-p x) (propositional-p (arg1 x)))
        ((binary-p x) (and (propositional-p (arg1 x))
            (propositional-p (arg2 x))))
        (t (atomic-p x))))

Notwithstanding, we will adopt the uniform notation approach (see [4]) to deal with the recursive structure of propositional formulas. We classify propositional formulas with the form (X o Y) and ~(X o Y) in two categories: those having a conjunctive behaviour, called \(\alpha\)-formulas, and those having a disjunctive behaviour, called \(\beta\)-formulas. Each \(\alpha\)-formula and \(\beta\)-formula has two components, \(\alpha_1\) and \(\alpha_2\) for the \(\alpha\)-formulas and, \(\beta_1\) and \(\beta_2\) for the \(\beta\)-formulas. The classification and components are given in the following tables:

<table>
<thead>
<tr>
<th>(\alpha)</th>
<th>(\alpha_1)</th>
<th>(\alpha_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &amp; Y</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>~(X &amp; Y)</td>
<td>~X</td>
<td>~Y</td>
</tr>
<tr>
<td>~(X \rightarrow Y)</td>
<td>X ~Y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\beta_1)</th>
<th>(\beta_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X \lor Y</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>~(X &amp; Y)</td>
<td>~X</td>
<td>~Y</td>
</tr>
<tr>
<td>X \rightarrow Y</td>
<td>~X</td>
<td>Y</td>
</tr>
<tr>
<td>X \leftrightarrow Y</td>
<td>X &amp; Y ~X &amp; ~Y</td>
<td></td>
</tr>
<tr>
<td>~(X \leftrightarrow Y)</td>
<td>X &amp; ~Y ~X &amp; Y</td>
<td></td>
</tr>
</tbody>
</table>

We define the functions alpha-formula and beta-formula in order to distinguish these two kinds of formulas. To access to their components, we define the functions component-1 (to obtain \(\alpha_1\) or \(\beta_1\)) and component-2 (to obtain \(\alpha_2\) and \(\beta_2\)). There are also formulas neither conjunctive nor disjunctive: the double negations and the literals. We define the functions double-negation and literal-p to recognize them. The component of a double negation \(~\neg Y\),
is the formula \( Y \). We define the function \texttt{component-double-neg} to build the component of a double negation. We omit all these definitions here.

The following theorem is a key lemma, needed to classify propositional formulas using the uniform notation. This result gives a new perspective of the concept \texttt{propositional-p} and then, a new way of defining functions by recursion on formulas:

\[
\text{(defthm uniform-definition-of-propositional-p)}
\begin{align*}
\text{(iff (propositional-p F)} & \text{ (or (alpha-formula F))} \\
& \text{ (beta-formula F))} \\
& \text{ (double-negation F))} \\
& \text{ (literal-p p F)))}
\end{align*}
\]

### 2.2 Semantic Tableau as Rules of Transformation

The semantic tableau method is a refutation system. To prove the validity of a formula \( X \), we start with \( \neg X \) until we eventually generate a contradiction. From a constructive point of view, the method works with a set of formulas and tries to build a model of that set. If it is not possible to build a model for the formula \( \neg X \), then \( X \) is valid.

Given a finite tree \( T \), with its nodes labeled with propositional formulas, the method of semantic tableau selects a branch \( \theta \) and a non-literal formula \( X \) in \( \theta \). If \( X \) is \( \neg \neg Y \), then the branch \( \theta \) is extended adding a new node labeled with \( Y \). If \( X \) is an \( \alpha \)-formula, then the branch \( \theta \) is extended adding two nodes labeled with the components \( \alpha_1 \) and \( \alpha_2 \) of the original formula. If \( X \) is a \( \beta \)-formula, then the branch \( \theta \) is extended adding two branches at the end, each of them with a node labeled with the components \( \beta_1 \) and \( \beta_2 \) of the selected formula. If we denote the result as \( T^* \), we say that \( T^* \) is obtained from \( T \) using a tableau expansion rule. If a branch does not have non-literal formulas, we can not apply the above process. If a branch have two complementary formulas we say that the branch is \textit{closed}.

A tableau for a set of formulas, \( \{A_1, \ldots, A_n\} \), is the one branch tree with \( n \) nodes labeled with \( A_1, \ldots, A_n \), or any tree \( T^* \) obtained from a tableau for the set of formulas \( \{A_1, \ldots, A_n\} \), using a tableau expansion rule. It can be proved that a propositional formula \( X \) is valid if and only if there exists a tableau for \( \{\neg X\} \) with all its branches closed.

To define a function in ACL2 implementing the semantic tableau method, we have to decide how to represent a tableau. This decision can affect on how the function is defined later. A tableau can be seen as a list of branches, and a branch as a list of formulas. In this way, the function that implements the semantic tableau method has to work recursively; that is, it takes a branch, apply it a tableau expansion rule and replace the original for the new one. If the branch considered is closed, it will be discarded and another branch will be analyzed.
This is a recursive process that works on branches: we begin with a branch $\theta$. If $\theta$ is closed, we have finished the process successfully. Otherwise, a non-literal formula $X$ is selected from $\theta$. If $X$ is $\neg\neg Y$, $Y$ is added to $\theta$ and the process will be applied again to it. If $X$ is an $\alpha$-formula, the components $\alpha_1$ and $\alpha_2$ will be added to $\theta$ and the process will be applied again to it. If $X$ is a $\beta$-formula, two new branches are built: $\theta_1$ adding the component $\beta_1$ to $\theta$ and $\theta_2$ adding the component $\beta_2$ to $\theta$. The process is applied to $\theta_1$, and, if it succeeds, it will be applied to $\theta_2$. If any of them does not succeed, the process on the original branch $\theta$ does not succeed.

In the process described above a non-literal and non-expanded formula must be chosen every time an expansion rule is to be applied. If the formula chosen has been expanded before, then the new branch generated by the process will have repetitions. To avoid this we can mark the expanded formulas or we can eliminate them. We use the second option to simplify the function definition: this is possible because in the tableau method for propositional logic the formulas are used only once.

Therefore, we can define the function associated with the method of semantic tableau, as a function that works with a list of formulas. This function builds new lists from the initial list of formulas, and recursively applies the same process to them. Thus, it can be seen as a transformation system, specified by a set of rules acting on a set of formulas. This kind of rule-based point of view is common to others provers based on transformations acting on set of formulas. The rules used in this case are the following:

1. Double negation rule:
\[
\{F_1, \ldots, F_{i-1}, \neg\neg G, F_{i+1}, \ldots, F_n\} \xrightarrow{st} \{F_1, \ldots, F_{i-1}, G, F_{i+1}, \ldots, F_n\}
\]
2. $\alpha$-formula rule:
\[
\{F_1, \ldots, F_{i-1}, \alpha, F_{i+1}, \ldots, F_n\} \xrightarrow{st} \{F_1, \ldots, F_{i-1}, \alpha_1, \alpha_2, F_{i+1}, \ldots, F_n\}
\]
3. $\beta$-formula rule:
\[
\{F_1, \ldots, F_{i-1}, \beta, F_{i+1}, \ldots, F_n\} \xrightarrow{st} \{F_1, \ldots, F_{i-1}, \beta_1, F_{i+1}, \ldots, F_n\}
\]
\[
\{F_1, \ldots, F_{i-1}, \beta, F_{i+1}, \ldots, F_n\} \xrightarrow{st} \{F_1, \ldots, F_{i-1}, \beta_2, F_{i+1}, \ldots, F_n\}
\]

### 2.3 ACL2 Definition of a Semantic Tableau Prover

Based on the above considerations, our ACL2 implementation of a tableau based theorem prover receives as argument a list of formulas that represents a branch of the tableau. If this branch is not closed, a selection function chooses a non-literal formula. The tableau expansion rules are applied to this formula, generating new branches. The function is recursively applied to these new branches. With this idea, we define the ACL2 function `closed-tableau`, implementing the tableau method for propositional logic:
(defun closed-tableau (S)
  (declare (xargs :mode :program))
  (cond ((endp S) nil)
        ((closed S) t)
        (t (let ((F (selection S)))
            (cond ((double-negation F)
                   (closed-tableau (add (component-double-neg F)
                                       (remove-one F S))))
            ((alpha-formula F)
             (closed-tableau (add (component-1 F)
                                  (add (component-2 F)
                                       (remove-one F S))))
            ((beta-formula F)
             (and (closed-tableau (add (component-1 F)
                                      (remove-one F S)))
                  (closed-tableau (add (component-2 F)
                                      (remove-one F S))))
            (t nil))))))

Several remarks are due about this definition. First, note that to implement
the control of this process we need a selection function, which determines the
chosen formula and, consequently, the expansion rule to apply. For this purpose,
we consider a function named selection (definition omitted) that receives a list
of formulas as an argument and returns the first non-literal formula from that
list, whenever there exists such formula. Nevertheless, we could have used any
function with the following properties:

1. If the argument of selection is a list with some non-literal formula, then
the function returns a non-literal formula from that list.
2. If the argument of selection is a list of literal formulas, then the function
returns nil.

This function can be executed on every compliant Common Lisp implement-
tation. For example, we have checked the validity of some Urquhart formulas
obtaining the following time results:

<table>
<thead>
<tr>
<th>N</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (msec)</td>
<td>130</td>
<td>840</td>
<td>5270</td>
<td>29380</td>
<td>156200</td>
</tr>
</tbody>
</table>

One of the base cases of this recursive function appears when the branch has
two complementary formulas. In such case we recognize the branch as closed. The
function closed, omitted here, checks if a list has two complementary formulas.
For the recursive calls, we have to build new branches by replacing the non-
literal formula chosen with its components. We define the function add to add
one formula to a branch avoiding repetitions.
3 Termination of closed-tableau

The definition of closed-tableau is not admitted immediately as an axiom in the ACL2 logic, since the default heuristics of the prover are not able to prove its termination. The termination proof of this function is not trivial: note the different behaviour of the recursive calls for $\alpha$-formulas and $\beta$-formulas; in particular, the $\alpha$ expansion rule obtains a larger set of formulas.

The declaration (xargs :mode :program) forces ACL2 to accept this definition without proving its termination. A function definition in :program mode is not included as an axiom of the logic (and therefore reasoning about it is not possible) until its termination is proved. Thus, a suitable measure and well-founded relation has still to be explicitly given to the prover. We will use a multiset relation for that purpose, as we explain now.

