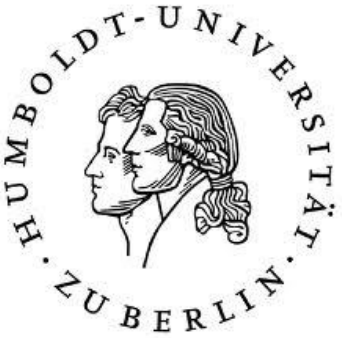
CONCURRENCY  SPECIFICATION AND PROGRAMMING

PROCEEDINGS OF THE INTERNATIONAL WORKSHOP

CS&P'2013

Warsaw, 25-27 September 2013

Edited by:
Marcin Szczuka
Ludwik Czaja
Magdalena Kacprzak



Warsaw Center
of Mathematics
and Computer Science

Published by Białystok University of Technology 2013

ISBN 978-83-62582-42-6

Preface

This volume contains the Proceedings of 22nd Concurrency, Specification and Programming (CS&P) Workshop held on September 25-27, 2013 in Warsaw.

There were 48 submissions. Each submission was reviewed by two program committee members. The committee decided to accept 40 papers.

The Workshop was initiated in the mid 1970s by computer scientists and mathematicians from Warsaw and Humboldt Universities, as Polish-German annual meetings. They were suspended for some years in the 1980s and reactivated in 1992. Thereafter, the Workshop, organized alternatingly by the Institutes of Informatics and Mathematics of the University of Warsaw and the Institute of Informatics of Humboldt University in Berlin on the basis of an exchange program, has been given the name CS&P.

It should be mentioned that the CS&P meetings, initially purely bilateral, since 1992 have developed into events attended by participants from a number of different countries beside Poland and Germany. The present CS&P'2013 meeting attracted contributors from: Canada, Egypt, France, Germany, Italy, Nepal, The Netherlands, Poland, Russia, Serbia, Slovakia, Sweden, Turkey, United Kingdom, United States, and Vietnam.

The organisation of this year's CS&P would not be possible without the resources and financing provided by several institutions. We would like to thank the Faculty of Mathematics, Informatics and Mechanics of the University of Warsaw and the Institute of Informatics of the Humboldt University of Berlin for the continuing financial and organisational support provided to CS&P over last twenty-two years. The essential financial backing received from the Warsaw Center of Mathematics and Computer Science made the organisation of CS&P 2013 possible. Our thanks go to the Białystok University of Technology for providing the means for publishing this proceedings volume. Last, but not the least, we are grateful for the significant financial support provided by the Vistula University in Warsaw.

September 2013
Warsaw, Poland

Marcin Szczuka
Ludwik Czaja
Magdalena Kacprzak

Program Committee

Hans-Dieter Burkhard	Humboldt Universität zu Berlin
Ludwik Czaja	The University of Warsaw and Vistula University
Anna Gomolińska	University of Białystok
Monika Heiner	Brandenburg University at Cottbus
Magdalena Kacprzak	Białystok University of Technology
Anh Linh Nguyen	The University of Warsaw
Hung Son Nguyen	The University of Warsaw
Wojciech Penczek	Institute of Computer Science, Polish Academy of Sciences
Lech Polkowski	Polish-Japanese Institute of Information Technology
Louchka Popova-Zeugmann	Humboldt Universität zu Berlin
Holger Schlingloff	Fraunhofer FIRST and Humboldt Universität zu Berlin
Serhat Seker	Istanbul Technical University and Vistula University
Andrzej Skowron	The University of Warsaw
Zbigniew Suraj	University of Rzeszów
Marcin Szczuka	The University of Warsaw
Matthias Werner	TU Chemnitz
Karsten Wolf	University of Rostock

Additional Reviewers

The members of the Program Committee want to thank the following persons for contributing to the review process of CS&P 2013.

Marek Bednarczyk	Anna Sawicka
Piotr Chrząstowski-Wachtel	Martin Schwarick
Mario Haustein	Jarosław Skaruz
Andrzej Jankowski	Maciej Szreter
Michał Knapik	Dominik Ślęzak
Irina Lomazova	Wojciech Świeboda
Artur Męski	Józef Winkowski
Christian Rohr	Bożena Woźna-Szcześniak
Andrzej Salwicki	Olena Yaskorska
Matteo Sammartino	

Table of Contents

DNA Tiles, Wang Tiles and Combinators

Marco Bellia and M. Eugenia Occhiuto

Dipartimento di Informatica, Università di Pisa, Italy
{bellia,occhiuto}@di.unipi.it

Abstract. In this paper we explore the relation between Wang Tiles and Schonfinkel Combinators in order to investigate Functional Combinators as an programming language for Self-assembly and DNA computing. We show: How any combinatorial program can be expressed in terms of Wang Tiles, and again, how any computation of the program fits into a grid of tiles of a suitable finite, tile set, and finally, how a program for Self-assembly DNA computing can be obtained. The result is a general methodology that, given any computable function, allows to define a Self-assembly program that can be used to construct the computations of the function

1 Introduction

In the last decade, one of the emerging approaches [1] to DNA Computing, is Self-Assembly [2]. It describes a computation in terms of a process in which small components, autonomously and automatically, assemble into larger, more complex, structures [3–5]. The assembly is based on the Watson-Crick complementary law and is effectively governed by various bio-chemical techniques [6]. However, in terms of computable functions, in the Self-Assembly computation process, it is possible to recognize:

(a) The computed application. The computed application is expressed by the small components to be assembled. In particular, these components include a representation for the function arguments, if any, i.e. the inputs of the application, and a representation for the function to be applied.
(b) The computation. The larger and more complex structures, that result at the end of the SelfAssembly process, form the effective computations. Each of such structures can be read as the complete trace of a computation, from its start to its end.

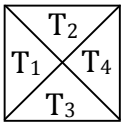
Various kinds of DNA Tiles has been introduced, in the years, in the various proposals, to be used as the small components of point (a) [7, 8]. In [9] the relation between DNA Tiles (TX, triple crossover, molecules) and Wang Tiles has been used to show how to simulate finite state automata with output, i.e. a transducers, in Wang Tiles. Moreover, by using compositions of transducers and the relation with Wang Tiles, [9] shows how the computation of general recursive

functions can be expressed using self-assembly. This allow to use the formalism of general recursive functions as a programming language for DNA computing. With the same aims, [10] introduced DSL as language for programming with the DNA Tiles of the aTAM model [11].

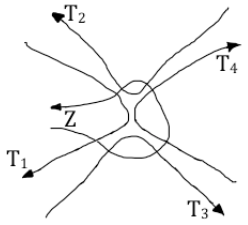
In this paper we explore the relation between Wang Tiles [12] and SKI combinators [13, 14] in order to investigate Functional Combinators [15, 16] as an High-/Intermediate level, programming language for Self-Assembly computations. The result is the definition of a language for Self-Assembly, SKI-Tiles, and of a general methodology that, given any computable function, allows to define a program, in SKI-Tiles, that compute each application of the function, by using Self-Assembly computations.

2 Wang Tiles

Wang Tiles [12] were introduced in 1961. It is a formal system based on the notion of tile. A tile may be graphically represented by a unit square with colored sides from a (possibly, denumerable) set T of distinct colors. Figure 1.a shows the form of a tile such that: *West* side has color T_1, *north* has color T_2, *south* has color T_3 and *east* has color T_4. Tiles must be arranged side by side on the plane (*computation grid*) in a way that adjacent tiles must have the adjacent side of the same color, see Figure 5: We will name this operation *Wang-arrangement*. The interest is on the set $\mathcal{F}$ of all the finite sets of distinct tiles: What tile sets of $\mathcal{F}$, can cover the infinite plane by using Wang-arrangement on copies of the tiles of the set, obtained by translation (no rotation, no reflection). In 1963, Wang showed that to each Turing Machine M corresponds a finite set $T_M \in \mathcal{F}$ such that the computation of M on a tape D can be emulated by a covering, with the (copies of) tiles of T_M, of a plane containing an initial row of tiles that describes D. Finally, Wang proved that the halting problem of Turing Machines can be reduced to the undecidability, for finite tile sets, of covering the infinite plane.



a. Wang Tile: A unit square with the sides colored by T1, T2, T3, T4



b. DNA Tile: 4 strands of DNA, T1, T2, T3, T4, are kept together by a suitable DNA structure, Z.

Fig. 1. Wang Tiles and DNA Tiles

3 SKI Combinators

3.1 The monoid SKI

SKI Combinators [14] is a formal system that expresses all the computable functions[1] without requiring any (bound) variable and by using only one operation: the monadic, functional *application*. Hence, it is the monoid[2] Σ, below:

$$\Sigma = S|K|I|\Pi|X|\Sigma\Sigma$$

where the application is represented by juxtaposition of a (left) term, representing a monadic function, to a the (right) term, representing the argument. Currying, higher order functions, and left associativity of application are provided for non-monadic functions. The symbols S, K, I are combinators (but other ones could be added in [15]), X is a set of (free) variable symbols, Π is a set of constant symbols. The terms of Σ are also called *combinatorial* terms, and the terms built by using the application operator, namely those in $\Sigma\Sigma$, are called (combinatorial) *application* term. Combinators obey to the following *application* laws, for $a, b, c \in \Sigma$:

$$I\ a == a$$
$$K\ a\ b == a$$
$$S\ a\ b\ c == a\ c\ (b\ c)$$

3.2 Bracket Abstraction and Bound Variables

The combinators S, K, I express the bracket abstraction in the following way (other characterizations are in [15]). Let $a \in \Sigma$ be any term, possibly containing a (free) variable $x \in X$. Then, we define the bracket abstraction of $a \in \Sigma$ with x, written $[x]a$, be the term $b \in \Sigma$ such that: $b\ x = a$. Such a term[3] always exists in Σ and can be obtained by using the following rules:

$$[x]x = I$$
$$[x]u = Ku, \text{ for } u \in \{S, K, I\} \cup \Pi \cup X \text{ and } u \neq x$$
$$[x](a\ b) = S([x]a)([x]b)$$

Hence, all the closed terms of the calculus are all the terms of Σ that do not contain variables.

3.3 Program, Computation, Recursion

Noting that in the application $\Sigma\Sigma$, there is no distinction between the terms that are functions (driving the computation to be done) and those that are arguments (forming the values). Any term becomes the function to be applied, when it is

[1] in its original formulation, in 1924, by Moses I. Schonfinkel, [13], the combinator "I", which could be expressed through SKK, was replaced by the combinator "U", in order to express first order predicates without the use of bound variables.

[2] Also Wang Tiles is a monoid, on Tiles as terms, with Wang-arrangement as the only operation

[3] moreover, for all terms $c \in \Sigma$, we have $b\ c = a[x \leftarrow c]$, i.e. b behaves like one λ-abstraction and when applied to c reduces according to Church's β-axiom [17]

in the left side, whilst it behaves as a value when it is in the right side of the application. A (combinatorial) program is any term of Σ. A program computes according according to the *application* laws of the SKI calculus. In order to obtain a notion of computation, we can encapsulate the application laws into the reduction system obtained by the binary relation on combinatorial terms, $\rightarrow$, defined in the following way. Relation $\rightarrow$ is called *combinatorial reduction.*

Definition 1 ($\rightarrow^*$). *Relation $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow^*$.*

$$\frac{\text{I a} == \text{a}}{\text{I a} \rightarrow \text{a}} \qquad \frac{\text{K a b} == \text{a}}{\text{K a b} \rightarrow \text{a}} \qquad \frac{\text{S a b c} == \text{a c (b c)}}{\text{S a b c} \rightarrow \text{a c (b c)}}$$

$$\frac{\text{a} \rightarrow \text{a'}}{\text{a b} \rightarrow \text{a' b}} \qquad \frac{\text{b} \rightarrow \text{b'}}{\text{a b} \rightarrow \text{a b'}}$$

Given a program a, a computation of a is any sequence, for $n \geq 0$[4]:

$$a \rightarrow a_1 \rightarrow ... \rightarrow a_n$$

Whenever a_n is such that for no $b \in \Sigma$, $a_n \rightarrow b$, then we say that: Program a has one *terminating* computation; $a \rightarrow a_1 \rightarrow ... \rightarrow a_n$ is a *terminating* computation of a; Program a computes a_n, or equally, a_n is the "value" computed by a. Relation $\rightarrow$ has Church-Rosser confluence property, since if $a \rightarrow a_1 \rightarrow ... \rightarrow a_n$ is a *terminating* computation of a then $a_n \equiv b_m$ for any other *terminating* computation $a \rightarrow b_1 \rightarrow ... \rightarrow b_m$ [18]. However, Σ contains *nonterminating* programs. As a matter of the fact consider the term Ψ of Definition2.

Definition 2 (The Kleene fixed-point combinatorial program calculator, Ψ). *Let $R \equiv S(S(KS)(S(KK)I))(K(SII))$. Then, $\Psi \equiv SRR$ is a combinatorial program. Moreover, Ψ is such that, for all pairs of terms $G, a \in \Sigma$, the following holds:*

$$(*) \qquad \Psi\ G\ a \equiv G\ (\Psi\ G)\ a$$

The proof of (*) is a trivial exercise. Ψ points out the elegance with which Schonfinkel monoid expresses the computable functions. In particular, Ψ introduces *recursively defined* terms on one hand, and computes the least fixed point of them, on the other hand. However, in Section 6, we use term equations for dealing with recursive definitions, because Self-assembly computation has a notion of term replacement that already support recursive definitions.

4 The Approach

We start introducing the structures and the properties that the Wang tiles must have in order to be used for expressing the combinatorial terms and their computation. Then, we show how to use such structures in order to get the definition and the computation of any combinatorial program.

[4] Obviously, $n = 0$ means that for no $b \in \Sigma$, $a \rightarrow b$

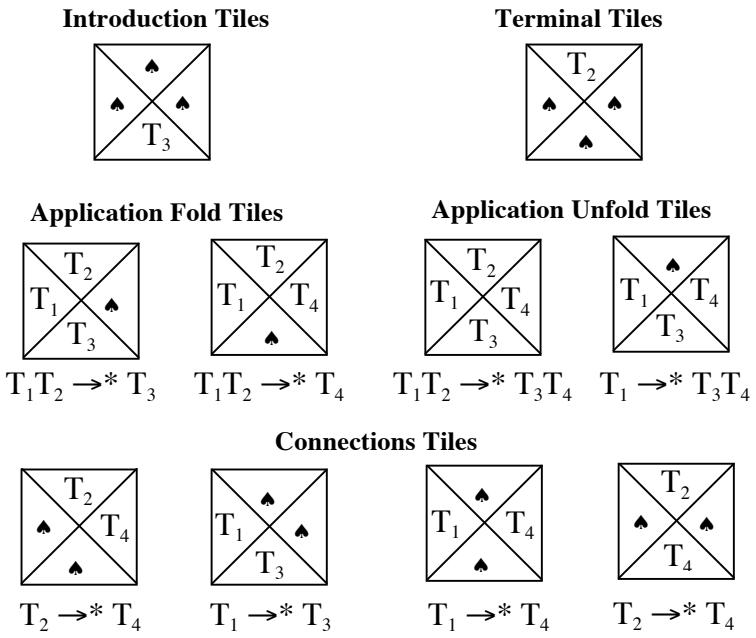
4.1 SKI-Tiles: A formalism of Wang Tiles for combinatorial programs

The colors that may occur in the tiles, are the combinatorial terms (of Σ): Different terms are different colors. In addition, a special color ♠ is used for combining the terms within a tile and for arranging the tiles in the computation grids. The sides of a tile may be colored with an input (i.e. the right part of an application term) or with a function (i.e. the left part of an application term) or with an output (the result of an application) or finally, with a connection term (which allows to arrange together distinct tiles and distinct parts of the computation grid). When more different colors occur in a tile, their arrangement in the tile sides obeys properties based on the combinatorial reduction. According to how colors are used in the tile sides, the tiles fall in one of the following five classes, shown in Figure 2.

- **Introduction Tiles** are the tiles that introduce the components, namely function and arguments, see Figure 2, of the computation to be made. These tiles may occur in the top line of a computation grid. No, specific, property is required to the color used in the tile.
- **Terminal Tiles** are the tiles that collect the result of a computation, see Figure 2. These tiles may occur in the bottom line of a computation grid. No, specific, property is required to the color used in the tile.
- **Application Fold-tiles** deal with the reduction of applicative terms that do not require any reduction on their subterms. Color T_1 is used for the function, color T_2 for the argument, whilst colors T_3 or T_4 for the reduced term: It obeys the properties that are indicated at the bottom of the tile in Figure 2.
- **Application Unfold-tiles** deal with the reduction of applicative terms that require some subterm reduction. Color T_1 is used for the function and color T_2, if any, for the argument, exactly as in the fold-tile s, but the reduced term is an application $T_3\ T_4$. This tile structure allows to use two distinct tiles, one for reducing the color T_3 and one for reducing the color T_4, separately. Constraints on the colors are indicated at the bottom of the tile in Figure 2.
- **Connection Tiles** they furnish tiles that are suitable to connect different parts of the computation grids and in some cases they may involve simple term reductions. Constraints on the colors are indicated at the bottom of the tile in Figure 2.

4.2 Soundness of SKI-Tiles

Apart from the introduction and the terminal tiles, all other tiles of SKI-Tiles, are combinatorial term reductions of $\rightarrow^*$. The Wang-arrangement operation corresponds to the (reflection and) transitivity of $\rightarrow^*$. Hence, computation grids contain only sound reductions on the combinatorial terms that are involved in the tiles, and in particular from the terms of the introduction tiles up to the term of the terminal tile of the grid.



Legenda. T_1, T_2, T_3, T_4 are colors for combinatorial terms; $\rightarrow^*$ is the reflexive, transitive closure of the combinatorial reduction.; The colors must obey the property, if any, that is put below the tile.

Fig. 2. The classes of tiles of SKI-Tiles for the Combinatorial Terms

4.3 The Computation Grids of S, K and I in SKI-Tiles

The combinators are completely defined in the Wang Tile formalism by the computation grids in Figure 3, for I and K, and in Figure 4, for S. The grid for I consists of only one fold-tile that switches the input on the output. The grid for K consists of 4 tiles: The tile on the left top corner is a fold-tile that collects the first argument and has "a" as output. The tiles on the right top and the left bottom corners are connection tiles. They are used for connecting the fold-tile on the bottom right corner of the grid. The latter tile contains, as output, the output of the grid. Actually, for S we give two grids of 9 tiles: Both are correct. The two grids differ for the tile on the right bottom corner. Both contains the same fold-tile on left top corner, and the same 7 connection tiles. The other tile is a fold-tile in the left grid, whilst it is an unfold-tile in the right one. The choice of the right grid may depend on the input terms, if a and b do not require any reduction then the left grid may be the best grid to be used. The grids in Figure

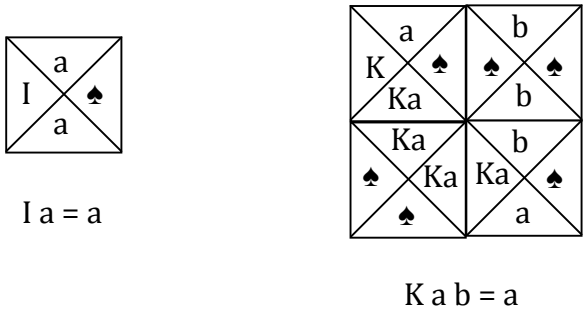


Fig. 3. The computations of K and I in the SKI-Tiles formalism

3, and in Figure 4 are defined for being used as grid components, hence do not contain any introduction or terminal tile.

Theorem 1. *The SKI calculus can be expressed in Wang Tiles*

Proof. The proof is easily obtained by induction on the pure combinatorial terms (i.e. $\Sigma-(\Pi+X)$) and by using suitable grid compositions. The extension to the entire Σ comes immediately since each symbol in $\Pi+X$ is an uninterpreted symbol.

The Theorem above is not surprising since Wang's result [12], but the theorem furnishes a constructive proof and a concrete way to do it. The next section shows how the approach effectively applies in a computation.

5 Applications and Examples

The section shows how the approach effectively applies in a computation. Consider the function $Proj_2^4$ that selects the second argument, from a sequence of four arguments. We write a program that, given four arbitrary terms, c_1, c_2, c_3, c_4, as inputs, computes c_2 as output. In combinatorial programming, the program can be obtained two different ways, according to a use of combinatory programming as an intermediate level or as a higher level programming language. We consider both view and for each of them we show the corresponding computation grid in the tile formalism of the previous section.

5.1 Combinatorial programming at an Intermediate Level

This way of programming is widely influenced by the use of combinators in the implementation of functional languages [19]. In order to obtain a combinatorial term for $Proj_2^4$, we start giving a formulation of $Proj_2^4$ in a functional language. In this case, we can express it by the lambda term:

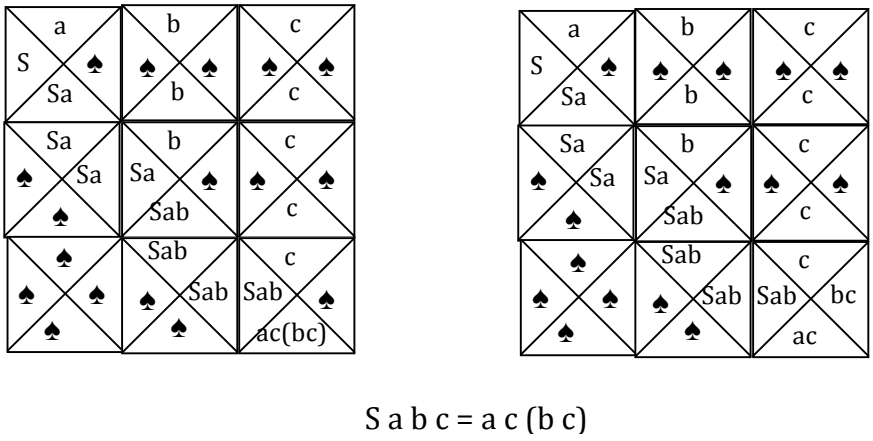


Fig. 4. Two different computations of S in the SKI-Tiles formalism

$\lambda\ x_1.\lambda\ x_2.\lambda\ x_3.\lambda\ x_4.\ x_2.$

Then, by using the technique for removing bound variables from lambda terms[5], we obtain the combinatorial term:

$K(S(KK)(S(KK)I))$.

Eventually, we have the combinatorial term T and its computation grid, in Figure5.

5.2 Combinatorial programming at an High Level

This way of programming uses the possibility of introducing new combinators and super combinators [16] in order to obtain a more expressive and neat solution to a possibly, more general problem than the given one. In this case, the problem may be solved by using a family, $Proj = \{f_n\colon D^n \to D\}$, of curried functions, each function being indexed by the arity. We can express each function of $Proj$ by the following combinatorial term: $T_p = K^{i-1}(WIK^{n-i})$, where n is the arity of f_n, $0 < i \leq n$ is the position of the argument to be selected, I is the corresponding combinator of SKI calculus. Finally, $K^m g = K(K^{m-1}g)$ is a variant of combinator K (for $m > 1$), whilst W is an additional combinator that obeys the following application law: $Wabc = b(ac)$. Then, the combinatorial program is now expressed by $T = (((K(WIK^2)c_1)c_2)c_3)c_4$, and its computation grid can be obtained by using the same methodology of Section 5.1.