We can define a measure on formulas, related to the uniform notation, ensuring that the measure of the components of an $\alpha$-formula, $\beta$-formula or double negation, are smaller than the measure of the original formula. We extend the measure given on $\Pi$ to include equivalences:

**Definition 2.** The uniform measure of a propositional formula $X$ is given by the function $\mu$:

1. If $X$ is atomic, $\mu(X) = 0$
2. If $X = \neg Y$, $\mu(X) = 1 + \mu(Y)$
3. If $X = Y_1 \circ Y_2$, with $\circ$ distinct of equivalence, $\mu(X) = 2 + \mu(Y_1) + \mu(Y_2)$
4. If $X = Y_1 \leftrightarrow Y_2$, $\mu(X) = 5 + \mu(Y_1) + \mu(Y_2)$

We can easily implement the measure $\mu$ in ACL2, defining a function uniform-measure, omitted here. The main property of this uniform measure is that it decreases on the components of compound formulas. For example, the property for $\alpha$-formulas is showed below (analogous properties for double negation and beta formulas are established):

```lisp
(defun branch-uniform-measure (branch)
  (cond ((endp branch) nil)
        (t (cons (uniform-measure (car branch))
                  (branch-uniform-measure (cdr branch))))))
```

Now we can define a suitable measure for the termination of the function closed-tableau. Recall that the argument of this function is a list of formulas, representing a branch of a tableau. The idea is to measure this argument by the list of the uniform measures of each of its formulas. The following function defines this measure:

```lisp
(defun branch-uniform-measure (branch)
  (cond ((endp branch) nil)
        (t (cons (uniform-measure (car branch))
                  (branch-uniform-measure (cdr branch))))))
```
Note that this measure can be seen as a multiset of ordinals (natural numbers). Thus, the multiset relation induced by $e_0$-$ord$-$<$ on multisets of ordinals is a well-founded relation that can be used as the well-founded relation needed to justify termination of $\text{closed-tableau}$. We simply make this defmul call to define in ACL2 the intended multiset well-founded relation:

\[
\text{(defmul (e0-ord-< nil e0-ordinalp e0-ord-<-fn nil nil))}
\]

After this defmul call, the function $\text{mul-e0-ord-<}$ is automatically defined and proved to be well-founded over multisets of ordinals. We can now verify the termination of the function $\text{closed-tableau}$, providing the measure of the arguments and the well-founded relation:

\[
\text{(verify-termination closed-tableau (declare (xargs :measure (branch-uniform-measure S) :well-founded-relation mul-e0-ord-<)))}
\]

This call to verify-termination generates a proof attempt to show that the measure $\text{branch-uniform-measure}$ decreases (w.r.t the multiset relation $\text{mul-e0-ord-<}$) in every recursive call of the function $\text{closed-tableau}$. With the help of some previous lemmas, this proof can be successfully completed in ACL2 (see the web page for details) and the function definition is admitted as an axiom in the logic. This allows formal reasoning about it.

For example, we can define a function to check the validity of a formula, calling the function $\text{closed-tableau}$ on the list built with the negation of the original formula:

\[
\text{(defun tableau-valid-p (F) (closed-tableau (list (negation F))))}
\]

A formal verification of this function is now possible. For example, we can prove in ACL2 the soundness and completeness theorem (see [4]) for this tableau based theorem prover, following the lines of a previous verification work of Boyer and Moore [2], where a tautology checker based on binary decision diagrams is formally verified using Nqthm. Nevertheless, we do not discuss this issue here, since we are concentrating on termination aspects and how multiset relations can help in the task of proving it.

### 4 Conclusions

We have presented a formalization of multiset relations in ACL2, showing how they can be used as a tool for proving non-trivial termination properties of recursive functions in ACL2. We have defined the multiset relation induced by a given relation and proved a theorem establishing well-foundedness of the multiset relation induced by a well-founded relation. This theorem is formulated in an abstract way, so that functional instantiation can be used to prove well-foundedness of concrete multiset relations. We also presented a macro named
defmul, implemented to provide a convenient tool to define these concrete multiset well-founded relation.

We initially presented this tool in [8], where we successfully used it to prove several non-trivial termination properties: a tail-recursive version of Ackermann’s function, a definition of McCarthy’s 91 function and a proof of Newman’s lemma for abstract reductions. In this paper we present how this tool can be applied to prove termination of an applicative Common Lisp definition of a tableau-based theorem prover. Proving termination allows us to formally verify the intended properties of the function, namely its soundness and completeness. One interesting aspect of ACL2 is that the functions verified are defined in an applicative subset of Common Lisp, and (under some conditions) they can be executed in any interpreter of that language.

Proving theorems in ACL2 is not a trivial task. A typical proof effort consists of formalizing the problem, and guiding the prover to a preconceived hand proof, by decomposing the proof into intermediate lemmas. If one lemma is not proved in a first attempt, then additional lemmas are often needed, as suggested by inspecting the failed proof. See the web page for a detailed description of the proofs presented in this paper.

The work presented in the second section is part of the ambitious project of providing a mechanically verified set of automated reasoning algorithms for some logics. We have begun with propositional logic and a well-known automated theorem proving technique, semantic tableau. We have seen that the multiset tool plays an unexpected role in the termination proof. This work can be extended to others ATP’s for this logic, others logics (first order, equational [7], modal, ...) and applications based on these logics; we think that the multiset tool will be important to develop this project.

References

Adding Traces to a Lazy Monadic Evaluator

Cristóbal Pareja, Ricardo Peña, Fernando Rubio, and Clara Segura
Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain
{cpareja,ricardo,fernando,csegura}@sip.ucm.es

Abstract. The debugging of lazy functional programs is a non yet satisfactorily solved problem. Different approaches have been proposed during the last years, all of them having a property in common: The graph produced by the traced program is different from the original graph, i.e. the one without traces. In this paper we propose a cleaner and more modular approach to the trace problem. We regard traces as observations of the program and at the same time we want to preserve the original graph. In this way, a clean separation between the trace and the program being observed is established. Consequently, there may be variables in the trace referencing parts of the graph (i.e. pointers from the trace to the graph), but not the other way around. By doing so the correctness is guaranteed, as the normal execution process is not altered. In order to reach this goal, a monadic approach is followed. The success of the approach is shown by simulating three up-to-date Haskell tracers.

1 Introduction

The debugging of lazy functional programs is a non yet satisfactorily solved problem (e.g. see [17]). In recent years there have been several proposals for incorporating execution traces to lazy functional languages. In [3], an extensive comparison of three of these systems can be found, namely Freja [9, 8], the Redex Trail System (RTS) [12, 13, 14] and the Haskell Object Observation Debugger (Hood) [4]. They have been incorporated to different Haskell [6] compilers. Freja is a question-answer system that directs the programmer to the cause of an incorrect value. RTS allows the user to travel backwards from a value along the redex history leading to it, and it is incorporated to the nhc98 compiler. In Hood, the programmer first instruments the program marking the variables he wants to observe and then the system produces a printing of their final value. Final value does not necessarily mean normal form, but evaluation to the degree required by the computation. Hood can currently be used with the Glasgow Haskell Compiler (GHC) [11], Hugs98 and also with nhc98.

Their implementation follows different strategies such as modifying the abstract machine or transforming the source program, but all of them have a property in common: The graph produced by the traced program is different from the original graph, i.e. the one without traces.

1 http://www.cs.york.ac.uk/fp/nhc98
2 http://www.haskell.org/hugs
In this paper we propose a cleaner and more modular approach to the trace problem. We regard traces as observations of the program and at the same time we want to preserve the original graph. In this way, we establish a clean separation between the trace and the program being observed. Consequently, there may be variables in the trace referencing parts of the graph (i.e. pointers from the trace to the graph), but not the other way around. Moreover, we would like to experiment with different trace systems without modifying the normal evaluation. In order to reach this goal, a monadic approach is followed: First a normal evaluator for a lazy language is defined; then it is trivially converted into a monadic one following well known patterns (e.g. see [16]). A state transformer monad is used for incorporating traces. The trace is kept in the hidden state while the normal evaluation proceeds in the visible part. When the evaluation finishes, a browser can access the state and print or consult the trace (see Fig. 1).

We define three different monads, one for every of the above mentioned systems: Freja, RTS and Hood. Due to our restriction of not allowing references from the program to the trace, a few limitations arise with respect to these systems.

The evaluator is written in Haskell and implements a simple version of Sestoft’s lazy abstract machine [12]. This has been chosen in order to keep manageable the explanation. However the basic elements such as the heap, the stack, and the updating and sharing of nodes are already present in this machine. The same approach could be used for evaluators based on more complex machines.

2 A Lazy Evaluator Based on Sestoft’s Abstract Machine

2.1 The Language

The language is an enriched $\lambda$-calculus with recursive let, (saturated) constructor applications and case expressions. It is shown in Fig. 2 where $x_i$ denotes a list of variables. The case expressions are used to force evaluation to weak head normal form, while the let ones are used to create new closures. To ensure sharing, the argument of an application is always a variable.
2.2 A Non-monadic Evaluator

We present an interpreter, written in Haskell, for Sestoft's machine Mark I. For the sake of clarity, we use some extra-notation in the presentation of the algorithms. In this simple machine there is no environment binding the free variables, so β-reduction is accomplished by substituting variables for variables in the body of a λ-abstraction. A configuration in this machine consists of a heap, a control expression and a stack. The stack contains three different objects: Arguments of pending applications, alternatives of pending pattern matchings, and update markers of pending updates. Weak-head normal forms are λ-abstractions and constructor applications. If a weak-head normal form is reached with an empty stack, the machine stops. Otherwise, the continuation is looked for in the stack. Function eval' is the responsible of detecting when to stop, while function step just performs a reduction using the rules of Sestoft’s machine:

\[
\text{type Config} = (\text{Heap}, \text{Exp}, \text{Stack})
\]

\[
eval :: \text{Exp} \to (\text{Heap}, \text{Exp})
eval e = eval' (\{\}, e, [\])
\]

\[
eval' :: \text{Config} \to (\text{Heap}, \text{Exp})
eval' (\Gamma, \lambda x. e, []) = (\Gamma, \lambda x. e)
eval' (\Gamma, C \bar{y}_i, []) = (\Gamma, C \bar{y}_i)
eval' e = (eval' \cdot \text{step}) e
\]

\[
\text{step} :: \text{Config} \to \text{Config}
\]

\[
\text{Rules for variables}
\]

\[
\text{step} (\Gamma \cup [y \mapsto \lambda x. e], y, s) = (\Gamma \cup [y \mapsto \lambda x. e], \lambda x. e, s)
\]

\[
\text{step} (\Gamma \cup [y \mapsto C \bar{y}_i], y, s) = (\Gamma \cup [y \mapsto C \bar{y}_i], C \bar{y}_i, s)
\]

\[
\text{step} (\Gamma \cup [y \mapsto e], y, s) = (\Gamma, e, \#y : s)
\]

\[
\text{where } e \neq \lambda x. e' \wedge e \neq C \bar{y}_i
\]

\[
\text{Rule for let}
\]

\[
\text{step} (\Gamma, \text{let } x_i = e_i \text{ in } e, s) = (\Gamma \cup [y_i \mapsto e_i | y_j / x_j], e | y_j / x_j, s)
\]

\[
\text{where } \bar{y}_i \text{ are fresh variables}
\]
{Rules for application}
\[ \text{step}\ (\Gamma, e, y : s) = (\Gamma, e, y : s) \]
\[ \text{step}\ (\Gamma, \lambda x. e, y : s) = (\Gamma, e [y/x], s) \]

{Rules for case}
\[ \text{step}\ (\Gamma, \text{case } e \text{ of alts}, s) = (\Gamma, e, \text{alts : s}) \]
\[ \text{step}\ (\Gamma, C_k \ y_i, C_j \ x_{ji} \rightarrow e_j : s) = (\Gamma, e_k \ [y_i/x_{ki}], s) \]

{Rules for updates}
\[ \text{step}\ (\Gamma, \lambda x. e, \#y : s) = (\Gamma \cup [y \mapsto \lambda x. e], \lambda x. e, s) \]
\[ \text{step}\ (\Gamma, C_k \ y_i, \#y : s) = (\Gamma \cup [y \mapsto C_k \ y_i], C_k \ y_i, s) \]

Sestoft proves some interesting properties for this machine. The most important for us is the fact that all free variables in the control expression or in the heap expressions are pointers (i.e. they are not program variables), and all bound variables (either let-bound, lambda-bound or bound in a case alternative) are program variables. Additionally, all pointers belong to \( \text{dom } \Gamma \) (i.e. they are defined in the heap) and all program variables are different, provided that all bound variables are different in the initial expression. In the rules above, pointers are denoted by \( y \), and program variables are denoted by \( x \). We will consistently use this convention along the paper.