[5] it roughly corresponds [15, 19] to the computation of $[x_1]([x_2]([x_3]([x_4]x_2)))$

6 Self-assembly Computations with SKI-Tiles

This section discusses the formalism of SKI-Tiles in the context of the Self-Assembly programming and extends the formalism with the notions of program and of computation of the Self-Assembly programming paradigm.

6.1 Wang Tiles vs. Self-Assembly

Wang Tiles and Self-Assembly share the same fundamental operation for connecting the tiles: Wang-arrangement. Nevertheless, there is a subtle but relevant

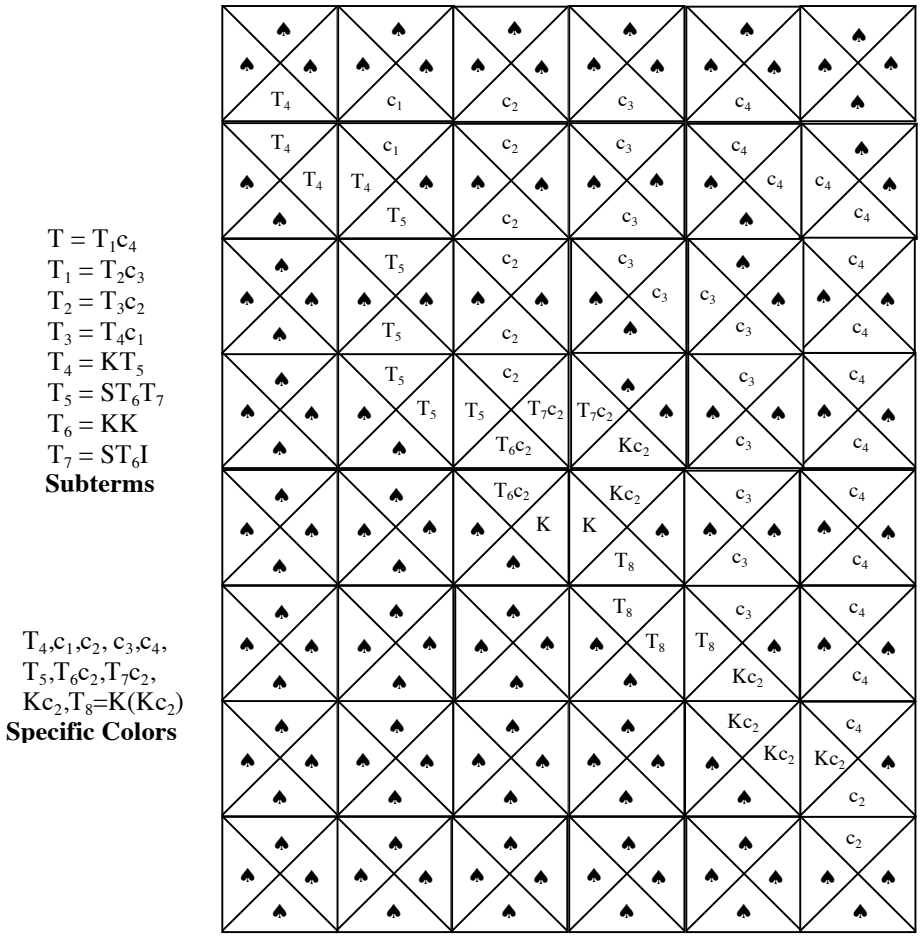


Fig. 5. The computation grid of $T = (((K(S(KK)(S(KK)I))c_1)c_2)c_3)c_4$

difference. The Wang formalism neither has a notion of program nor of computation: The aim is the construction of some computation grid that must be assembled with the tiles of a given tile set. Differently, Self-assembly is a programming paradigm with a notion of program, semantics and computation, that consider all the grids that can be assembled by applying Wang-arrangement to the tiles of the program.

6.2 The SKI-Tiles language for Self-Assembly programming

The section formalizes the notions of program and of computation in order to make SKI-Tiles a language for Self-Assembly programming. Then, it introduces a (combinatorial) formulation of conditional, booleans and numbers for the use of programs, for arithmetic programming, in SKI-Tiles.

Chemical Context Let $H \equiv \{T, g, \tau\}$ be triple defining the physics of molecular self-assembly [5] of the programs. We assume that for all programs, the set of color T, the binding strength function g and the temperature parameter τ are chosen in a way that Wang-arrangement can apply always and only when the tiles abut on sides that are colored by a same color.

Programs. A program is a finite sequence of quadruples of the form (T_1, T_2, T_3, T_4). The use of quadruples introduces a convenient, linear notation for tiles [4], in particular the quadruple (T_1, T_2, T_3, T_4) corresponds to the tiles in Figure 1 provided that T_1, T_2, T_3, T_4 are colors of the SKI-Tiles formalism.

Semantics. Let P be a program. The semantics of P is the set of all *sound* computation grids that can be obtained from P by τ-stable derivation.

Seed and τ-stable Derivation. Let P be a program. Let s be the seed tile of A_0, i.e. the only tile of the grid A_0. Then, $A_0 \rightarrow_P ... \rightarrow_P A_n$ is a computation. Moreover, $\rightarrow_P$ is the τ-stable Derivation (of P in H) and is such that $A \rightarrow_P B$ if and only if B is obtained from A by Wang-arrangement, with a (copy of a) tile of P, which satisfies the chemical context H.

Sound Computation Grid. Unfortunately, the Wang-arrangement does not always produce meaningful computation grids when unfold tiles are admitted. Hence, a computation grid is said sound if and only if the property hold:

- The topmost row contains only introduction tiles and only one of them, the seed, has color $T_3 \neq$ ♠, *and*
- The bottom row, if any, contains only terminal tiles and only one of them has color $T_2 \neq$ ♠, *and*
- The leftmost column, if any, contains only tiles with a ♠ as east side, and
- The rightmost column, if any, contains only tiles with a ♠ as west side, and
- No unfold-tile occurs in the grid, *or*
- The unfold-tiles satisfy the sub-grid property.

Definition 3 (Quasi-grids.). *A quasi-grid is a $n \times m$ grid of tiles with $n, m > 1$ and such that: The tiles of the first column, exception for the top tile, have a ♠ as west side; The tiles of the first raw, exception for the leftmost one, have a ♠ as north side; The tiles of the last column, exception for the bottom tile, have a ♠ as east side; Finally, the tiles of last raw, exception for the rightmost one, have a ♠ as south side.*

Definition 4 (Sub-grid Property.). *Let G be a computation grid and A be an unfold-tile of G. Then A satisfies the sub-grid property if A is the left top corner of a quasi-grid of G.*

It is worth noting, that the unfold-tiles involve the reduction of combinatorial terms of the form $a\ b$ with the aim of reducing, firstly, a to some a' and b to some b', separately, and then, of reducing $a'\ b'$. Hence, this leads to a sub-computation that behaves like a quasi-grid. As an example, the tile $A \equiv (T_5, c_2, T_6c_2, T_7c_2)$ (4th tile from the top, of the 3rd column, from the left) of computation grid in Figure 5, is an unfold-tile which satisfies the sub-grid property: In particular, the tile is the left top corner of a quasi-grid of 4 tiles. Moreover, even if the tile $B \equiv (T_7c_2, c_3, c_2, \spadesuit)$ was in the program, the sub-grid property would forbid to put it on the east side of A, i.e. the replacing of $(T_7c_2, \spadesuit, Kc_2, \spadesuit)$ with B.

Booleans, Conditional, Numbers in Functional Programming. We list some usefull functional structures for arithmetic calculus, including Barendregt numbers [17] and use them in writing arithmetic programs in functional programming[6]:

- $True \equiv \lambda x.\lambda y.\ x$
- $False \equiv \lambda x.\lambda y.\ y$
- Conditional is implicitly expressed by $True$ and $False$
- $Pair \equiv \lambda x.\lambda y.\lambda z.\ z\ x\ y$
- The number 0 is $[0] = Pair\ True\ (Pred\ [0])$ [7]
- The successor of n is $[n+1] = Pair\ False\ [n]$, for $n > 0$
- Program for Test on 0: $Zero = \lambda x.\ x\ True$
- Program for Predecessor: $Pred = \lambda x.\ x\ False$
- Program for Addition: $Add = \lambda x.\lambda y.\ (Zero\ x)\ y\ (Pair\ False\ (Add\ (Pred\ x)\ y))$
- Program for Product: $Prod = \lambda x.\lambda y.\ (Zero\ x)\ x\ (Add\ (Prod\ (Pred\ x)\ y)\ y)$
- Program for Factorial: $Fact = \lambda x.\ (Zero\ x)\ [1]\ (Prod\ x\ (Fact\ (Pred\ x)))$

Additional Combinators for SKI-Tiles This section extends the set of combinators, to include some combinators, C, B, P, that are of general use in combinatorial programming [15], and some other that are convenient in expressing, in SKI-Tiles, the programs listed above.

[6] We use λ-notation to express the terms: In particular application is term juxtaposition, is left associative, and has precedence on abstraction. Finally, recursive definitions use equations of the form $x = E$, where E is an abstraction and x is a functional variable that cannot occur bound in E

[7] In the original formulation [17], [0] is $Pair\ True\ False$. Here, we extend the domain of the numbers with the *undefined* value, $Pred[0]$.

- Left application combinator is B: $B\ a\ b\ c == a\ c\ b$
- Right application combinator is C: $C\ a\ b\ c == a\ (b\ c)$
- Combinator for Pair is: $P\ a\ b\ c == c\ a\ b$
- Combinator for True is: $T_b\ a\ b == a$
- Combinator for False is: $F_b\ a\ b == b$
- Combinator for Pred is: $P_r\ a == a\ F_b$
- Combinator for test on 0 is: $Z\ a == a\ T_b$
- Combinatorial term for 0 is: $[0] = P\ T_b\ (P_r[0])$
- Combinatorial term for $n+1$ (with $n > 0$) is: $[n+1] = P\ F_b\ [n]$
- Combinatorial Program for Addition:
  $+ = S(CS(B(CC(CZI))I))(C(C(PF_b))(B(CC(C+(CP_rI)))I))$
- Combinatorial Program for Product:
  $\star = S(BC(S(CZI)I))(B(CS(C(C+)(CC(C\star(CP_rI)))))I)$
- Combinatorial Program for Factorial: $F_t = S(B(CZI)[1])(S(C\star I)(CF_t(CP_rI)))$

Let $\mathcal{N}$ be the the minimal set such that $\mathcal{N} = \{[0], PF_b[n] \mid [n] \in \mathcal{N}\}$. Then, $\mathcal{N}$ is the set of (the combinatorial terms for) numbers, whilst $\mathcal{B} \equiv \{T_b, F_b\}$ is the set of terms for booleans.

Self-Assembly Programs in SKI-Tiles Programs in SKI-Tile, for the predecessor, the addition, and the factorial, have the listing in Figure 6: The listing contains only the application tiles. Each program must be completed adding (as by default) the suitable, connection tiles, introduction tiles, and terminal tiles. About the connection tiles, each program includes connection tiles of whatever kind but that involve only one of the program colors (the program colors are all the colors, but ♠, that occur in the program). For instance, the connection tile $(+(Pr\ [2])m, ♠, +(Pr\ [2])m, ♠)$ is included, but $(+(Pr\ [2])m, ♠, +[1]m, ♠)$ is not, in the program for $+$. About the terminal tiles, these programs compute numbers, hence numbers are the only colors that can be contained in a terminal tile to be included in the programs. Finally, the introduction tiles must contain only colors for numbers and for the name of the program.
In SKI-Tiles, the colors are the combinatorial terms that occur in the program tiles. But the terms occurring in the tiles of the programs in Figure6are not always combinatorial terms because of the the symbols n, m, b. Symbols n ad m are variables ranging on a finite subset of $\mathcal{N}$, whilst b is ranging over $\mathcal{N}$, and the tiles of the programs are in fact, tile schemata.
Finally, note that the program P_r has no computation grid for computing $P_r[0]$.