2.3 A Monadic Evaluator

Our monadic evaluators make use of the state transformer monad (see e.g. [11, Chapter 10]). The visible part of the monad returns the final heap and the final control expression. The hidden part —i.e. the state— stores the trace of the computation. By defining different states and the specific functions modifying the state, different trace systems can be constructed. The specific functions to be defined are \( \text{start} \), which computes the initial state, and \( \text{change} \) which computes the state change at every transition of the machine. Notice that the translation from \( \text{eval}' \) to \( \text{evalM}' \), and from \( \text{step} \) to \( \text{stepM} \) are trivial.

```haskell
data ST s a = ST (s → (s,a))
instance Monad (ST s) where ...

start :: Exp → State
run :: Exp → (Exp → State) → (State, Heap, Exp)
run e₀ start = (s_f, h, e_f)
    where s₀ = start e₀
          ST f = evalM e₀
          (s_f, (h, e_f)) = f s₀```


evalM :: Exp → ST s (Heap, Exp)

\[
evalM e = evalM' (\{ \}, e, [])
\]

evalM' :: Config → ST s (Heap, Exp)

\[
evalM' (\Gamma, \lambda x.e, []) = return (\Gamma, \lambda x.e)
\]

\[
evalM' (\Gamma, C y_i, []) = return (\Gamma, C y_i)
\]

\[
evalM' c = \text{do } c' ← \text{stepM } c
\]

\[
evalM' c'
\]

\[
\text{stepM :: Config → ST s Config}
\]

\[
\text{stepM } c = \text{ST } (\lambda s. \text{let } c' = \text{step } c
\]

\[
s' = \text{change } c \; c' \; s\]

\[
\text{in } (s', c'))
\]

where the function \text{stepM} reflects the clean separation between the normal abstract machine transition represented by \text{step}, and the additional information we want to produce, that is, the building of a trace. Function \text{change} is responsible for carrying out such a trace construction. In general, it may depend on both the configuration \(c\) previous to the transition and the configuration \(c'\) after it. However, in most of the cases, it only depends on the previous one \(c\). In the trivial case where no traces are desired, the \text{change} function is just the identity function with respect to the state:

\[
\text{type State } = ()
\]

\[
\text{start } e_o = ()
\]

\[
\text{change :: Config → Config → State → State}
\]

\[
\text{change } c \; c' \; s = s
\]

3 Specific Functions for RTS Traces

An RTS tracer computes a trace allowing the user to ‘travel’ backwards from the final expression to the original one by following the trail of redexes. Moreover, when an expression has several subexpressions, the user is offered the possibility of knowing the normal form of each subexpression and of following its redex trail in order to detect an erroneous reduction. With this in mind, we consider an RTS state to consist of a list of traces representing the redex history of the current control expression. Each trace points to an expression in this redex history, so avoiding the garbage collector to consider it as garbage. If the expression is not a normal form, it will have subexpressions which can be independently reduced. In our language, the only ones having this property are applications \(e \; y\), and expressions \text{case } e \text{ of } \text{alts}. In both cases there is a single subexpression \(e\). So, the traces for applications and \text{case} expressions record also the redex trail of its subexpression as a list of traces. The state has also a second component which can be seen as an emulation of Sestoft’s machine stack. Each time an argument or a set of alternatives is stored in machine’s stack, the current expression and its redex trail are stored in the state’s stack. A new history is started for the
subexpression. When this one reaches its normal form, the history of its parent expression is recovered from the state stack and it is re-started again. For our purposes, variables and *let* expressions are not traced. They are considered just as intermediate expressions leading, in the first case to the corresponding expression bound for the variable in the heap and, in the second case, to the main expression of the *let*. The following declarations implement the above ideas:

\[
\text{data RTSTrace } a = \text{Root } a \\
\mid \text{Node } a \ [\text{RTSTrace } a] \\
\mid \text{NF } a
\]

\[
\text{type RTSState } = ([\text{RTSTrace Exp}], [(\text{Exp}, [\text{RTSTrace Exp}]]))
\]

\[
\begin{align*}
\text{start } e &= ([\text{Root } e], []) \\
\text{change } (\Gamma, y, s) c' (ts, ets) &= (ts, ets) \\
\text{change } (\Gamma, \text{let } x_1 = e_1 \text{ in } e, s) c' (ts, ets) &= (ts, ets) \\
\text{change } (\Gamma, e_0 @(e y), s) c' (ts, ets) &= ([\text{Root } e], (e_0, ts) : ets) \\
\text{change } (\Gamma, e_0 @(\lambda x.e), y : s) c' (ts, (e_1, ts_1) : ets) &= (\text{Node } e_1 (\text{NF } e_0 : ts) : ts_1, ets) \\
\text{change } (\Gamma, \lambda x.e, \# y : s) c' (ts, ets) &= (ts, ets) \\
\text{change } (\Gamma, e_0 @(\text{case } e \text{ of } alts), s) c' (ts, ets) &= ([\text{Root } e], (e_0, ts) : ets) \\
\text{change } (\Gamma, e_0 @(C_k y) i, alts : s) c' (ts, (e_1, ts_1) : ets) &= (\text{Node } e_1 (\text{NF } e_0 : ts) : ts_1, ets) \\
\text{change } (\Gamma, C_k y i, \# y : s) c' (ts, ets) &= (ts, ets)
\end{align*}
\]

In Fig. 3 the trace generated for the following simple expression is shown:

\[
e @ \text{let } x_1 = \lambda x_2.e_1 \\
in e' @ \text{case } e_2 \text{ of } C_1 \rightarrow e_3 \\
C_2 \ x_3 \rightarrow x_1 \ x_3
\]

In the right-hand side, the trace for the main expression can be seen, while the traces for the different subexpressions are spread out across the figure. Every trace starts with a normal form \( \text{NF } e \) and ends with a root indicator \( \text{Root } e \). The downwards arrows represent the order in which the trace components are generated.

### 3.1 The RTS Browser

The RTS browser allows the user to travel backwards from the final expression to the original one along the trail of redexes and also to follow the redex trail
Adding Traces to a Lazy Monadic Evaluator

of its subexpressions. When a subexpression trail is being followed, the user is allowed to come back to the trail of its parent expression.

The main work is done by function menu. Its first argument is a list of traces, that represents the current trail of redexes that have not been visited yet. Initially, it is the list of traces in the final state. The second argument is a list of lists of traces, initially empty. The head of this list keeps the traces list of the parent expression, the second element is the traces list of the grandfather expression, an so on. This function first shows the current redex, and then shows a menu with several options. This is done by the function showMenu. The options may be one or more of the following ones:

1. Show the previous redex in the current trail (the next one in the list).
2. Start the trail of the subexpression.
3. Come back to the parent expression trail.
4. Stop.

The offered options depend on the current trace \( t \) and on the lists of parents’ trails \( tss1 \). The stop option (number 4) is always offered. Option 3 is only offered if the list of parents is non-empty. The rest of options are as follows:

- If the current trace is \( \text{Root } e \) and the list of parents is empty, then the end of the main trail has been reached and only option 4 is offered.
- If the current trace is \( \text{Node } e \ ts \), the user is offered the possibility of going a step further in the current trail and also of jumping to the redex trail of its subexpression.
- If the current trace is \( \text{NF } e \), then the user is offered the possibility of continuing with the current trail.
The Haskell program is the following:

```haskell
browser :: Heap -> Exp -> [RTSTrace Exp] -> IO ()
browser h ef ts
  = do putStr("The final expression is: " ++ showE h ef)
      menu h ts []

menu :: Heap -> [RTSTrace Exp] -> [[RTSTrace Exp]] -> IO()
menu h ts1@(t:ts2) tss1
  = do let e = expr t
       tse = traceOfSubExpr t
       ts3:tss2 = tss1
       putStrLn("The current redex is: " ++ showE h e)
       i <- showMenu t tss1
       case i of
         1 -> menu ts2 tss1 -- show the previous redex
         2 -> menu tse (ts1:tss1) -- go to normal form of subexpression
         3 -> menu ts3 tss2 -- come back to parent expression
         4 -> return () -- stop
```

Functions `expr` and `traceOfSubExpr` respectively extract from a trace `t` its expression and the trace of its subexpression. Function `showE` prints an expression by using the heap to follow the definitions of the free variables. This is an important feature of real browsers because expressions may be shown with different precision degrees. For our purposes, we may assume a simple solution. For instance, that pointers are followed only one step. If a normal form is reached, then this is printed with its own free variables as single question marks `?`. This avoids entering in cycles. If an unevaluated expression is reached instead (i.e. a `case`, an application or a `let`), then a double question mark `??` is printed.

### 4 Specific Functions for EDT Traces

A Freja tracer must construct an Evaluation Dependency Tree (in what follows abbreviated EDT) reflecting how the main expression is reduced to normal form. If an expression consists of several subexpressions, the tree will have as children the EDTs of each subexpression, plus an additional tree showing how the main expression is itself reduced. We call this last tree a `substitute` for the main expression. In our language, the only expressions having subexpressions are applications and `case`, and they have only one subexpression. As in RTS traces, variables and `let` expressions are not traced. So, an EDT trace consists of a `sequence` of substitutes for the main expression showing its evaluation path to normal form. This sequence is recorded in the state as a list of EDT traces. Also, for every intermediate expression, the trace of its only subexpression must be recorded. The natural way to do it is by using again a list of traces. In turn, each of these traces will have the same structure and so on recursively until all normal forms are reached. Notice that this leads us to a structurally identical definition to that of RTS traces. Thus, we can reuse the type definitions used
in the previous section. We just rename them in order to make clear that they represent Freja traces instead of RTS ones.

\[
\text{type } \text{EDTTrace } a = \text{RTSTrace } a \\
\text{type } \text{EDTState } = \text{RTSState}
\]

The only difference with RTS traces is that now, every list of substitutes must begin with a \text{Root} node and end up with a \text{NF} one. That is, EDT traces are just the \text{reverse} of RTS traces. This should be clear from the fact that RTS traces show the evaluation path of every subexpression but \text{backwards} instead of \text{forwards}. It turns out that to define a tracer constructing EDT traces on the fly (i.e. as the program is running) is a very inefficient task because it implies to concatenate each new trace \text{at the end} of a list representing the forward history of the current expression. A more efficient approach is to construct first an RTS trace for the program and then to \text{reverse} the whole tree. The function to do that is surprisingly simple, as it only needs to call the \text{reverse} function in the appropriate places:

\[
\begin{align*}
\text{-- reverse the trace} \\
\text{edtTrace } e_0 &= (\text{map } \text{rev} (\text{reverse } (\text{NF } e_f : \text{tsMain})), \Gamma, e_f) \\
\text{-- run the RTS tracer} \\
\text{where } ((\text{tsMain}, []), \Gamma, e_f) &= \text{run } e_0 \text{ startRTS} \\
\text{rev } (\text{Node } e \text{ ts}) &= \text{Node } e \text{ (map } \text{rev} (\text{reverse } \text{ts})) \\
\text{rev } t &= t
\end{align*}
\]

Notice that no \text{start} or \text{change} functions are defined now, as we reuse those defined for RTS.

Figure 3 can also be used as an example of an EDT trace. The only difference is that now, the pointers from the main trace to the subtraces should point to the top element \text{Root } e instead of to the bottom one \text{NF } e.

4.1 The EDT Browser

In order to locate the error, the EDT browser follows a question-answer protocol. It always shows at the terminal the current expression and its normal form (its \text{value}). Initially, the current expression is the original one. It assumes that the current expression wrongly reduces to its value and seeks to find why. The following method is used:

1. First, it shows the subexpressions of the current expression (in our language there is at most one) and asks whether the reductions to their corresponding values are correct or not.
2. If any of the subexpressions is wrong, this one becomes the current expression and the search proceeds in this subtree.
3. If all the subexpressions are correct, then it asks whether the first substitute reduces correctly.
If the answer is ‘yes’, then the reduction from the current expression to its first substitute is the wrong one.

Otherwise, the first substitute becomes the current expression and the search proceeds in the tree starting with it.

The main Haskell function is quite simple:

```haskell
browser :: Heap -> [EDTTrace Exp] -> IO ()
browser h (Root e :t:ts)
  = do
    let NF nf = last (t:ts)
    putStrLn ("Main expression: " ++ showE h e ++ "=>" ++ showE h nf)
    a <- question
    case a of
      Y -> return () -- Initial expression is correct, stop
      N -> browser' h t ts nf -- Initial expression not correct, search
```

The invariant assertion of `browser' h t ts nf` is that the expression `e` in `t` wrongly reduces to the value `nf`, that `t ≠ Root e`, and that `ts` is the trace of substitutes of `e`:

```haskell
browser' :: Heap -> EDTTrace Exp -> [EDTTrace Exp] -> Exp -> IO ()
browser' h (Node e ts1) ts2 nfe
  = do
    let Root se :t':ts' = ts1
    NF nfse = last (t':ts')
    t'':ts'' = ts2
    sust = expr t''
    putStrLn ("Subexpression: " ++ showE h se ++ "=>" ++ showE h nfse)
    a1 <- question
    case a1 of
      -- Subexpression is correct, investigate main expression
      Y -> do
        putStrLn ("Main expression: " ++ showE h sust ++ "=>" ++ showE h nfe)
        a2 <- question
        case a2 of
          -- Main expression also correct, error located
          Y -> putStrLn("Wrong reduction is: " ++ showE h e ++ "=>" ++ showE h sust)
          -- Main expression incorrect, investigate successors
          N -> browser' h t'' ts'' nfe
          -- Subexpression not correct, investigate successors
          N -> browser' h t' ts' nfse
      browser' h (NF e) [] nf
      = putStrLn ("Error not located. Reconsider your answers")
```

The actual Freja browser offers more possibilities to the programmer. For instance, the user can move backwards and forwards in the dialog, reconsidering any of the previous answers. We are sure that the information collected in our
trace is enough to ‘travel’ to any redex produced during program execution, and that more sophisticated browsers could be programmed having the same trace structure as a basis. Thus, obtaining all the facilities of the real browser is simply a matter of programming conventional algorithms, whose development is out of the scope of this paper.

5 Specific Functions for Hood Traces

In Hood the user can mark which program variables he wants to observe. The system will then show the result of evaluating the expressions associated to those variables. In our case, it only makes sense to observe let-bound variables. A let-bound variable $x$ may have several different incarnations $\{y_j\}$ which are pointers. So, the tracer must watch when a let-bound variable $x$ is renamed with an incarnation $y$, and maintain a list of incarnations for $x$. When the program stops, the browser prints the expressions bound in the heap for these incarnations of $x$. As a first approach, Hood’s state consists of a table $tpv$ with program variables (of type $\text{VarP}$) used as access keys, and lists of pointers (of type $\text{VarH}$) as associated values.

\[
\text{type StateHood} = \text{Table VarP } [\text{VarH}]
\]

Initially, the variables selected by the user are introduced in this table, with an associated empty list of incarnations. Each time a variable $x$ is renamed in a let rule, if $x$ belongs to the observed variables (or it is an incarnation of an observed variable) its incarnation is added to the list for $x$. To this purpose, we assume that there exists a function

\[
\text{lastN} :: \text{Heap} \to \text{Int} \to [\text{VarH}]
\]

giving the names of the last $n$ fresh variables which have been bound in the heap. These names are obtained from the configuration $c'$ after the machine transition. The definitions of $\text{start}$ and $\text{change}$ for this simple version of Hood follow.

\[
\text{start}' :: [\text{VarH}] \to \text{Exp} \to \text{StateHood}
\]

\[
\text{start}' \; xs = \text{addPairs emptyTable} \; (\text{zip} \; xs \; (\text{repeat} \; []))
\]

\[
\text{start} = \text{start}' \; \text{userSelectedVariables}
\]

\[
\text{change} \; (\Gamma, \text{let} \; \{x_i = e_i\}_{i \in \{1..n\}} \; \text{in} \; e, s) \; (\Gamma', \ldots) \; tpv
\]

\[
= \text{foldl} \; \text{insert} \; tpv \; [(x, y) \mid (x, y) \leftarrow \text{zip} \; xs \; ys, \; \text{isObserved} \; x \; tpv]
\]

where $xs = [x_1, \ldots, x_n]$

\[
y = \text{lastN} \; n \; \Gamma'
\]

\[
\text{change} \; c \; c' \; tpv = tpv
\]

where $\text{insert} \; tpv \; (x, y)$ is assumed to insert $y$ in the list associated to $x$, and $\text{isObserved} \; x \; tpv$ gives True if and only if $x$ belong to table $tpv$.

In Hood, it is also possible to observe program variables bound to functions. If $x$ is a program variable bound to a functional expression, Hood observes all
applications of (the incarnations of) \( x \) to different arguments, and collects the pairs \((\text{argument, result})\) of these applications. When the program stops, the browser prints the observations of every incarnation of \( x \) as a collection of pairs. No distinction need to be done between the different incarnations.

To emulate this feature, the first step is to introduce the incarnations \( y_j \) of \( x \) in the table \( \text{tpv} \) of program variables as before. Then, all the \( y_j \) are observed until (if ever) they are referenced after being updated by a lambda normal form (see first rule for variables in Sect. 2.2). Should this happen, the resulting lambda form in the control expression is observed until it is applied to an argument (see second application rule and flag \( m \) below). Then a pair \((y, y')\) is stored in an auxiliary stack \( \text{ps} \) of pairs, being \( y \) the incarnation of the lambda and \( y' \) the pointer to the argument. Finally, when the body of the lambda is reduced to a normal form, the pair \((y', \text{nf})\) is stored in a table \( \text{thl} \) for lambda heap-variables, using \( y \) as access key.

The refined Hood state adds to the simple one the new table \( \text{thl} \), the auxiliary stack \( \text{ps} \), and the flag \( m \). Both the stack pairs and the flag are embedded into corresponding \textit{Maybe} types. In the stack, \textit{Nothing} values are pushed when they are needed to simulate the slots of Sestoft’s stack, for example in application and \textit{case} rules. In the flag, a \textit{Just} \( y \) value means that the lambda bound to \( y \) is being observed until it is applied to its argument, while a \textit{Nothing} value means that no lambda is being observed.

```
type HoodState =
  (Table VarP [VarH], --relates program variables to incarnations
   Table VarH [((VarH,Exp)],--relates lambda variables to pairs(arg,result)
   [Maybe (VarH,VarH)], --auxiliary stack for observing lambda-bodies
   Maybe VarH) --auxiliary flag to observe lambda-applications
```

Function \( \text{start}' \) and the rule given for \textit{let} above, should be modified for this more complex state. In the first case, an empty \( \text{thl} \) table, an empty auxiliary stack and a \textit{Nothing} flag should be returned. In the second case, \( \text{change} \) behaves as the identity function for the three new components.

```
change \((\Gamma \cup [y \mapsto \lambda x.e], y, s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{ps}, m)\)
  | \( y \in \text{range} \text{tpv} \) = (\text{tpv}, \text{insert thl} (y, []), \text{ps}, \text{Just} y)
  | otherwise = (\text{tpv}, \text{thl}, \text{ps}, \text{Nothing})
change \((\Gamma, \lambda x.e, y' : s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{ps}, \text{Just} y)\)
  = (\text{tpv}, \text{thl}, \text{Just} (y, y') : \text{ps}, \text{Nothing})
change \((\Gamma, \lambda x.e, y' : s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{ps}, \text{Nothing})\)
  = (\text{tpv}, \text{thl}, \text{ps}, \text{Nothing})
change \((\Gamma, e, y, s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{ps}, m)\)
  = (\text{tpv}, \text{thl}, \text{Nothing} : \text{ps}, \text{Nothing})
change \((\Gamma, \text{case } e \text{ of } \text{alts}, s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{ps}, m)\)
  = (\text{tpv}, \text{thl}, \text{Nothing} : \text{ps}, \text{Nothing})
change \((\Gamma, C_k \overline{y_i}, s)\) \( c' \) \((\text{tpv}, \text{thl}, \text{Just} (y, y') : \text{Nothing} : \text{ps}, m)\)
  = (\text{tpv}, \text{insert thl} (y, (y', C_k \overline{y_i})), \text{ps}, m)
```
\[ \text{change } (\Gamma, C_k \ y_i, s) \ c' (tpv, \ thl, \ Nothing : ps, \ m) \\
= (tpv, \ thl, \ ps, \ m) \]
\[ \text{change } (\Gamma, \lambda x. e, s) \ c' (tpv, \ thl, \ Just (y, y') : Nothing : ps, \ m) \\
= (tpv, \ insert \ thl \ (y, \ (y', \lambda x. e)), \ ps, \ m) \]
\[ \text{change } (\Gamma, \lambda x. e, s) \ c' (tpv, \ thl, \ Nothing : ps, \ m) \\
= (tpv, \ thl, \ ps, \ m) \]

5.1 The Hood Browser

Hood’s browser is quite simple. It just prints, for each observed value, and for each incarnation of it, the values collected in the two tables. For non-functional values, only table \(tpv\) is used. For functional ones, both \(tpv\) and \(thl\) tables are needed: The first one gives the different incarnations of the functional variable, each one representing a possibly different function; the second one gives the collected pairs \((\text{arg}, \text{result})\) collected for the incarnation.

```haskell
browser :: Heap \rightarrow Table VarP [VarH] \rightarrow Table VarH [(VarH, Exp)] \rightarrow IO ()
browser h tpv thl = mapM_ printPV (dom tp)
  where printPV x = mapM_ printHL (associatedValue x tpv)

printHL y
  | y 'notElem' dom thl = showV h y
  | otherwise = mapM_ printPair (associatedValue y thl)
printPair (y,e) = putStrLn (showV h y ++ "+=>" ++ showE h e)
```

where function \(\text{showV}\) is very similar to \(\text{showE}\). It shows the expression bound in the heap to a pointer \(y\). Notice that it is trivial to modify the browser in order to restrict the amount of information shown. This could be useful when a lot of data have been stored. In that case, the user would prefer to obtain incrementally the information he desires. Thus, initially only part of the information should be shown, and the user could ask the system to show him the rest of information as needed.

6 Conclusions

The work presented here can be regarded in several different ways. The most interesting for us is to look at it as a \textit{what if} paper. We imposed ourselves a severe restriction at the beginning: \textit{Not to modify either the original program or the abstract machine executing it}. Keeping this restriction, we have tried to emulate the main features of three up-to-date tracers for lazy languages. Our purpose was to investigate how far we could reach in this task, and also to know which could be the observations we would require both from the program and from the abstract machine in order to collect enough information to build the traces.

The conclusions in this respect are very optimistic: There is not need to modify anything in order to collect the required information. The tracers essentially need to have access to the machine configuration \textit{previous} to each reduction.
step. Only in one case—the lastN function needed for Hood’s tracer—an access to the configuration after the reduction step was needed. The reason was to know the most recent n pointers created in the heap. The features offered by the emulated tracers are of course more sophisticated that the ones presented here, but this has to do with the kind of browsers they provide rather than with the collected information. Thus, obtaining the real tracers is just a matter of developing user-friendly interfaces, using conventional techniques.

A second conclusion is that correctness of our approach is guaranteed by definition. Our tracers are implemented by a function change which is not allowed to influence the machine configuration, i.e. the heap, the control expression or the control stack. In this way, a traced program will produce exactly the same reductions and results as the original one.

Another conclusion is that the collected data structures for RTS traces and for EDT traces are basically the same. This was already anticipated in [3]. The difference between these two tracers resides in how this information is presented to the user, i.e. they differ in the browsers. In the conclusions of [3], the authors explain that perhaps the EDT tracer is more appropriate for beginners because of its friendly question-answer interface, while RTS seems more adequate for expert programmers knowing about redexes and reductions. Our work offers a third possibility of having a common trace collector and two browsers for the user to choose between. In fact, it could be possible to develop a browser combining the best of both worlds. It could start using a question-answer style, and then an RTS-like browser could be used to faster locate the error, once it has been approached by using Freja.

Finally, the work presented here provides a framework and a complete set of algorithms which can be used to experiment with other alternative designs of trace systems before embarking on a fully-fledged implementation. As developing a complete debugger is costly, it is useful to be able to write prototypes. In our approach, the creation as such prototypes only needs the programmer to define the state of the trace and the functions start and change. Therefore, they can be developed in a very short time. For instance, it could be possible to simulate other existing debuggers like [7, 11] and [5, 2], or to implement modifications to them.

Acknowledgments

This work has been partly supported by the Spanish project CICYT-TIC2000-0738. The authors want to thank the rest of members of the UCM functional programming group for valuable comments on a previous draft version of this paper.

References

Adding Traces to a Lazy Monadic Evaluator


Functional Vs Object-Oriented Distributed Languages

Robert F. Pointon\textsuperscript{1}, Steffen Priebe\textsuperscript{2}, Hans-Wolfgang Loidl\textsuperscript{1}, Rita Loogen\textsuperscript{2}, and Phil W. Trinder\textsuperscript{1}

\textsuperscript{1} Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh, Scotland; \{rpointon,hlloidl,trinder\}@cee.hw.ac.uk
\textsuperscript{2} Fachbereich Mathematik und Informatik, Philipps-Universität, Marburg, Germany; \{priebe,loogen\}@informatik.uni-marburg.de

Abstract. Conventional distributed programming languages require the programmer to explicitly specify many aspects of distributed co-ordination, including resource location, task placement, communication and synchronisation. Functional languages aim to provide higher-level abstraction, and this paper investigates the effectiveness of this for distributed co-ordination. The investigation contrasts and compares contrasts Java and two Haskell-based distributed functional languages, Eden and GdH. Three distributed programs are used as case studies, and the performance and programming effort are reported.

1 Introduction

Improved network technology and a rapidly expanding infrastructure of interconnected computer systems have generated a resurgence of interest in distributed languages. This is demonstrated by the popularity of the modern object-oriented language Java.

Theoretically non-strict functional languages offer additional benefits for constructing distributed systems. These languages have sophisticated, e.g. polymorphic, type systems to further enhance the safety of distributed systems. They also have a highly-dynamic model of distribution with communication on-demand, a relatively high degree of distribution transparency with the runtime-system managing details of communication and synchronisation, and the potential to safely abstract over distribution-control primitives with higher-order polymorphic functions.

These benefits of distributed functional languages are bought at the price of an elaborate implementation. This may explain why, despite many distributed functional language designs over the past decade, robust well supported implementations have only recently become available. These include Erlang \textsuperscript{5}

\textsuperscript{*} Supported by UK’s EPSRC (grant GR/M 55633), British Council/DAAD (travel grant no. 1097), and the Austrian Academy of Sciences (APART 624).
and Haskell with Ports [10], which we discuss in Section 6. The authors represent groups who have implemented two non-strict distributed functional languages based on Haskell [12], namely: Eden [2] and Glasgow distributed Haskell (GdH) [13].

The paper opens with Section 2 introducing distributed language concepts and describing the distributed programming model underlying Eden, GdH and Java. For the comparison three different applications of increasing size were chosen. The first two applications, Broadcast Ping (Section 3) and the Chat Room (Section 4) were implemented in all three languages so as to contrast and compare the style of programming. The final Multi-user Shared Map (Section 5) application is implemented only in GdH so as to show some features unavailable (or non-trivial to implement) in the other systems. Section 6 relates our work to other distributed functional languages, and finally a summary is presented in Section 7.

2 Distributed Concepts and Languages

2.1 Distributed Concepts

Table 1. Language Comparison of Distributed Concepts.

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>GdH</th>
<th>Eden</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm</td>
<td>Functional</td>
<td>Functional</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>Decentralised</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Location Aware</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Fault Tolerant</td>
<td>Partial</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Techniques:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Lookup</td>
<td>N/A</td>
<td>N/A</td>
<td>Registry</td>
</tr>
<tr>
<td>Thread Placement</td>
<td>Remote evaluation</td>
<td>Process</td>
<td>RMI</td>
</tr>
<tr>
<td>Synchronisation</td>
<td>MVar &amp; shared data</td>
<td>Implicit</td>
<td>Synchronised class</td>
</tr>
<tr>
<td>Communication</td>
<td>MVar &amp; shared data</td>
<td>Channel</td>
<td>RMI</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Non-strict</td>
<td>Mixed</td>
<td>Strict</td>
</tr>
<tr>
<td>Fault-tolerance</td>
<td>Exceptions</td>
<td>Exceptions</td>
<td>Exceptions</td>
</tr>
</tbody>
</table>

We compare the distributed programming models of Java, Eden and GdH using the following concepts. Decentralised: co-operating systems, e.g. client-server, vs centralised — a single system, often hierarchal with a single point of failure. Location Aware: able to identify and make use of the attributes and resources at each processing element (PE) location, vs location independent — where the location is irrelevant or handled automatically. Fault Tolerant: able to detect and recover from errors.

Depending on the concepts supported by the languages they provide certain distribution related techniques. Resource Lookup: the means for locating/testing
for resources from another location. **Thread Placement:** how work is sent to remote locations. **Synchronisation:** the way of co-ordinating activities. **Communication:** the means for sending and receiving information. **Evaluation:** the execution model of the language, where *strict* results in function arguments being evaluated before entering a function, vs *non-strict* where evaluation is delayed until the arguments are used. **Fault-tolerance:** the method of handling errors. Table 1 summaries and presents the concepts and techniques used by the three languages.

### 2.2 Languages

**Java** [6] is well known and only briefly outlined here. It supports distribution via the Remote Method Invocation (RMI) API, which allows Java programs to invoke methods of remote objects. Dynamic class loading allows programs to load code on-demand, in particular, the code can be loaded across the internet. A subset of the commands and classes are shown in Figure 1.

```plaintext
\locks objects of \texttt{expr} to ensure mutual exclusion in \texttt{statement}
\synchronised (expr)
\texttt{statement}
\to allow remote objects by defining new classes whose:
\begin{itemize}
  \item interface extends \texttt{java.rmi.Remote}
  \item implementation extends \texttt{java.rmi.server.UnicastRemoteObject}
\end{itemize}
\associate a name with a new remote object
\texttt{static void java.rmi.Naming.rebind(string name, remote obj)}
\return a reference to a remote object
\texttt{static remote java.rmi.Naming.lookup(string name)}
```

**Fig. 1.** Java Mechanisms for Distribution.

**GdH** [13] supports both parallel and distributed computation using two classes of threads: purely-functional threads and explicit side-effecting I/O threads. The parallel concepts are inherited from Glasgow Parallel Haskell (GpH) [14], and the concurrent I/O concepts form Concurrent Haskell.

Remote I/O thread placement with implicit communication of the result is achieved by a new language primitive for remote evaluation (revalIO). Communication and synchronisation may be implicit: threads on one PE can share variables with threads on other PEs. Data transfer is lazy: only the immediately required portion of a data structure is transferred when required by another thread. I/O threads can explicitly communicate and synchronise using polymorphic semaphores (MVars) and abstractions built on top of them, such as channels.
and buffers. Thread location is exposed to the program level so that a program can use resources unique to a PE.

Furthermore evaluation strategies, as developed for parallel execution in GpH [14], allow control of the evaluation degree and the evaluation order of Haskell expressions. Strategies provide the means for tailoring the non-strict behavior of evaluation as well as communication. We have started to extend evaluation strategies to express distributed co-ordination by specifying both work and data placement. The most important basic strategies and combinators are presented in Figure 3.

---

**Fig. 2.** GdH Mechanisms for Distribution.

---

**Fig. 3.** GdH Strategies for Distributed Co-ordination.
Eden uses explicit processes together with implicit communication via stream-based channels. Lean communication is achieved by an eager sending protocol and communication connections can be built up flexibly by using dynamic reply channels. Complex process nets can be constructed to realize transformational and reactive systems.

Process specifications are explicitly visible in the language: Process abstractions (see first part of Figure 4 for an example) specify which output a process will generate depending on process parameters $p_i$ (supplied on creation time) and process input $in_i$ (supplied later via communication). One example is the predefined `merge` process abstraction, which joins a set of streams into a single stream non-deterministically allowing for many-to-one communication. Such an abstraction serves as a template for later process instantiation: When a process abstraction is applied to a full set of input parameters via the `#` operator, a process will be created. The system function `noPE` can be used to query the runtime system for the number of available PEs. This is useful as it is advisable to relate the number of processes to the number of available PEs.

As future work, a system function similar to the GdH `myPEId` and an extension of the process instantiation operator (`#`) for direct process placement could easily be added to Eden to support more location-awareness.

\[
\begin{align*}
\text{--- process abstraction} \\
pabs & :: pt_1 \to \ldots \to pt_k \to \text{Process} \ (it_1, \ldots, it_m)(ot_1, \ldots, ot_n) \\
pabs \ p_1 \ldots p_k & = \text{process} \ (in_1, \ldots, in_m) \to (o_1, \ldots, o_n) \\
\text{where} \ldots \\
\text{--- process instantiation} \\
\# & :: (\text{Transmissible} \ a, \text{Transmissible} \ b) \Rightarrow \text{Process} \ a \ b \to a \to b \\
\text{noPE} & :: \text{Int}
\end{align*}
\]

Fig. 4. Eden Mechanisms for Distribution.

3 Broadcast Ping

Broadcast Ping is similar in nature to the UNIX ping command in that it determines the round trip time between machines. This particular version of ping determines the time necessary to instantiate a remote thread, perform a trivial environment variable lookup to confirm the location, and return this result. A distinguished, or main, PE performs this process for every PE and returns a list of locations and timings. The ping program demonstrates remote thread creation; access to a remote resource; and communication from remote threads.
3.1 Programming Style

Table 2 summarises the program size for each language. We separate the computational code from the code for the co-ordination of the distribution and comment on language specific features of the implementations.

The GdH and Eden programs share the same structure with only the methods for determining the PEs and creating a remote task changing. Java uses a decentralised model and therefore the program is split into two components (client and server) together with a common interface. The greater size also results from the fact that Java forces the programmer to consider exceptions when defining methods, i.e. to implement fault-tolerance — though this may be a design advantage when compared to other languages where the programmer often delays implementation of exception handling until it is required.

Table 2. Ping Programming Style.

<table>
<thead>
<tr>
<th></th>
<th>GdH</th>
<th>Eden</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>16</td>
<td>17</td>
<td>37</td>
</tr>
<tr>
<td>Lines for distribution</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Other comments</td>
<td>—</td>
<td>—</td>
<td>Fault-tolerant</td>
</tr>
</tbody>
</table>

3.2 Performance

The program in each language is executed on a network of four machines using ushas as the main PE. Table 3 gives the execution times. Where a remote PE is accessed, communication costs dominate and there is no significant difference between the languages. However, the two centralised languages GdH and Eden are faster in the case where communication is local, because the Java program has separate client and server programs and must perform inter-process communication and registry lookup to connect.

Table 3. Ping Performance (times in milli-seconds).

<table>
<thead>
<tr>
<th></th>
<th>GdH</th>
<th>Eden</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>ushas (local)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>bartok</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ncc1705</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>brahms</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
3.3 Discussion

The differences between centralised and decentralised languages are very apparent from this program in the structure of the program, though the actual performance is similar for all systems. Java possesses a potential advantage in that the enforced exception handling results in a more robust program, whereas the other systems crash if a PE fails during a ping.

4 Chat Room

The Chat Room is a client server type application, where multiple users chat to each other through a message board maintained by the server. The program also has additional extensions for maintaining simulation information at the server and a nominated super user is allowed to manipulate it. The Kitekk simulation is discussed in [11] and the environment is implemented in Java as described in [16]. The client server nature demonstrates the instantiation of remote threads; the use of shared immobile resources for communication; the creation of individual GUIs for users to interact with the client threads; and fault-tolerance in handling of remote GUI errors.

![The Chat Room Interface. (With all X screens re-directed to one host.)](image_url)
4.1 Programming Style

We have investigated two alternative implementations of the system, one with the server represented as *shared synchronised state*; and one with a *server as a process* managing its internal state and operating on a queue of messages from the clients. Table 4 summarises the program size and details of in each language.

<table>
<thead>
<tr>
<th></th>
<th>GdH</th>
<th>Eden</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>500</td>
<td>≈ 450</td>
<td>1500</td>
</tr>
<tr>
<td>Lines for distribution</td>
<td>15</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Other comments</td>
<td>—</td>
<td>(Incomplete) Fault-tolerant</td>
<td></td>
</tr>
</tbody>
</table>

*Shared Synchronised State* is where the server stores the state and the runtime system maintains the state ensuring consistency, mutual exclusion of access and lack of deadlock, etc. The existing Java implementation used a server as shared state and the GdH implementation duplicated this model. Java allows the programmer to mark classes as “synchronised” to enforce mutual exclusion of access to the objects of that class, after which no additional effort is needed by the programmer. GdH required the user to setup an *MVar* to hold the state and then use the code from Figure 6 (similar to what exists in the Haskell semaphore libraries) to implement mutual exclusion and create a function `lock` through which all access can take place.