7 Conclusions

We have investigated three computation formalisms, Wang Tiles, Schonfinkel Combinators and Self-Assembly Programming, in order to define a high level programming language for Self-assembly and DNA computing. We have defined the formalism SKI-Tiles: It states the structures and the properties that the Wang tiles must have in order to express combinatorial terms and the computation of combinatorial programs in the Wang Tiles formalism. We have discussed

the soundness of SKI-Tiles. We have used the formalism SKI-Tiles as the kernel of a language for Self-Assembly programming. In order to do it we have revised the notion of computation and introduced the sound computation grid. We called this language the SKI-Tiles language. We have shown programs for Self-Assembly programming that are written in the SKI-Tiles language. These programs compute a partial function for predecessor on naturals, and functions for addition and factorial.

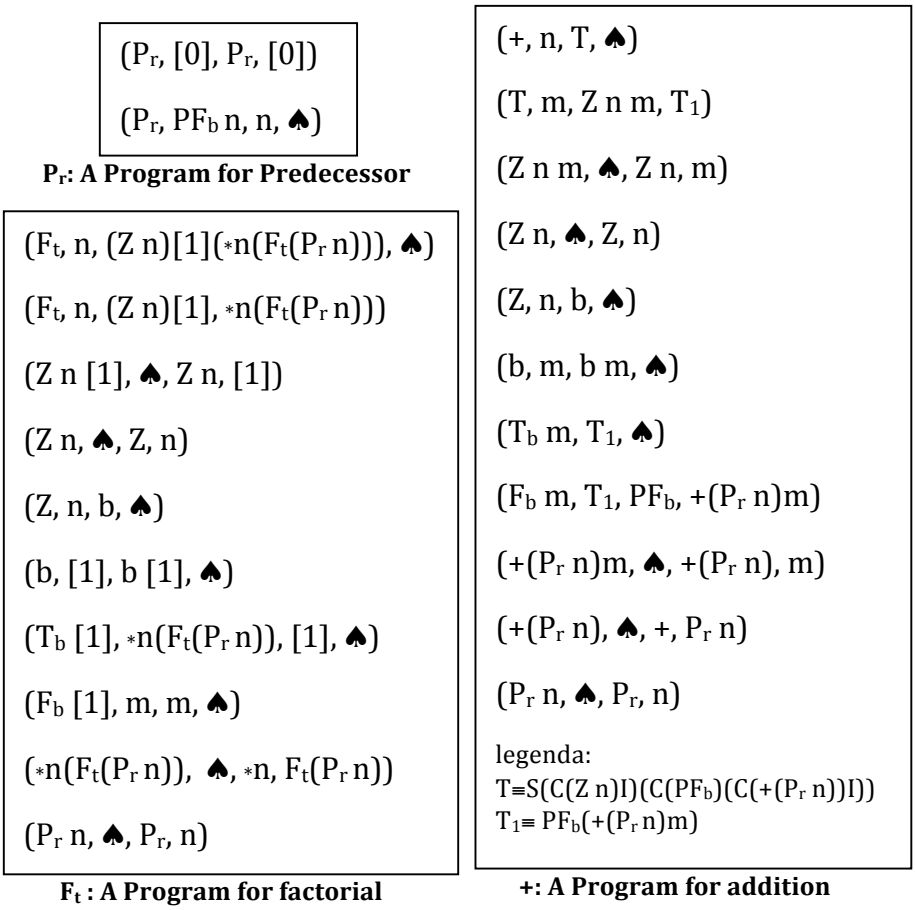


Fig. 6. Self-Assembly Programs for Predecessor, Addition and Factorial in SKI-Tile

References

1. Doty, D.: Theory of Algorithmic Self-Assembly. Comm. ACM **55**(12) (2012)
2. Winfree, E.: On the Ccomputational Power of DNA Annealing and Ligation. In: 2^{th} DIMACS Meeting on DNA Based Computers. (June 1996)
3. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and Self-Assembly of Two-Dimensional DNA Crystals. Nature **394** (1998) 539–544
4. Adleman, L.: Towards a mathematical theory of self-assembly. Technical report 00-722, Department of Computer Science, University of Southern California (2000)
5. Rothemund, P., Winfree, E.: The Program Size Complexity of Self-Assembled Squares - extended abstract. In: ACM Symposium on Theory of Computing. (2000) 459468
6. Rothemund, P.: Using lateral capillary forces to compute by self-assembly. PNAS **97**(3) (2000) 984–989
7. LaBean, T., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J., Seeman, N.: The construction, analysis, ligation and self-assembly of dna triple crossover complexes. J. Am. Chem. Soc. **122** (2000) 1848–1860
8. Mao, C., LaBean, T., Reif, J., Seeman, N.: Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature **407** (2000) 493–496
9. Jonoska, N., Liao, S., Seeman, N.: Transducers with programmable input by dna self-assembly. In: Molecular Computing. LNCS 2950 (2004) 219240
10. Doty, D., Patitz, M.: A domain-specific language for programming in the tile assembly model. In: Proceedings of DNA. (2009) 2534
11. Winfree, E.: Simulations of Computing by Self-Assembly. In: 4^{th} DIMACS Meeting on DNA Based Computer. (June 1998)
12. Robinson, R.M.: The Undecidability and Nonperiodicity for Tilings of the Plane. Inventiones math. **12** (1972) 177–209
13. Schonfinkel, M.: On the Building Blocks of Mathematical Logic. in From Frege to Gdel - A Source Book in Mathematical Logic 1879-1931Harvard University Press, 1967 (1924)
14. H.B.Curry, R.Feys: Combinatory Logic. North-Holland Publishing Company, Amsterdam (1956)
15. Turner, D.: Another algorithm for bracket abstraction. The Journal of Symbolic logic **44**(2) (1979)
16. Hughes, J.: Graph reductions with super-combinators. Technical monograph prg-28, Oxford University Computing Laboratory, Programming Research Group (1982)
17. Barendregt, H.P.: Functional Programming and Lambda Calculus. in Handbook of Theoretical Computer Science: Formal Models and Semantics, Elsevier-The MIT Press (1990)
18. Huet, G.: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. J. ACM **27**(4) (1980) 797–821
19. Jones, S.L.P.: The Implementation of Functional Programming Languages. International Series in Computer Science, Prentice-Hall (1987)

Engineering MAS – A Device Integration Framework for Smart Home Environmentss

Jack Betts and Berndt Müller

University of South Wales, United Kingdom
{jack.betts,bertie.muller}@southwales.ac.uk

Abstract. We introduce a layered approach to multi-agent programming and motivate this with a perspective to smart home environments. Apart from the core layer, layer components can be updated at runtime to reflect, e.g., attributes like credibility and the addition of proprietary functionality. The Layered Agent Framework (LAF) is defined by interfaces and organised into layers. This approach minimises system fragmentation while allowing developers to create and maintain meaningful and effective agents. A Petri net model is provided to visualise and execute prototypes of the agents. Although fully functional, the Petri nets will later be translated into dedicated programs with a smaller footprint and more efficient execution.

1 Introduction

The paper discusses a flexible, dynamic, and software-developer friendly framework for the development of multi-agent systems with the application domain of home automation and assistance in mind. The resulting framework will be easily adaptable for other domains, since the smart home encompasses a multitude of technologies and has many requirements - such as real-time interaction - also found in other areas.

The remainder of this section will give an introduction to the application domain and the problems we are addressing with our approach. Section 3 presents our approach to a layered agent-based framework for the integration of devices in the home-automation domain. Some notes on the implementation can be found in Section 3.3. We close with an outlook on the next steps and possible further developments in Section 4.

1.1 What is a Smart Home?

When reading about smart devices or smart solutions, usually very little is revealed about the actual methodology used to make them ‘smart’. In some cases this may be due to the fact that the developer wants to conceal the fact that they are using agent technology, in other cases the marketing of a ‘smart’ service may distract from reality and from the fact that the smartness of the service is just an added timer or a simple remote control on a mobile phone. When we think of a smart home, we use the term ‘smart’ to denote the use of intelligent solutions that would help organise our everyday life in and around the home. So, what would be involved? Rather than a timer giving the illusion of an intelligent system agent technology should be utilised to enable the ‘smart

home' to learn from the user and adapt to better help the user. This learning should be unobtrusive from the perspective of the user. User interaction will be encouraged but not enforced in order for the system to learn quicker.[1] The following is a (probably not exhaustive) list of areas affected by smart-home technology:

- energy systems and power consumption
- entertainment systems (audio, video, and text)
- health support systems
- household appliances
- shopping and banking transactions
- education and reference sourcing
- news provision

From this list it becomes clear that a wide range of areas have to be covered and many different requirements have to be addressed. Our aim is to provide an infrastructure to enable reasoning and communication amongst all entities involved in these areas. This includes stationary and mobile entities situated within the home and mobile entities that may enter and leave the home environment. A change of environments may require different representation of data or data dependencies [1] and this has to be supported by the framework.

1.2 The role of AI in a Smart Home

Focussing on smart home applications, we are faced with a variety of AI techniques that can be employed to assist the user in an unobtrusive, reliable, and helpful way. Our main focus is on agent orientation and the use of multiple (mobile) agents. To achieve the level of integration required to address all (or at least most) of the challenges, we have to allow for different types of communication and different types of actions. Communication and stored data have to be encrypted where necessary and to an appropriate standard. Access to parts of the system has to be authenticated and regimented.

1.3 Core Areas

The core areas supported by our system architecture are

- cooperation by communication
- fault tolerance
- user experience
- intelligent unobtrusiveness.

Communication is a foundation for all of these core areas. Unobtrusive assistance can only be achieved if the information to be provided is available everywhere at any time[2], so that the optimal (or near optimal) moment and place can be chosen to convey this information to the user (or agent). Fault tolerance also requires communication in order to re-schedule tasks while a device is unavailable. Methods to support the core functionality are part of a core layer in our design.

[1] In this paper we are not concerned with the actual learning or the provision of the intelligence, but rather with an architecture to support the development of such systems.

[2] This is an idealised vision. In practice it will be sufficient to have the possibility of the information being made available at many locations in the home most of the time.

1.4 What is new in our approach?

The methodology introduced in this paper is designed to help the software developer by supplying familiar concepts from the world of object orientation to that of agent orientation without making everything behave like an agent. We introduce several layers with notions of inheritance and overriding as known from established object oriented frameworks. Components and patterns are encouraged for the supply of common agent functionality. For example, agent or task generation can be accomplished by instantiation from a blueprint, much alike the creation of an object from a class.

The way the architecture is designed has dynamic systems and an excellent user experience in mind. Adding or changing functionality as necessary within the running system is provided on layers on top of the agent core. These layers can be modified dynamically without the need of restarting the system, hence integrating seamlessly in the user's workflow. The architecture can support agents can keep previous copies of protocols, roles, and layer-specific functionality in case an update fails. Hereby, an immediate rollback to a previous version of a layer is possible and a report the update as failed can be communicated without making the system inoperative. For the smart home environment, this means unobtrusive updates and structural changes are possible without noticeable interruptions of the overall system's functionality.

2 Summarising the *status quo*

We briefly summarise some existing work on smart home automation. This is not meant to be a full account of the literature, but serves as an indication of different approaches. Section 2.1 discusses AI techniques used in various flavours of home automation. Section 2.2 gives an overview of agent-based approaches. In both sections, we point out which aspects have influenced the design decisions for the model presented in Section 3.

2.1 General Approaches

In the past decade, home automation has been discussed at various levels. Some approaches involve AI techniques [2, 3], others are looking at home integration from a predominantly (electrical-)engineering perspective (e.g. [4]) or from a sociological viewpoint (e.g. [5]).

Many approaches tackle only one aspect of home automation, e.g. the heating system or power consumption.

2.2 Agent-based Approaches

A multitude of papers discuss energy management or other specific home automation systems. One of these, [6], makes the point that the home automation market is fragmented. Different technologies compete, are incompatible and co-exist, rather than cooperate. It is argued that interoperation can be achieved by means of an abstraction layer that would allow access to different home automation devices in a uniform and generic

way, independent of the underlying technology. [7] is concerned with resource allocation and optimisation using an abstract negotiation protocol. Agents negotiate near optimal settings for minimising power consumption to reduce the greenhouse effect by integrating appliances and heating systems. The paper remains at an academic level in that no solutions are offered for an implementation in a real-world situation.

Other approaches are more general and in line with the present endeavour, but remain unspecific. [8] describes a home automation system called HORUS, which is based on an agent architecture including managers, IO handles, video-camera handlers, and alarm communicators. The IP-communication-based system remains rather vague about the format of rules and their interpretation in a setting with existing appliances.

3 Our Approach

The centrepiece of our approach is a component-based and software-developer friendly system architecture that will allow legacy systems to be integrated into a multi-agent-based framework built around the requirements of smart home automation. As such it will allow the developer to dynamiacally incorporate many aspects including *learning of behavioural patterns*, *handling of sensitive data*, *unobtrusive conveyance of information and general assistance*, and *monitoring of various sensors*.

The software development approach is based on object orientated design, but does not simply replace objects with agents as some approaches in the past have[3]. Instead, we use agents alongside traditional objects to reflect different capabilities of the various system components. In doing so, we avoid having to discard legacy components and re-design them for an all-agents system. Also, we avoid unnecessary communications complexity that an all agents approach would incur.

The main reason for focusing our approach on the smart home environment is that future real-world use of agent technology will only be of benefit when technologies and requirements meet and a (more or less) seamless interaction is guaranteed. This is of foremost importance and means that isolated studies are generally not scalable. By providing the interfaces for the use of modern agent-based interaction we open up possibilities without restricting the use of more traditional software and hardware paradigms. For the home environment, this means that new devices and appliances might be constructed to include some additional inexpensive hardware/software components to allow integration with others, while legacy components may be integrated by simple plug-ons (software and hardware), such as a communications-enabled wall-plug adapter that has some basic control over an appliance (e.g., switching it on or off or monitoring power draw and detecting usage patterns of the socket and making this information accessible this information to the agent network.).

The components introduced in this paper will be part of a home system design consisting of stationary and mobile devices and learning control systems based on software agents. The stationary devices are in the main part traditional household appliances whose controls become part of a dynamically learning distributed control system including some stationary devices (perceptors, like motion and temperature sensors; control hubs, acting somewhat like servers on which computationally expensive tasks of

[3] "Agents can be seen as the successors of objects and classes ...", `http://aose.org`

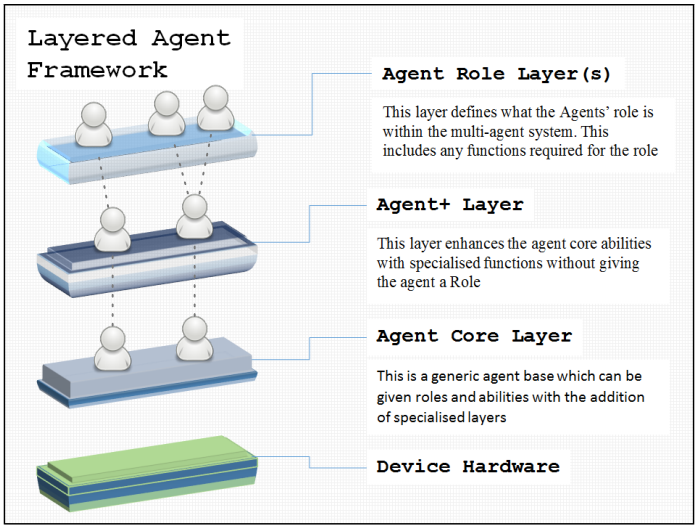


Fig. 1. Layered Agent Framework

learning and processing of data is done) and mobile devices migrating between environments and allowing the overall system to learn behavioural patterns as well as hosting agents themselves to carry out independent tasks.

3.1 Theoretical Model

When creating multi-agent systems a common platform on which the agents are built is necessary. This may be a common communication protocol to enable all the agents to talk to each other. This can be a common runtime like a Java Virtual Machine (JVM), or an operating system (Linux) providing a common platform for all agents. Previous research has brought about a common intermediate layer (MCAPL) that executes agents programmed in a variety of agent programming languages after translation [9, 10].

Common communication elements and interfaces are required to enable agents to work together, however different multi-agent systems developed by different development teams generally cannot communicate effectively with each other without specific adapters. A simple solution would be to use a common open communication protocol and declare this a standard for all agent communications[4]. This is a solution only if all agents on all systems followed this standard, and that the protocol offered everything that all MAS developers need.

This leads to the question: How is it possible to allow agent developers as much freedom as possible without imposing too many restrictions on their code? Our approach is the design of a Layered Agent Framework (LAF, see Figure 1) that provides developers with a generic agent layer that can be specialised in order to fulfil defined roles. This specialisation is achieved through the addition of further layers. The system

[4] http://www.fipa.org

is not limited to one role per agent; the *Agent Role Layers* can be stacked to provide a multi-role agent system. The ordering of the role layers determines the priorities of the agent. Any role layers that are packaged with the system are default roles. Default roles can not be dropped by the system and cannot be superseded in priority by non default role layers (default role priority set by the developer or the system). This layered approach loosely couples additional roles to an agent, allowing roles to be changed, updated or multiplied without requiring the agent to be fully restarted. Not only are the agent roles loosely coupled but so is the entire agent device. This agent system is intended to produce a basic prototype agent device which manufacturers can integrate into an appliance, apply their role layer and then ship it as a single unit. If the manufacturer improves a layer, they can issue a new updated layer to devices since the agent roles are entirely software based. We describe the layers of Figure 1 in some detail in the following paragraphs.

Device Hardware The *Device Hardware* is at a basic level the processor, memory, storage and communications and a Linux operating system to manage it all. Linux grants the *Agent Core Layer* access the hardware (Linux handles hardware drivers). Any specialist hardware API(Application Programming Interface) for example a GPU or heating element. APIs are defined in either the *Agent+ Layer* if the special hardware is attached to the *Agent Device Unit* or in the *Agent Role Layer* if the special hardware is part of the appliance or utility which the agent device is attached to. For example if a coffee machine has an *Agent Device* attached to it, the *Agent Role Layer* would contain the APIs required for the *Agent Core* to use the coffee machine functionality. The *Agent Device* can be embedded within an appliance or attached to one. The hardware layer will provide a flag indicating to the agent whether it is embedded or attached.

Agent Core Layer As the name implies this layer is to be considered the core of the system. This layer will be generic across all implementations to combat system fragmentation[5] and maintain a high level of agent interoperability. This layer is an adaptable agent, adaptive in that is must work to complete tasks with any roles assigned to it. An agent engine will form the main part of this layer along with an array of layer APIs as well as a dynamic action cache used to formulate plans based on role and available abilities. The *Agent Core Layer* is not to be designed for dynamic updates to any functionality. Instead any functionality that requires an update is to be overloaded in the Agent+ layer. If an update within this layer is absolutely necessary (like security) then the agent device will require a restart unlike updating in any other layer. The Common Agent Engine provided by the *Agent Core Layer* creates by default at least one agent upon system start. This agent is considered the *Master Agent* of the system and takes on any default roles assigned to it. *Slave Agents* can be created to handle any addition

[5] System fragmentation refers to parts of a (common) system being incompatible with other parts. E.g., Apple iOS devices have low levels of fragmentation, since an app written for an iPhone 3 will work on an iPhone 5 and (mostly) vice versa. Compare this with the Android platform where there are huge differences, such that apps will generally come with extensive lists of supported devices/configurations. For our framework, this means that agents should be able to work together regardless of the developer, manufacturer, or purpose.

roles assigned to the agent system. The *Master Agent* has full control of the system hardware and *Slave Agents*. In doing so the *Master Agent* will have more responsibilities attributed to it compared to the *Slave Agents* who will only have to satisfy their role requirements.

Each agent has access to the *Agent+ Layer* and assigned roles from the *Agent Role Layer*. *Slave Agents* will have access to the hardware but the *Master Agent* will have assigned a 'Hardware Need Value' to each agent dependant on their role. This value represents a proportion of system resource usage and on the system adds up to 1. Consider a TV with access to a GPU and TV Tuner hardware and two agents currently running on it. One *Slave Agent* is tasked with recording a user's favourite shows and this has a GPU need of 0.2 and a Tuner need of 0.5. The other agent is the *Master Agent* for the TV system and is responsible for displaying content, programming recordings, and responding to user interaction with a GPU need of 0.8 and a Tuner need of 0.5 (assuming the user is watching live TV). With the TV off, the *Master Agent* allocates almost all GPU and Tuner usage to the *Slave Agent* based of its needs. This is a simplistic view of how resources could be managed and is intended to demonstrate one of the extra roles (resource management) the *Master Agent* will be assigned and how this may work.

Agent+ Layer This is an enhancement layer to the *Agent Core* designed to provide dynamic updates and extend core functionality when required. When an update is applied to the *Agent+ Layer* the *Agent Device* does not need to power down. The updating layer simply becomes locked while the update occurs and then – once successful – returns to an operational state. For instance, assume an *Agent Device* has a GPU(Graphics Processing Unit) and this *Agent Device* is attached to a fridge. The fridge will not have any need for a GPU so the *Agent Role Layer* will contain no API's for a GPU. Therefore there must be a way in which *Agent Device* developers can add any functionality directly to the *Agent Device* without having to create pseudo role for it. The *Agent+ Layer* allows for this kind of extension of core functionality.

Agent Role Layer Role specific functionality and APIs are stored here. The *Agent Role Layer* and *Agent+ Layer* are where agent developers will spend their time as these layers define what the agent is and how it should behave. If the agent role requires a certain type of hardware or proprietary algorithm to work as intended this would be implemented in the *Agent Role Layer*. An agent can have any number n of roles ($n \in \mathbb{N}$ can be 0) provided the hardware can support that number of roles. This can allow for agent-network-wide load balancing as roles can be duplicated to a numerous compatible *Agent Devices*. An Agents' compatibility for a role is dependent on hardware requirements for the role. An agent can accept a role if specialist hardware is not available to it, in this sense the agent takes on a support role. For example a security system might be trying to identify who is in the house. The Home PC is not in use and neither is the coffee machine so the security agent asks them to take on a support role and help process some of the data. The two devices are both in possession of a CPU capable of processing the data required by the security agent and therefore are compatible for the support role. Once the support role is no longer required, agents can make the decision to drop any extra roles[6]).

[6]Security agents can force agents to drop certain roles like security roles.

Types of Agents using the Layered Agent Framework Agents using the Layered Agent Framework come in three forms:

Standalone device (Agent Device) The agent is not attached to any appliance or utility. This type of agent acts as an agent controller for a specific area of the agent network, or a worker agent which can be used by other agents on the network for data processing much like a server.

Attached to an appliance/utility (Attached Agent Device) The agent is attached to an appliance and is enabled to use the functionality of the appliance. The actual agent device is a separate unit to the appliance. This setup is designed to allow current appliances and utilities to be adapted for agent technology by interfacing the agent unit with the appliance or utility. This also allows the agent unit to be removed for repairs or to disable the agent capabilities and control over an appliance.