```
lock :: MVar a → (a → IO(a,b)) → IO b
lock sync update = do
  v ← takeMVar sync
  (nv,res) ← update v  -- apply the update operation
  putMVar sync nv
  return res
```

Fig. 6. GdH Implementation of a Mutual Exclusion Lock.

*Server as a Process* requires the programmer to implement a process which explicitly manage its external state in response to messages received. Thus the programmer is responsible for defining the access protocol unlike shared synchronised state where the protocol is defined in the runtime system.

In Eden the server and each client are implemented as a process which changes state according to the protocol when receiving messages. Communication is implicit in Eden, and message queues are modeled as lazy lists, thus the
communication operations are therefore replaced by list operations. The central part of the Client process shown in Figure 7 is the function cloop. It handles incoming messages from the Server, changes the Client’s state, and sends new messages to the Server according to the communication protocol.

```haskell
client = process toClients → toServers
    where (_, toServers) = unsafePerformIO (cloop cInitState toClients)

    cInitState = ([], initialSU, CNone) -- initial client state

    cloop :: (ClientState, [ToClient]) → IO (ClientState, [ToServer])
    cloop (s@(out, su, state), tcs) = do
        ...
        -- process messages

server = process toServers → toClients
    where...
    -- similar structure to client
```

Fig. 7. Eden Client and Server Structure.

The `unsafePerformIO` in `client` is introduced to allow side-effecting I/O operations like GUI control, whereas in GdH every remote computation is enclosed in an I/O thread by default.

```haskell
main = print fromClientsM -- demand to generate processes
    where
        fromClientsMerge = merge fromClients
        fromClients = [client # tc | tc ← toClients]
        toClients = map (CEmpty:) (server fromClientsMerge)
```

Fig. 8. Eden Process System Instantiation.

The process system itself is constructed via a set of mutually dependent list definitions in Figure 8. The predefined merge process is used to join the client streams into a single input stream for the server. The ease of creating process nets comes at a cost: Eden currently lacks the ability to specify a particular PE with the runtime system using a round robin scheme to choose a process PE.

### 4.2 Discussion

The GdH and Java implementations are similar though the Java is far more verbose. The Eden implementation reveals the ease at which the process creation can be abstracted away and defined at a higher-level, thus vastly simplifying all the communication and process creation issues. The lack of location awareness
is highlighted in Eden as there is currently no way to specify which PE to place the process on, this could be a potential handicap for some programs where the programmer wants something to occur on a particular PE.

5 Multi-user Shared Map

The Multi-User Shared Map is another client server type application with multiple users interacting under the co-ordination of the server. Maps of geographical information systems (GIS) in ARC [3] format are used to provide the data — in this case the maps are comprised of approximately 150,000 drawing lines. The program demonstrates how a large shared data structure can be used implicitly from multiple PEs in the following ways. Unique resources on different PEs are used to generate a single monolithic data structure. Scalability gained by spreading the structure across multiple PEs. Parallel co-ordinated data access as multiple users interact simultaneously with the data.

![Fig. 9. Zooming into the Map.](image)

5.1 Programming Style

This program is written only in GdH to illustrate some of the communication control strategies supported by the language. The core GdH program is shown in Figure [10] and proceeds as follows. The main function (on the main PE) reads in the quadtree map. A GUI running window is mapped across all the available PEs whilst sharing the quadtree structure. The sub-function showit then is invoked by the GUI to show a particular region which it does by determining the visible components of the quadtree via visible and then drawing it to the GUI with draw.

The function window implements each remote GUI and in doing so specifies the way how the communication of the quadtree from the main PE should be performed. We have investigated the following alternatives for the communication. In each case the only modification in the code affects the strategy applied to the (shared) data structure that is to be transferred.

*Implicit Sharing:* the data is transferred on the demand from each of the remote windows. This requires no change to the original code.
main = do
  q ← readQuad "map" 'usingIO' rnf³
  pes ← allPEId
  mapM (rforkIO (runGui (window q))) pes

window q gui = do
  init gui
  ...
  show it b = do
    let m = visible q b
    draw gui m 'demanding' rnf m

⁎ The strategy ‘usingIO’ is used to ensure that the entire quadtree is read in and processed before the program continue.

† The ‘demanding’ strategy on the draw is used to ensure that the visible components of the quadtree have been fully evaluated before sending them to the GUI, this avoids the problem of the GUI blocking while it does the evaluation.

Fig. 10. Structure of Basic Map Program.

Eager Transfer: transfer a copy of the entire quadtree to the remote PE before the GUI continues. The changes required for this are shown in Figure 11.

Explicit Functional Communication: send work requests to the main PE where the quadtree is and let the main PE calculate the visible components and return this result to the GUI. The changes to control the visible evaluation are shown in Figure 12.

Explicit I/O Communication: explicitly communicate with a server process on the main PE, sending a request for the server to calculate the visible result, then wait for the result. In this case the remote GUI is started with a shared reference to a communication channel (ch) to the server rather than a shared quadtree. The introduction of the explicit I/O can be seen in Figure 13. This is identical in approach to the explicit functional method yet implements it using I/O commands in the style more similar to traditional languages.

5.2 Performance

In measuring the performance of the different strategies for communication, timing code is placed around the communication. The results collected are at each step as the user zooms into the map, as in Figure 9, and then reverses and zooms out, thus traversing the same data space of the map. In Figure 14 these discrete steps of traversal are plotted along the x-axis, and the time in milli-seconds for each step is plotted along the y-axis.

The two explicit approaches yielded identical results which was unsurprising since they are using the same technique, therefore they are only shown once on the graph. From Figure 14 it can be seen that the eager approach has a huge startup cost for the first step but after that it has the lowest map traversal times.
window q gui = do
  init gui 'demanding' rnf q
  ... showit b = do
    let m = visible q b
    draw gui m 'demanding' rnf m

*a Demand the full quadtree when the window is initilised.

Fig. 11. Eager Approach for Map Sharing.

window q gui= do
  init gui
  ... showit b = do
    let m = visible q b
    draw gui m 'demanding' (rnf ((m 'using' rnf) 'on' mainPE))

*a Evaluate the visible function 'on' the main PE and then demand the result back at the GUI.

Fig. 12. Explicit Functional Map Communication.

main = do
  ch ← newChannel
  q ← readQuad "map" 'usingIO' rnf
  pes ← allPEId
  mapM (rforkIO (runGui (window ch))) pes
  let server = do
    respond ch (\ b → return (visible q b) 'usingIO' rnf)
    server
    forkIO server

window ch gui = do
  init gui
  ... showit b = do
    m ← request ch b
    draw gui m 'demanding' rnf m

*a The forked server thread calculate the visible components in response to the requests it receives.
*b The GUI sends off a request down the channel to the server.

Fig. 13. Explicit I/O Map Communication.
The explicit approaches are much slower due to always having the communication overhead. The sharing approach varies considerably but the sharing causes it to eventually match the fast eager times.

![Event Timeline]

**Fig. 14.** The Effect of Different Communication Strategies.

5.3 Discussion

The implicit sharing technique required no changes to the code and is often sufficiently fast. We would expect that the sum of the times to traverse the entire data space using the sharing method would be equal to the sum of the times via the eager approach — the communication may be more spread out, but no more communication is needed. For the explicit approach the sum of the times would be higher as the communication must always be repeated. It proves straightforward to code alternative communication schemes using different strategies in GdH.

6 Related Work

Modern distributed languages often make use of advanced concepts for the computation component of the language by using class hierarchies with inheritance, strong typing or automatic garbage collection. However, their control component usually relies on explicit communication and task placement with few facilities for abstracting over these basic constructs. In this section we relate our languages, Eden and GdH, to those distributed functional languages that provide
some abstraction facilities over their basic co-ordination constructs. A more de-
tailed survey of distributed extensions of Haskell is given in [15].

Haskell with Ports [10] is a library for Concurrent Haskell adding additional
monadic commands for communication between PEs. This results in a decen-
tralised distributed language with explicit communication and moderate support
for fault-tolerance. Ports are an abstraction over UNIX-style sockets, realising
dynamically typed uni-directional communication channels. Ports are first-order
values in the language and the powerful Haskell monad libraries can be used to
build abstractions over ports.

ERLANG [1] is a strict, impure, first-order, untyped language with explicit
communication constructs. In our classification ERLANG qualifies as decen-
tralised, location-aware with explicit port-based communication. ERLANG has
been developed and is used in telecommunications applications and therefore
emphasises important pragmatic issues such as real-time fault tolerance via time-
outs and exceptions, hot loading of new code, message authentication etc.

In a distributed language logic variables can be used as shared single-
assignment variables in order to realise bi-directional communication channels.
Several languages such as Distributed Haskell [4] and Curry [7] use this con-
struct to obtain a declarative co-ordination component of their language. Similar
constructs are used in OZ [8], a distributed language combining functional,
object-oriented, and logic paradigms for computation as well as co-ordination.
Finally, Brisk [9] is based on a virtual shared heap, similar to GdH, in order to
support implicit communication.

7 Summary

We have compared two distributed functional languages, Eden and GdH, with
Java using three programs. Java’s distribution model is decentralised, and hence
is the most powerful, followed by GdH which is location aware, followed by Eden
which is neither. Although both the Eden and GdH versions of the programs are
significantly shorter than the Java version, only in Eden is the distributed co-
ordination higher-level than Java. In effect the distributed co-ordination avail-
able in Java and GdH is at a similar level of abstraction. However, it is easiest
to abstract and compose co-ordination constructs in GdH, e.g. to construct a
lock from an MVar. The value of separating of co-ordination and computation
concerns has been established for parallel programming [14], and we find that it
also aids distributed programming. For example, it is easy to specify alternative
communication strategies in the multi-user shared map. Performance of all three
languages is similar for a simple program (ping), although tuning is easiest in
GdH, merely requiring changes to the strategies.

In terms of selecting a language for a given application, the following tentative
guidelines are proposed. If a dynamic client server model with fault-tolerance is
required then use Java. If large complex data structures are to be shared then use
GdH. If the problem has a regular process model and is mostly purely functional
then use Eden.
References

Abstract. Persistent programming offers the prospect of seamless integration of programs with long-lived data, offering the prospect of constructing systems that allow more rapid program development, and also simplifying the process of writing applications whose purpose is to handle long-lived data. While there have been some previous attempts to produce persistent functional languages, the majority of these have been interpreted, and performance has generally been seriously compromised. It has therefore become something of a shibboleth that persistence cannot be implemented efficiently in a purely functional language. This paper offers the first systematic study of this claim.

This paper describes the first-ever implementation of orthogonal persistence for a compiled purely functional language, based on an existing St Andrews persistent object store. Preliminary performance results show that it is possible to implement orthogonal persistence efficiently and there is hope that the result is more efficient than more straightforward approaches such as binary I/O.

1 Introduction

There is a clear symbiosis between the functional and persistent programming models. In a pure functional programming language, such as Haskell, there is no distinction between data and program: functions are themselves values. Similarly, persistent programming makes no distinction between programs and data that persist longer than a single session. The combination of the two techniques should therefore lead to a simple, elegant model of programming with long-lived values and functions.

The similarities at the language level also extend to the implementation level. In a functional framework, it is natural to use graph reduction and exploit shared pointers in order to achieve an efficient implementation of lazy evaluation. In the persistent world, sharing is preserved within the persistent store through the use of shared persistent identifiers to create on-disk graph structures. This similarity extends to techniques that can be used to access persistent data or programs.

This paper describes the design and implementation of an orthogonally persistent version of the Haskell functional language. The system has been constructed with efficiency in mind: we have used a well-developed St Andrews persistent object store as the basis of our implementation of persistence, and we have integrated this with the fastest implementation of Haskell which is currently available – the highly-optimised Glasgow Haskell Compiler. This allows us to study the previously unresearched question...
of whether persistence can be implemented efficiently in a purely functional language. Folklore results suggesting the opposite (based on conclusions from interpreted implementations) have led to the construction of a number of competing, but less-capable, systems implementing variants of binary I/O. It is our contention that it is not only possible to achieve acceptable efficiency in an orthogonally persistent system, but that this efficiency can be greater than the less general solutions based on binary I/O.

We intend to use the system that we have developed in a number of ways. Firstly it will underpin our work on the hypersheet model of program development [5]. Secondly, it will be used to help the development of long-lived applications such as the Bremen ToolBus interface [11][12]. Thirdly, it will allow applications to handle long-lived data more efficiently, while preserving both sharing and laziness.

The remainder of this paper is structured as follows. Section 2 considers the design space, outlining a basic design based on the use of monads. Section 3 discusses implementation issues. Section 4 discusses related systems based on the use of binary I/O or orthogonal persistence. Finally Section 5 concludes.

2 Design

In order to avoid unnecessary duplication of effort, we have based our system on a simple and reliable persistent object store (POS) [2] already used to implement a number of persistent languages including Napier [17], Galileo [1]. This store provides a number of basic operations that allow data to be read from or written to persistent storage either as raw data or as persistent objects, with rudimentary support for concurrent access. Since we intend to build our own, more appropriate, high-level interface to the store, this is a better target than an apparently more sophisticated design, such as the PJama store [7],[22], which is specialised to handle Java objects and which provides high-level features that would hinder our implementation. By exploiting memory mapping features in the operating system, unnecessary disk accesses can be reduced or eliminated, so this store also promises to offer reasonable efficiency levels.

Our high-level design is based on the premise of explicit persistence. Having explicitly marked an object as persistent, it follows that all objects reachable from it will persist automatically. Thus orthogonal persistence [3] is maintained while retaining the ability to mark selected objects as persistent explicitly. This helps avoid the time and space performance penalties that can arise with implicit persistence through saving unwanted intermediate values. Our objective is to allow programmers to implement persistent applications using our Haskell interface, rather than to provide a persistent programming environment as with PolyML, for example [14][15]. Such an environment could be constructed using our interface, of course – one of our long term aims is to do so using hypersheet modules [5] – but we do not wish to constrain programmers to use our environment.

2.1 The Haskell POS Interface

We provide three basic operations on the persistent store: open, close, commit. These operations use the standard monadic I/O interface [18], so permitting the use of standard error handling mechanisms etc.
data PSHandle

open :: FileName -> IO PSHandle
close :: PSHandle -> IO ()
commit :: PSHandle -> IO ()

The `open` operation creates a handle to a named persistent store; `close` closes the store attached to the handle, committing any changes that have been made; `commit` commits the store without closing the associated handle.

These operations are similar to those provided by many persistent languages such as Napier [17], or Staple [10], except that the use of an explicit handle rather than implicit state potentially allows multiple persistent stores to be used simultaneously.

### 2.2 Accessing Values

Two operations provide access to persistent values:

get :: Persistent a => PSHandle -> PSId -> IO a
use :: Persistent a => PSHandle -> PSId -> a -> IO a

The `Persistent` class is used for doing the dynamic typing and is described in the next section.

The `get` operation provides access to a value which is known to be persistent. `PSId` is the persistent identifier: a string name that uniquely identifies the persistent value. If the persistent identifier cannot be found in the persistent store then an error is raised within the `IO` monad.

The `use` operation is used to store a value in the persistent store, or to retrieve an already persistent value. The third argument is the value of the object. If the object is not persistent (the persistent id does not occur in the persistent store), this closure will be stored with the given persistent id. If the id refers to an object which is already persistent, however, then the previously stored value will be retrieved and used. This allows transparent access to the persistent store – it is not necessary to know whether or not a value is persistent – the persistent store is simply an extension of the in-memory heap that can be used as a long-term cache to avoid costly re-evaluation.

### 2.3 Dynamic Typing

When retrieving a value from persistent store using either `get` or `use`, it must be checked against the known static type for the value. One possibility is to use a `typeof` operator to produce a unique hash value for the type, which can be stored with the persistent value and checked against the fetched value.