Embedded in the appliance/utility (Embedded Agent Device) These agent devices will be part of the appliance/utility and cannot be removed. If a hardware fault occurs the system will default back to a 'dumb' unit until repairs are carried out.

3.2 Executable Model

We introduce a Petri net model based on the nets-within-nets paradigm. It builds on the MULAN multi-agent architecture and is implemented in the RENEW tool. We give a brief overview of MULAN in Section 3.2 and then introduce our model in Section 3.2.

Multi-Agent Petri Nets We focus on the MULAN architecture shown in Figure 2 as introduced in [11]. MULAN separates the multi-agent system into four parts or layers. The layers are: (a) agent network, (b) agent platform, (c) agent, and (d) protocol.

Each layer is represented by its own Petri net(s). Protocols specify the agent programs. Agents reside on a platform that provides internal and external communication facilities and is located in an environment within the multi-agent network of the multi-agent system. The latter determines the communication structure available to agents.

Because it is unrealistic to assume legacy products to (fully) support agent communication, we generalise parts of the MULAN architecture to reflect this scenario. In particular, the *agent platform* will become simply a *platform* and the *multi-agent network* will be referred to as *network*.

Multi-Agent Petri Net Components The Multi-Agent Petri Net Components (MAPNCs) run on a MULAN-based architecture. They constitute the building blocks of agents and protocols, e.g. for agents created at runtime by other agents. MAPNCs are presently limited in that their 'template' or 'blueprint' has to be defined prior to run time and have to have certain properties discussed below.

The possible templates are stored in a place of the agent net similar to the protocols. Whenever an instance is required, a copy of one of these templates is initialised according to its task and the initialised copy is then moved onto the platform from on which it operates. As a mobile agent, the generated instance (of an agent) can then traverse the network to reach other platforms and interact with (agent) nets at remote locations.

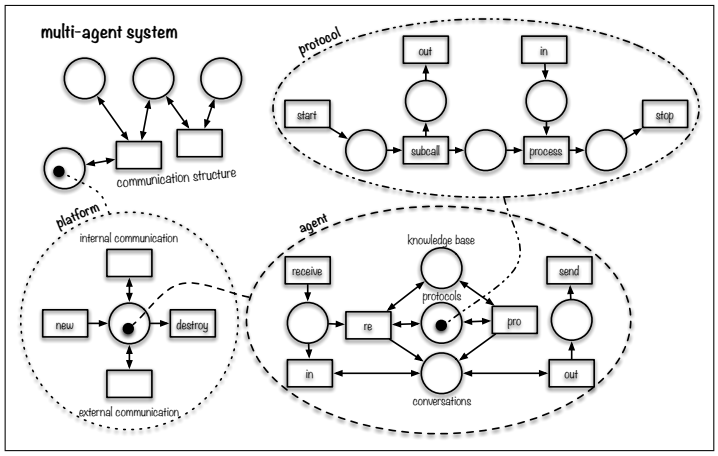


Fig. 2. MULAN agent architecture [11]

Figure 3 shows the structure of the extended MULAN architecture that includes the layer structure introduced in Section 3.1.

The extended MULAN provides the multi-layer support for agent communication by providing the layer functionality in an additional place regulating the communication by forcing synchronisation according to the layer information and allowing only communication based on the specifications therein. Communication can be 'negotiated' to take place on different levels according to the different layers.

If a method is overridden, this will result in its protocol becoming unavailable for execution in the Petri net model.[7]

3.3 Towards an Implementation

This section discusses some issues related to the implementation of our framework.

Brief Roadmap Designing any system requires an awareness of implementation. With this in mind any design decisions must made with consideration towards aspects like available system resources and response times. Our aim is to provide an architecture for flexible agent-based, object-oriented systems (Layered Agent Framework, LAF). The Agent Core represents the main component that every system using the LAF will be based on. Elementary communication features and functionality will be implemented on the first prototype system. At least one Agent+ layer will have to be created for testing with the Agent Core. This basic system set up will then be used to test the dynamic update capability of the framework as well as simple extensions to the Agent Core

[7] This is easily achieved by removal of a required resource, i.e. an input token to the respective transition and appropriate transition synchronisation. This is not shown explicitly in the extended model in Figure3, because it is implemented at a protocol net level and the relevant inscription detail had to be omitted for brevity and readability.

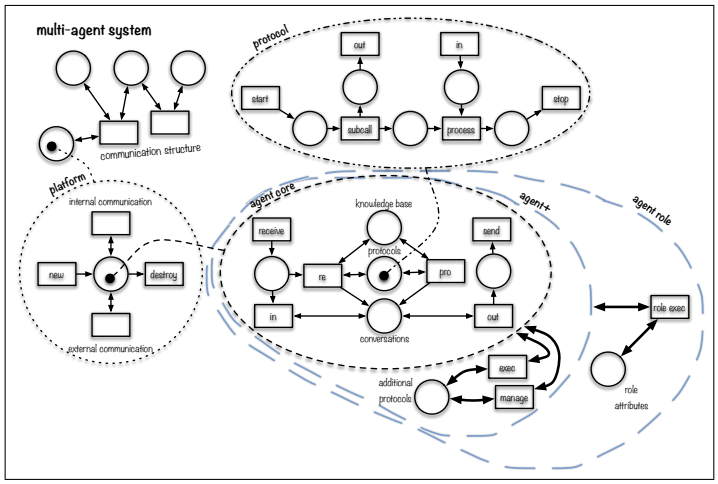


Fig. 3. Extended MULAN agent architecture

functionality. Once the Agent+ Layer has been implemented and tested, Agent Role Layers can be added. An arbitrary number of Agent Role Layers can be developed and deployed to make full use of the flexible design of the Layered Agent Framework. The details of what extensions, roles or testing will be done has not yet been determined. The framework will be supported by a set of development tools that will assist the developer with the development of bespoke solutions, i.e., Agent+ layers and agent roles. The tools will include templates and components to assist the construction of agent plans, for example they will supply action templates and agent creation blueprints.

Manual Override Every system build using LAF must have a manual override in case of user preference or fault, *"open the pod bay doors HAL"* [8]. This manual override will place the agent into disabled state in which it has no control over its appliance or utility. Such agents can still communicate their state on the network and attempt to assist other agents(assuming no fault). The manual override simply (in software terms) blocks the agents' control over its appliance/utility which from then on will need to be user operated. The agent can be 'enabled' by the user flipping a switch or by asking the agent network to enable the agent again(expressed consent must be given).

The Challenge of Real-Time Responsiveness There are many challenges facing the final implemented framework and any system based on it. Most importantly, the system must work in real time. Having a system working in a home environment in which the user must wait more than a minute for a response is generally unacceptable. A user can expect delays in response for processing actions such as *"What times are the buses to Cardiff"*, *"one moment.... 1315 and each hour from then till 2100"* this kind of scenario

[8] Stanley Kubricks' *2001 A Space Odyssey*. HAL is asked to open the pod bay doors, HAL refuses this request with the response *"I'm sorry Dave, I'm afraid i can't do that"*.

is acceptable. However if the user asked for the curtain to be drawn and had a 2 minute wait before the curtain even started to move is very undesirable. This also poses the question of user feedback when there are unavoidable delays, but this is outside the scope of this paper, since it would be the programmer's duty to supply the feedback.[9] Aside from real-time responsiveness the system must have a high fault tolerance and some form of system crash recovery. A computer crashing is one thing but an entire home of systems crashing could be disastrous and dangerous. Hence, we need systems with the ability to recover quickly from a major fault or system-wide disruption. Any solutions must attempt not to affect the user in a noticeable way. The perfect recovery solution would be the system completely recovering from a system wide fault without the user knowing there was ever anything wrong.

The Challenge of Agent Security Security is a major concern in the home, especially considering the amount of personal data that will be stored on the users' lives. There are many ways in which data can be protected, ranging from passwords to strong data encryption. As security measures placed upon data are increased the less dynamic and freely available that data becomes and a balance needs to be negotiated such that some data become accessible without compromising the required level of data protection. One mechanism could be a relational notion of trust that agents can have with other agents. The idea is that agents who have proven they are not a threat to the system over time will become trusted by other agents. An untrusted agents' request may be rejected, in contrast trusted agents are more likely to have requests completed by other agents. Certain agents can be pre-set by developers to have a trust limit for example security systems should only have complete trust in other home security components with a certified ID. A private trust value held by every agent on every other agent allows new devices to use the network without compromising on the security of the network. Trust levels would range from 0 to 1, representing the range from *untrusted* to *fully trusted*. A new device will usually start on a neutral trust level of 0.5 (unless otherwise pre-set). Unless the new device was a permanent addition to the smart network (e.g., a cooker) for which exceptions can be made, if permitted by the security system. Trust values will be updated dynamically at run time and can be influenced by the security system.

4 Conclusion

We have introduced an agent architecture that supports object-oriented concepts most software developers are familiar with and that adds component-based agent layers. These layers provide basic and extended agent reasoning and communications facilities in a maximally flexible way. The architecture is primarily targeted at smart home automation. For this, it supports dynamic reconfiguration and seamless integration with non-agent-based systems. This is required because the user experience is of foremost importance in the smart home application area. Easily configurable components can be added at runtime to provide additional features and to configure security features as required by individual sub-systems.

[9] The tools to be developed for the framework will have templates for a variety of devices and appliances that can be equipped with basic user feedback functions.

A Petri net implementation has been presented in this paper. The next steps will be to implement the framework on inexpensive hardware that can be integrated into new appliances or added to existing devices, e.g. in the form of a simple wall-plug adapter.

Also following successful testing of the framework, will be the provision of a tool kit with components and design patterns to ease the construction of agents, layers, and roles. Ideally, we would like to link the visual creation of the latter (by means of agent-based object Petri nets) to the development environment, providing an automated translation of the Petri-net representation into an agent program. The Petri net model could then be analysed using existing methods and tools, while the properties are preserved by a verified translation procedure.

References

1. Köhler, M., Farwer, B.: Modelling global and local name spaces for mobile agents using object nets. Fundamenta Informaticae **72**(1-3) (2006) 109–122
2. Ceccaroni, L., Verdaguer, X.: Agent-oriented, multimedia, interactive services in home automation
3. Fraile, J., Bajo, J., Lancho, B., Sanz, E.: Hoca home care multi-agent architecture. In Corchado, J., Rodríguez, S., Llinas, J., Molina, J., eds.: International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008). Volume 50 of Advances in Soft Computing. Springer Berlin Heidelberg (2009) 52–61
4. Sriskanthan, N., Tan, F., Karande, A.: Bluetooth based home automation system. Microprocessors and Microsystems **26**(6) (2002) 281–289
5. Blackwell, A.F., Rode, J.A., Toye, E.F.: How do we program the home? gender, attention investment, and the psychology of programming at home. International Journal of Human-Computer Studies **67**(4) (2009) 324 – 341
6. Nunes, R.J.C.: Home automation - a step towards better energy management. IST – Technical University of Lisbon / INESC-ID R. Alves Redol, 9, 1000-029 Lisboa, Portugal (2003)
7. Abras, S., Ploix, S., Pesty, S., Jacomino, M.: A multi-agent home automation system for power management. In Andrade-Cetto, J., Ferrier, J.L., Pereira, J.D., Filipe, J., eds.: ICINCO-ICSO, INSTICC Press (2006) 3–8
8. Giordana, A., Mendola, D., Monfrecola, D., Moio, A.: Horus: an agent system for home automation. In: 13th Workshop on Objects and Agents (WOA 2012), CEUR Workshop Proceedings Vol-892 (ISSN 1613-0073) (2012)
9. Dennis, L., Farwer, B., Bordini, R., Fisher, M., Wooldridge, M.: A common semantic basis for BDI languages. In Dastani, M., Seghrouchni, A.E.F., Ricci, A., Winikoff, M., eds.: Proceedings of the International Workshop on Programming Multi-Agent Systems (ProMAS 2007). (May 2007) 88–103
10. R. H. Bordini, L. A. Dennis, B.F., Fisher, M.: Directions for agent model checking. In M. Dastani, K. V. Hindriks, J.J.C.M., ed.: Specification and Verification of Multi-agent Systems. Springer US (2010) 103–123
11. Köhler, M., Moldt, D., Rölke, H.: Modelling the structure and behaviour of petri net agents. In Colom, J.M., Koutny, M., eds.: Applications and Theory of Petri Nets 2001. Volume 2075 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 224–241

Experiments with Simulated Humanoid Robots

Hans-Dieter Burkhard and Monika Domańska

Humboldt University Berlin, Institute of Informatics, Germany
http://www.naoteamhumboldt.de

Abstract. Experimenting with real robots is limited by the available ressources: Complex hardware is costly, and it needs time and experience for setup and maintenance. Simulated robots can be used as alternative. Our RoboNewbie project is a basic framework for experimenting with simulated robots. It serves as an inspiration for beginners, and it provides room for many challenging experiments. The RoboNewbie agents run in the simulation environment of SimSpark RCSS, the official RoboCup 3D simulator, where the simulated robots are models of the humanoid Robot NAO of the French Company Aldebaran. Different example agents provide easily understandable interfaces to simulated sensors and effectors of the robot as well as simple control structures. The framework has been successfully used at different courses where the participants needed only few hours to understand the usage of the framework and to develop own agents for different tasks.

Keywords: Robotics Tutorials, RoboCup, e-Learning

1 Introduction

Understanding grows with active commitment: to "do" something, to master it, provides a deeper understanding. Experiencing with own experiments is of course an important prerequisite for studies in Robotics and Artificial Intelligence as well. But experimenting with real robots is difficult not only because of expensive hardware. Maintaining the robots and set ups for experiments are very time consuming even for experienced people. Experiments at home as needed for e-learning require a deep technical understanding by the students, i.e. experiences that they are just going to learn. So it is not surprising that simple hardware is still broadly used in robot experiments, hardware which is far behind the recent technical developments, not to talk about e.g. complex humanoid robots. The collection of papers in [1] can be understood as an illustration of our statements.

Simulated robots in simulated environments can be used as an alternative for complex hardware. The RoboCup community has more than 15 years of experiences with real and simulated robots in the field of soccer playing robots [2]. Soccer playing robots have been established as a challenging test field for the progress in scientific research and technical developments. Robots have to be able to control their bodies and their motions according to soccer play, they must perceive a dynamically changing environment and they have to choose

successful actions out of many options in real time. They have to cooperate with team mates and to pay attention to opponents. Several thousand scientists and students are participating in the annual RoboCup competitions in different leagues with different types of real and simulated robots. The humanoid robot Nao of the French Company Aldebaran [5] is used in the Standard Platform League, while its simulated version is used in the 3D-Simulation League. The official SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS) [3] provides an excellent environment for experiments with simulated complex robots (see Section 3). It provides a physical simulation using ODE [7] for the body dynamics of the robot Nao and the soccer environment.

Our RoboNewbie Project is a basic framework based on JAVA for the development of simulated humanoid robots. It provides easy understandable interfaces to simulated sensors and effectors of the robot as well as a simple control structure. It runs in the environment of the SimSpark RCSS, thus it can but need not be used for soccer playing robots. Users can develop their own motions, e.g. for dancing, gymnastics or kicking a ball.

The RoboNewbie Project implements some kind of "minimalistic approach" with respect to Robotics. Users are able to start without special knowledge about robots. They can learn by their own experiences about the basic concepts of perception, motion, control, synchronization, and integration. All related program code in RoboNewbie is understandable from simple principles without further knowledge. That concerns the structure of the code as well as the underlying computational methods. As soon as users learn more about Robotics, they will be able to extend the programs accordingly, e.g. concerning complex motions or world modelling.

Following some hints of the reviewers, we would also like to emphasize the potential of the framework for the research on foundations. e.g., on computational models as well as on different problems in cognitive science. It can be useful in verifying models and in gathering large data sets for experiments in data mining.

The paper is organized as follows: After an overview about the concept and the downloadable resources of the RoboNewbie project, we give a short overview about SimSpark RCSS, and we describe the communication between the RoboNewbie agents and SimSpark RCSS. The main part of the paper in Section 5 discusses the details of the RoboNewbie framework, and the paper ends with results of practical evaluations and our conclusions.

2 The RoboNewbie Project and its Resources

The main goal of the RoboNewbie Project is to provide an uncomplicated starting point to the programming of complex robots with minimal requirements and pre-knowledge. The users are only supposed to have some programming background (Java) and some technical/mathematical understanding. More knowledge about robotics can be provided in parallel to the exercises with RoboNewbie, e.g. in introductory tutorials (as we already did) or by e-Learning material. At

its present stage, RoboNewbie is not prepared as a complete course material on university level like e.g. the course "Autonomous Multiagent Systems" at the University of Texas [8]. But it is planned to integrate it in a e-learning course on Robotics.

The objectives behind RoboNewbie is the realization of the following requirements:

- Holistic view on robots: For beginners and especially for pupils in schools, it is more appealing to see a robot behave like a human than to test and calibrate the behavior of a sensor. Of course, when dealing with more complex tasks, users will experience the need to have better knowledge about the usage of sensors and actuators, and then they may draw their own conclusions.
- Motivating scenario: Application fields from daily life with known properties and rules are well suited. Robots which imitate human skills are especially motivating.
- Scalable tasks: Unexperienced users should have no difficulties to perform first steps with own experiments and later move to more complex tasks with unlimited challenges.
- Low requirements: The usability would be restricted if peoples need to have education on Robotics or if they are supposed to have deep knowledge in hardware and software. To be usable at schools, basic programming skills and interests in mathematics and natural sciences should be sufficient.
- Low costs: The costs of a learning system include money and efforts for purchase, set up, and maintenance, respectively. They should be as low as possible to permit a broad usage.

The users of the RoboNewbie project can find all materials on the web page of Berlin United – Nao Team Humboldt [6]. Besides links to RoboCup, Nao (Aldebaran) and the SimSpark-Wiki, it contains resources for download:

- Description of Installation and first steps.
- Sources of the RoboNewbie Agent programmed in JAVA 7 and prepared for usage under Netbeans.
- Quick start tutorial: Introduction to the features and the usage of the agent.
- Motion Editor for the design of Keyframe Motions (needs JAVA 3D to be installed).
- SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS) for Windows with an introduction to SimSpark RCSS as far as needed for RoboNewbie.

All provided code is open source. Some parts of the RoboNewbie code use code of the RoboCup team magmaOffenburg [4].

3 SimSpark RoboCup 3D Soccer Simulation

SimSpark RCSS is developed and used by the RoboCup community in the 3D simulation league. SimSpark is a generic physical multi agent simulator system for agents in three-dimensional environments. It uses the Open Dynamics Engine

(ODE [7]) for detecting collisions and for simulating rigid body dynamics. ODE allows accurate simulation of the physical properties of objects such as velocity, inertia and friction.

The Simulator SimSpark RCSS consists of two programs (server for simulation and monitor for visualization and interaction) and configuration files. It models a soccer field with the player bodies (adapted from the robot hardware of Nao) and the ball. It also controls the rules of the soccer game, i.e. it controls the game according to the decisions of a referee.

SimSpark RCSS can be used as open source software. This was also an important criteria for its usage. It can be downloaded from [3] for different platforms. A complete preconfigured version for Windows 7 is provided for RoboNewbie which can be downloaded from the RoboNewbie web page [6]. Nevertheless, the RoboNewbie agents run with SimSpark RCSS under other platforms, too. By some small changes in the configuration files, the soccer rules are simplified for first usages with RoboNewbie.

The SimSpark RCSS project itself is constantly evolving according to the progress in the RoboCup initiative. The version (compiled in June 2012) on the RoboNewbie web pages serves for stable usage and avoids potential incompatibility problems by new RoboCup versions.

SimSpark RCSS is documented in a Wiki [3] with download links to the latest versions as used in the competitions. The Wiki documentation is thought to represent the actual state of the simulator by continuous updates. But since different developers are volunteering in parallel on different tasks in the project, the structure of the Wiki is not always optimal, and occasionally some outdated information is still present. Moreover, the Wiki is directed to experienced users which makes it sometimes difficult to understand for novices.

To provide an easy access, the down loads of the RoboNewbie project contain an introduction to SimSpark RCSS which refers to the provided version (as described above). It gives the user an overview about

- Simulation using SimSpark RCSS: The SoccerServer and the Monitor.
- The Nao-Model used by SimSpark RCSS.
- Communication between Agents and SimSpark RCSS (with explanations of the message formats as background information).
- Synchronization between SimSpark RCSS and the Agents.
- Monitor and User Interface.
- Running a Game.

Actually, our description of SimSpark RCSS provides also some "background" information which is not needed for beginners, e.g. details about the message formats. Since RoboNewbie permits an easy and direct access to the items of messages like sensor values and motor commands, the syntax of messages must not be known by users. Nevertheless, we have included the information for deeper understanding of RoboNewbie in case of interest.

4 Communication between Agents and SimSpark RCSS

SimSpark RCSS implements the soccer environment including the bodies of the Nao robots. It models all physical interactions between players, ball and environment. The agents implement the control of the players.

The interface between the physical environment and the control of real robots is constituted by sensors and actuators: Robots perceive the world by sensory data (e.g. by vision, accelerometer, force sensors etc.), and influence the world by their actuators (motors, voice etc.).

In simulation, the sensory data are calculated by the simulator according to the situation in the simulated world (e.g. observable objects) and sent via message exchange to the agent. Then, like a real robot, the agent can update its belief about the situation and decide for actions it wants to perform. A real robot would then activate its actuators (e.g. motors at the joints) to perform the intended actions. In simulation, the agent communicates with SimSpark RCSS again by messages which transmit the actuator commands. Both are synchronized by a communication cycle of 20 milliseconds.

In SimSpark RCSS, the message transfer is optimized for minimizing the server load: All sensory data are packed in one server message to be sent at the beginning of a communication cycle. Vice versa, the agent can send all action commands by a single agent message before the end of a cycle. The message formats follow a special syntactic scheme based on symbolic expressions (S-expressions). As a consequence of collecting data into one message, the preparation of the data in an agent needs more efforts than in a real robot. It is a special feature of the RoboNewbie agent that this preparation is hidden from the user: The agent provides special getter- and setter-methods which allow the access to the sensor (perceptor) data and the setting of actuator (effector) commands in a similar way as in a real robot.

The interaction between the server and the agent works as follows:

1. At the beginning of a cycle at a time t, the server sends individual server messages with sensations to the agents.
2. During this cycle, the agents can decide for new actions depending on their beliefs about the situation.
3. Before the end of this cycle, the agents should send their agent messages to the server for desired actions.
4. The server collects the agents messages and calculates the resulting new situation (poses and locations of the players, ball movement etc.) according to the laws of physics and the rules of the game. This is done during the following cycle at time t+1. (Note that the server message sent at the beginning of this cycle regards the situation calculated in the previous cycle at time t).
5. At the beginning of the subsequent cycle, at time t+2, the sensor data in the server message is based on the effects of the actions at time t+1 which were chosen by the agent according the information from time t.