```haskell
class Persistent a where
typeof :: Hash
...
This is not sufficient, however, for polymorphic functions or closures. In this case it is necessary to unify the statically determined type against the type returned from the persistent store. The work of Leroy and Mauny [13], on dynamic polymorphic types in CAML, may help here. We are investigating the problem at the moment, but believe it is not insurmountable, even if it proves necessary to incorporate a simple unification algorithm into the runtime system. Although such an operation may be relatively expensive, it only has to be done once as each object is moved from persistent store.

3 Implementation

The implementation uses technology taken from the GUM parallel runtime system for Glasgow Parallel Haskell [25]. This parallel technology, in turn, borrows heavily from conventional object store implementations. We are effectively therefore using standard techniques, modified by our experience with the parallel system (notably the fetching code), and adapted to suit the Glasgow Haskell Compiler.

The low-level implementation is tailored to suit the STG machine implementation [19]. However, the principal features of the implementation should be applicable to other systems.

3.1 The St Andrews POS

We use the lower level of the shadow-paged Napier POS [2], modified to improve efficiency. Disk accesses for neighbouring words of persistent store are reduced by using memory-mapping – words in the same disk page will already be paged into memory.

The basic operations provided from the system are:

- pid_t SHcreate_object(postore_t pos, int size): used to create new persistent objects.
- int SHread_word(postore_t pos, pid_t pid, int offset): reads a word from the persistent store at offset offset from persistent object pid.
- int SHwrite_word(postore_t pos, pid_t pid, int offset, int value): writes the word value to offset offset in the persistent object pid.
- pid_t SHread_key(postore_t pos, pid_t pid, int offset): like SHread_word except that it returns a pid rather than a data word.
- void SHwrite_key(postore_t pos, pid_t pid, int offset, int value): like SHwrite_word except that it writes a pid rather than a word.

The first problem to resolve is to construct a mapping between Haskell closures and POS objects. The general format of Haskell closures is given in Figure 1. Each closure comprises a fixed length header, whose structure is the same for all closure types, an additional variable length header, whose content depends on the closure type, a (possibly empty) list of pointers and a (possibly empty) list of non-pointer data.

The corresponding general format of POS objects is shown in Figure 2. The primary difference between this and the preceding Haskell closure format is that the POS object
makes no provision for the data in the Haskell header. Since this contains no pointers into the Haskell heap, we simply prepend this to the other non-pointer data in the POS object.

The mapping we have used is shown in Figure 3. It is necessary to add one word to indicate the size of the Haskell header, so that this can be easily separated from the non-pointer data.

### 3.2 Basic Values

The `get` or `use` operation introduces a PHETCHME closure that includes the persistent id of the object in persistent store. The name PHETCHME is by analogy with the GUM runtime system, where FETCHME closures are used to reference globally addressed objects in a parallel machine. The format of a PHETCHME closure is very simple and is shown in Figure 4.

When a PHETCHME closure is evaluated, it is replaced by the appropriate closure from the POS. If this closure contains any further pointers to persistent values, these
will be represented by additional new PHETCHME closures. Thus, structured objects such as lists are converted from their POS store format incrementally on demand.

It is important to note that this lazy conversion does not imply that additional disk accesses take place – because the POS is memory-mapped, an entire disk block of POS objects will be cached in local memory when any object is fetched from disk. The advantage of the incremental technique is in avoiding unnecessary conversion and type checking work, and in reducing garbage collection overheads, and perhaps also disk accesses, if only part of a persistent object is required. This is shown in Figure 5.

3.3 Preserving Sharing

To preserve in-memory sharing, we maintain a hash table mapping persistent ids. to local addresses. This is used when constructing a PHETCHME as part of a get or use operation: rather than creating a PHETCHME, an indirection is introduced to the existing local copy of the persistent value (which may by now have been evaluated). This is shown in Figure 6.

The hash table needs to be fully integrated with the garbage collector: although the hash table entries are not themselves garbage collection roots, they need to be updated if a compacting collector has moved them, and eliminated if they become garbage. In a generational collector it is possible (and highly desirable) to optimise the hash table update process so that only new generation entries are checked during a minor garbage collection. This has not yet been implemented, however.
3.4 Reflecting Updates to the POS

The description of the PHETCHME process in fact only applies to basic (data) values. Suspensions that are taken from the POS require a slightly different mechanism if the result of evaluation is to be reflected as changes in the POS.

In order to achieve this effect, we have introduced a new type of indirection closure: the PHETCHED closure type (shown in Figure 7). Non-normal form closures that are fetched from persistent store do not replace the original PHETCHME closure; rather it is transformed into a PHETCHED closure that points to the newly fetched value.

When a PHETCHED closure is evaluated, its address is recorded in an update frame, as happens in the STG machine for conventional indirections. The embedded closure is then evaluated. Following evaluation, the PHETCHME will be extracted from the update frame and its update code executed using the address of the new normal form as its argument. The effect of this is twofold: firstly the PHETCHME will be either overwritten with the new value or else turned into a normal indirection; and secondly, the new value will be written to the POS at the stored persistent id. This is shown in Figure 8.

Once again, it is important to observe the use of memory-mapping acting as a disk cache for the POS. The write will not actually take place immediately, but will happen following the next high-level commit operation. At this point, all updated pages...
will be written back to disk. If several changed values inhabit the same page, this may considerably reduce the amount of disk traffic that is required.

3.5 Functions and Partial Applications

Functions, partial applications and some suspensions require some special conversion when being read. This is need due to the fact that from one program to another the addresses of the code can change, for that reason, we need to calculate the actual address of the code each time that we read one function from th POS. However this can be solved fairly easy, storing the offset of the function code with respect to one know address in the module that we could recalculate each time that we run the program.

3.6 Cost Analysis

There is a clear performance trade between using a value from the POS, and using an in-memory value. We have not yet investigated techniques to avoid POS accesses when the cost of recomputation is cheap. Clearly cost analyses, such as the granularity analysis that we have proposed for a parallel context [9], will be important in determining whether it is worthwhile to use a persistent value in preference to the functionally equivalent closure that is provided by the use operation.

4 Related Work

4.1 Binary I/O

Binary I/O (i.e. non-textual I/O) has two main purposes: it can be used to exchange information more directly with programs written in other languages, such as C, without the overhead of writing parsing routines; and it can be used to improve the (space or time) efficiency of reading and writing data within an application.

Binary I/O systems are usually (though not necessarily) sequential. This is important since it degrades both access and update time. They also generally support only flat (fully evaluated) data structures, and does not handle sharing of values.
Much work has been done on the problem of providing binary I/O facilities in Haskell. While this was supported by the very first (prototype) Haskell compiler built on the LML compiler at Glasgow [10], subsequent compilers ignored the standard \texttt{Binary} class and \texttt{Bin} type, to the extent that it was removed from Haskell 1.3 and later variants pending a better design.

Removing the \texttt{Binary} class from the standard Haskell libraries has successfully cleared the field for new experimental implementations which possess a number of important advantages over the original design [23][28][29]. A major deficiency of the original design was that it failed to support sharing particularly well, being defined in an over-complicated compositional fashion. It also failed to support efficient compression or inter-language interfacing.

Reig’s implementation of binary I/O [23] builds on the PJama store [7][22]. Only allows pure data to be stored (that is, it is not possible to store functions or suspensions). Access is through conventional sequential file reading rather than through direct access, and a binary file must therefore be completely rewritten if it is updated.

Wallace’s implementation [23], in contrast, is designed to support efficient compression, in order to reduce disk usage. Like Reig’s approach, file access is sequential, and binary files are treated as complete units: it is not possible to update them incrementally.

Pil has been investigating first class I/O [20] and Dynamic and Type Dependent Functions [21] in Concurrent Clean [27]. At present, this implementation does not preserve sharing, neither does it allow suspensions nor functions to be stored persistently. It is planned, however, to implement all of these features in Clean 2.0.

Shields, Sheard and Peyton Jones [24] have been working on an approach to dynamic typing based on a concept they call \textit{staged computation} which extends a type system to types only known at run time (as in a persistent store) while still supporting type inference and polymorphism.

### 4.2 Persistent Functional Languages

A number of other functional persistent languages have been investigated and/or implemented. Perhaps the oldest was PSASL [8], a (lazy) persistent version of SASL [26] developed as an undergraduate project at St.Andrews. It allowed the top level environment to be named, stored and retrieved. This was quickly superceded by STAPLE [16] which had a more user friendly syntax based on that of an early version of a subset of Haskell. STAPLE allowed the retrieval and combination of modules from the persistent database to form the interactive user environment. Sharing and state of lazy evaluation were both preserved in the object store. Both PSASL and STAPLE were however byte-code interpreted. A new abstract PCASE [6] machine was employed to carry out reduction. It has the interesting property that all free variables of a function, even calls to top level functions, are treated uniformly by storing them in a heap object attached to the closure of the function being defined.

Amber [4] and CAML [13] are strict functional languages which support persistence by providing functions to export and import values to/from a file system. Such values are wrapped up in a dynamic type and a type check is made on import. CAML exhibits a loss of referential integrity in that if a value is read from persistent store more than once, the resulting values are copies and are not shared.
Poly/ML is a commercially available persistent functional system. Users can store Poly/ML objects in persistent databases. The run-time system transparently loads objects from persistent store into physical memory on demand. Databases may be arranged in a linked hierarchy, allowing a number of users to share common data and still develop applications in their private working environments. The system is non-lazy and sharing is not preserved.

5 Conclusions

We have reported on the implementation of a persistent lazy version of Haskell which has been brought about by integrating the back end of the Glasgow Haskell compiler (producing native code) with a run time system running on top of a St. Andrews persistent object store.

The system allows users to mark values for explicit persistence. We have identified and implemented a medium level interface which allows us to open, close and commit persistent stores and allows transparent access to persistent values via get and use.

The system allows the preservation of sharing and of the state of evaluation of persistent objects.

References

21. M. Pil Dynamic Types and Type Dependent Functions These Proceedings
<p>| Author Index | | | |
|--------------|--------------|----------------|
| Ábrahám-Mumm, E. | 94 | Guirado, R. | 209 |
| Affenzeller, M. | 36 | Gulías, V.M. | 540 |
| del Aguila, I. | 160 | | |
| Albertos, P. | 385 | Hannemann, U. | 94 |
| Aleman-Flores, M. | 316 | Hidalgo, M.J. | 612 |
| Alonso, J.A. | 612 | Hurtado, M.V. | 146 |
| Barreiro, M. | 540 | Ida, T. | 479 |
| Bienvenido, J.F. | 209 | von Issendorff, H. | 110 |
| Blanco Ferro, A. | 583 | Izumi, T. | 509 |
| de Blasio, G. | 300 | | |
| Borboni, A. | 352 | Jakac, W. | 52 |
| Börger, E. | 20 | Janoušek, V. | 369 |
| Bosch, A. | 160 | Jorge, S. | 540 |
| Buchberger, B. | 1, 469 | Kameyama, Y. | 509 |
| Campani, C.A.P. | 242 | | |
| Cañadas, J. | 160 | Labra Gayo, J.E. | 597 |
| Češka, M. | 369 | Lambán, L. | 555 |
| da Costa, S.A. | 457 | Loidl, H.-W. | 642 |
| Crespo, A. | 385 | Loogen, R. | 642 |
| Crespo, J.L. | 343 | López Brugos, J.A. | 233 |
| Cueva Lovelle, J.M. | 597 | Luengo Díez, M.C. | 597 |
| Di Nardo, E. | 67 | Machado, J.P. | 457 |
| Dierneder, S. | 223 | Maravall, D. | 437 |
| Diverio, T.A. | 457 | Marin, M. | 479 |
| Domínguez, C. | 555 | Marin, R.L. | 160, 209 |
| Dreiseitl, S. | 52 | Martín-Mateos, F.J. | 612 |
| Du Bois, A.R. | 570 | Martín-Rubio, F. | 196 |
| | | Mauerkirchner, M. | 172 |
| Faglia, R. | 352 | Meier, A. | 494 |
| Fernandez Lopez, P. | 328 | Meine, S. | 45 |
| Flores-Parrá, I.M. | 209 | Menezes, P.B. | 242, 285, 457 |
| Frappier, M. | 421 | Mittelmann, R. | 36 |
| Freire Brañas, J.E. | 583 | Mora, E. | 343 |
| Freire Nistal, J.L. | 583 | Moraes Claudio, D. | 285 |
| | | Moreno-Díaz, R. | 300 |
| Garcia Baez, P. | 328 | Moreno-Díaz, R. jr. | 316 |
| Garcia-Cabrera, L. | 270 | | |
| Garcia Rosa, R. | 450 | Németh, G. | 139 |
| González Rodríguez, B.M. | 597 | Nobile, A.G. | 67 |
| González Cobas, J.D. | 233 | Notare, M.R. | 457 |</p>
<table>
<thead>
<tr>
<th>Author</th>
<th>Page Numbers</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Palma-Méndez, J.T</td>
<td>196</td>
<td>Sánchez Penas, J.J.</td>
</tr>
<tr>
<td>Paniagua-Aris, E.</td>
<td>196</td>
<td>dos Santos Leal, L.A.</td>
</tr>
<tr>
<td>Pareja, C.</td>
<td>627</td>
<td>Sato, M.</td>
</tr>
<tr>
<td>Parets-Llorca, J.</td>
<td>146, 270</td>
<td>Scharinger, J.</td>
</tr>
<tr>
<td>Pascual, V.</td>
<td>555</td>
<td>Scheidl, R.</td>
</tr>
<tr>
<td>de Pedro, T.</td>
<td>450</td>
<td>Schwaninger, M</td>
</tr>
<tr>
<td>Peña, R.</td>
<td>627</td>
<td>Segura, C.</td>
</tr>
<tr>
<td>Pichler, F</td>
<td>36</td>
<td>Shimogawa, T.</td>
</tr>
<tr>
<td>Pirozzi, E.</td>
<td>67</td>
<td>Sorge, V.</td>
</tr>
<tr>
<td>Pointon, R.F.</td>
<td>642</td>
<td>Stauner, T.</td>
</tr>
<tr>
<td>Pollet, M.</td>
<td>494</td>
<td>St-Denis, R.</td>
</tr>
<tr>
<td>Priebe, S.</td>
<td>642</td>
<td>Steffen, M.</td>
</tr>
<tr>
<td>Pröll, K.</td>
<td>52</td>
<td>Suarez Araujo, C.P.</td>
</tr>
<tr>
<td>Quesada-Arencibia, A.</td>
<td>316</td>
<td>Suzuki, T.</td>
</tr>
<tr>
<td>Quintela, J.</td>
<td>657</td>
<td>Táboas, M.</td>
</tr>
<tr>
<td>Raya, A.</td>
<td>437</td>
<td>Trinder, P.W.</td>
</tr>
<tr>
<td>Resconi, G.</td>
<td>352</td>
<td>Tünner, S.</td>
</tr>
<tr>
<td>Ricciardi, L.M.</td>
<td>67</td>
<td>Vieira Toscani, L.</td>
</tr>
<tr>
<td>da Rocha Costa, A.C.</td>
<td>570</td>
<td>Vojnar, T.</td>
</tr>
<tr>
<td>Rodríguez-Fórtiz, M.J.</td>
<td>270</td>
<td>Windsteiger, W.</td>
</tr>
<tr>
<td>Rodríguez-Millán, J.</td>
<td>393, 405</td>
<td></td>
</tr>
<tr>
<td>Rubio, F.</td>
<td>627</td>
<td></td>
</tr>
<tr>
<td>Rubio, J.</td>
<td>555</td>
<td>Zorrilla, M.E.</td>
</tr>
<tr>
<td>Ruiz-Reina, J.L.</td>
<td>612</td>
<td></td>
</tr>
</tbody>
</table>