A special feature of SimSpark RCSS is the use of so-called perceptors instead of sensors. The perceptor data can be regarded as already pre-processed sensor

data. For example, the image data from the camera are not presented by a pixel matrix. Instead, the vision perceptor sends a collection of observable objects with egocentric coordinates relatively to the camera of the observing agent. In a similar way, actions commands of the agent are encoded as so-called effector values and sent to the server which translates them to motor control commands. The calculation of perceptor values and the interpretation of effector values are part of the simulator, too. On the agent side, a server message has to be parsed for the contained perceptor values, and the action commands have to be collected to the agent message. Both constitute a significant burden for a beginner while it provides only few insights to robotics. The RoboNewbie users need not to care about that, because the RoboNewbie agent does all this work in the background.

Besides some effectors related to initial connection with SimSpark RCSS, there are Hinge Joint Effectors for each of the 22 hinge joints and a Say Perceptor (as of a loudspeaker with limited capacity). The following perceptors are available in SimSpark RCSS (for details see the Wiki or our SimSpark description):

- Vision Perceptor (as of a camera in the center of the head).
- Hinge Joint Perceptors at each of the 22 hinge joints.
- Accelerometer in the centre of the torso.
- GyroRate Perceptor in the centre of the torso.
- Force Resistance Perceptor at each foot.
- Hear Perceptor (as of a directed microphone with limited capacity).
- Game State Perceptor (reporting the actual game state of the soccer match).

5 RoboNewbie Framework

The RoboNewbie framework offers a comfortable interface for agents interacting with SimSpark RCSS. It includes sample agents which illustrate basic concepts and methods of Robotics and Artificial Intelligence. Users can start exercises with these agents and learn how to use RoboNewbie and what the programming of robots is like. They can make their own experiences with different topics and algorithm by modifications and extensions.

It was a main goal of the project, to provide easily understandable concepts, methods and programs. There are no complicated structures, and all code is documented in detail. As a consequence, some more demanding concepts were replaced by simplier approaches (e.g. keyframe motions instead of inverse kinematics, approximated coordinates of observed objects etc.). Nevertheless, the clear structure of the project supports extensions for more challenging solutions if wanted.

5.1 Low Level Interface Functionalities

The framework includes interface functionalities on two levels. The lower one corresponds to the hardware-near functionalities of robots, while the higher one

is concerned with more abstract control functionalities. Especially for the lower level, parts of the code of the team magmaOffenburg [4] was used by us as documented in our source files.

The hardware-near layer encapsulates the network protocol for interaction with SimSpark RCSS and it allows access to the simulated hardware entities corresponding to sensors and motors. The access is implemented by getter functions for perceptor values of different perceptors which can be used similar to sensor signal queries of real robots. Related setter functions for effector values can be used for the control of actuators.

Especially the low level interface functionalities for SimSpark RCSS are a hurdle for beginners and need time consuming work even for experienced users. They concern tasks like network connection, synchronisation with the server, parsing of nested server messages, syntactical analysis of S-expressions, synthesis of agent messages with a lot of technical non-robotics details. The users of RoboNewbie need not to care about all this details, the framework offers ergonomic methods for the interaction with the simulated environment in an easy understandable way similar to the methods used by the operating systems of real robots. Users can learn to use these methods after a short training time (cf. the evaluation in Section 6).

The synchronization protocol was already described in Section 4. The user needs not to care about the communication, except the delays by the protocol and the duration of the cycles given by 20 msec. It is necessary to fetch a server message at each cycle and to send the agent message before the end of the cycle. The related control structures are already implemented in the examples and explained by the tutorial. Hence, if the calculations during one cycle do not exceed the cycle time, there will be no problem. The needed time depends of course on the used computer, the example agents run without problems even on less powerful machines.

The first example "Agent_BasicStructure" in the tutorial let the users start with an agent which already implements all low level communication. The agent simply rises an arm by setting related effector values. The user can experiment with other values and other effectors just to understand the basic structures.

5.2 Perception

The available perceptors were already listet in Section 4. All perceptor values can be queried by related getter methods using the perceptor names instead of the acronyms of the server messages. This allows a comfortable access to the perceptor data which corresponds to the access of sensor values by a related operating system of a real robot.

RoboNewbie has already implemented the necessary conversion from the nested server messages to the perceptor values. For that, the server message are parsed for the constituents of a tree like structure (again, thanks to the code of the team magmaOffenburg [4]). According to the analyzed acronyms in the expressions of the tree, the corresponding perceptor values are filled in by RoboNewbie.

The programs "Agent_TestPerceptorInput" and "Agent_TestLocalFieldView" illustrate the usage of the related getter methods and the perceptor values. The examples serve also as an illustration to the usage of the logger functions described in Subsection 5.5. As an exercise of the tutorial, the user can implement an agent, which lifts the robots arm, when it senses another robot and moves the arm down, when it does not sense any robot. Which arm is lifted should depend on the side where the other robot is seen.

Special efforts are needed for the vision perceptor. It provides coordinates of all objects in the vision range of the camera. SimSpark RCCS in its common version does not communicate image data. Instead, the communicated information can be understood as the result of basic image interpretation, it contains coordinates of the goal posts, the lines, the ball, and the body parts of robots.

The vision perceptor provides values by egocentric coordinates relatively to the camera in the centre of the head. Further calculations are necessary to get the coordinates of objects relatively to the body of the robot. Accurate calculations would need the inspection of the cinematic chain. The necessary data are available by the hinge joint perceptors. More calculations including self localization are necessary for the transformation into alocentric coordinates. RoboNewbie does not provide related programs following the intended "minimalistic" approach, because they would not be understandable by beginners without preknowledge about Robotics. Instead, the implementation of related methods can serve as exercises during courses in Robotics.

As a simple substitute, we have decided to provide only approximations for the conversion from camera coordinates to robot coordinates. They are documented in the sources and easily to understand. Users can make experiments according to the accuracy and draw own conclusions on cinematic relations.

Visual information is provided by SimSpark RCSS only at each third cycle, and the robot would have to act blindly in between when there are no vision data available. Hence, the vision information should be stored for the following cycles. Moreover, the vision perceptor is limited by the camera view range of 120 degrees horizontally and vertically. The robot has to move its head to observe more objects in the world. Again it is useful to store objects seen before in other directions. In general, such updating and memorizing of observations is maintained as belief of the robot in a so called world model. Updates may regard corrections according to robot motion, guesses for movements of invisible objects and integration of information communicated by other robots.

Again, a fully elaborated world model is far behind the scope of beginners. Hence, RoboNewbie provides a very simple version, where just the observed objects are stored in a simple form. The coordinates of those objects are referenced with respect to the robots coordinates. Turnings of the head are already regarded by RoboNewbie, but only by the approximate calculations as described above. Other movements of the robot like turning or walking are not regarded. Time stamps indicate the last time of observing an object. The example "Agent_TestLocalFieldView" illustrating the perception features of RoboNewbie is provided for the users.

5.3 Motions

All intentional motions are performed by controlling the hinge joints by sending effector values (speed of motors) to SimSpark RCSS. Then the physical simulation engine calculates the effects of the commands regarding physical laws and updates the simulated world accordingly.

Simple motions like turning the head or rising the arms can be easily programmed by the users as in the already mentioned examples. The motions can be controlled using the feedback of hinge joint perceptors. i.e. by sensor-actor coupling, where the delay of observing an action has to be regarded as described in Section 4. There is much room for own experiments of users.

More complicated motions like walking need coordinated movements of different joints, users may learn about these problems after some trials. We have decided to provide keyframe motions in RoboNewbie because they are easily to understand and to design. The interpolation mechanism for keyframe motions in RoboNewbie realizes a linear interpolation - users may implement other interpolation methods like splines if they want. Keyframes are stored as text files which can be edited by any text processing system. Therewith, users could even design and change motions while using the programs as a blackbox.

RoboNewbie comes with a set of predefined keyframe motions for walking, turning, stand up and others. Users can change these motions (by changing the related text files). New motions need an integration into the program "keyframeMotion", details are explained in the tutorial and the source code documentation.

According to simplicity, there are no concepts implemented for interruption of motions: Each motion is performed completely until its end, and there are no cyclic motions, e.g. for walking. Instead, continuous walking can be performed by subsequent calls of a two-step-walk.

The design of keyframe motions is supported by a graphical Motion Editor. It can be downloaded from the RoboNewbie Web page as well. It shows the postures of the robot for selected keyframes. Then the keyframes can be edited in two ways. In the graphical representation the posture can be kneaded into the desired posture with the mouse. Alternatively, each joint angle can be set to specified values which are immediately presented by the graphics. Transitions between keyframes can be defined with specific transition times resulting in a keyframe sequence as usual.

The program "agentKeyframeDeveloper" helps in designing keyframes. A robot performs the motion of the actually edited keyframe file. After each change, the new motion is performed immediately. If the robot falls down during such a motion, it stands up by itself. Another helpful program can be used to mirror keyframes from one side to the other.

The example "agentSimpleWalkToBall" illustrates the motion concepts. As an exercise of the tutorial, the users can change that program to implement obstacle avoidance (walk around the ball without touching it). They can use motions for walk, stop and turn. Additionally, the agent must be able to recognize the ball and to decide for the appropriate motion according to the ball position.

Another exercise is the design of a new motion for kicking the ball. Users can furthermore do their own experiments e.g. with dancing robots.

In general, keyframe motions are useful for special motions like standing up, but they are not so well suited e.g. for walking. Walking is still a challenging problem in Robotics. The users of RoboNewbie will get some understanding about the task. Moreover, the framework is well suited as a basis for other implementations and for Machine Learning by more educated users. But according to our "minimalistic" approach, related implementations are not provided.

5.4 Control Cycle and Decision Making

The basic control cycle follows the classical deliberation approach, often denoted as the "sense–think–act–cycle", or by related similar names. This corresponds closely to the cycle given by SimSpark RCSS: At first, sensations are provided to the agent, then the agent decides for appropriate plans and then it sends the related action commands back to the server.

Critical remarks may come from the community of Embodied Robotics/AI, e.g. concerning the centralistic and symbolic computations in the classical approach. To realize concepts of Embodied Robotics/AI one needs to put more emphasis on local sensor actor coupling, distributed control, embodiment, situatedness, emergent behaviour etc. The real robot Nao as well as its simulated counterpart with the central control (i.e. our agent) are not primarily designed for such purposes. It is possible to design sensor actor couplings and other behavioural concepts in the RoboNewbie framework, too. One might even split the agent into different "parallel" acting parts (implemented e.g. by threads) to simulate distributed controls, but some synchronization is unavoidable by the server cycles of SimSpark RCSS.

At the same time, thinking in terms of the "sense–think–act–cycle" is quite natural for beginners because it reflects some causal dependencies. It provides an intuitive and easily maintainable structure in the design of robots. Therefore, the control cycle in RoboNewbie adopts the related terms for structuring the run-methods of the agents by cyclic calls of methods sense, think and act. The think-method is sometimes omitted in case of simpler ("reactive") agents.

The sense method is reponsible for receiving and processing the perceptor data by the related RoboNewbie methods. The act methods calls the transfer of the agent message with the effector commands. What is left is the further analysis of the perceptor data (e.g. a more elaborated world model) and the decision for plans and actions to be performed by the robot now and possibly in the future. By the given structure of RoboNewbie, all this can be included in the think method. The think method can of course be split into more dedicated deliberation methods which may be organized hierarchically if needed. Again, all this is left to exercises during related courses. RoboNewbie provides just a simple example for illustration, the Agent_SimpleSoccer.

The Agent_SimpleSoccer is able to perform a very simple soccer play: As long as it is behind the ball and sees the opponent goal, it walks forward while pushing the ball with its feets. If the condition is not fulfilled, it turns around

until it sees the ball, walks to the ball, turns around the ball until it sees the opponents goal, and then it starts walking towards the goal again. The decisions are made by a simple decision tree whenever the previous motion is completed (note that motions can not be interrupted as described above).

It is obvious, that the play of Agent_SimpleSoccer can be improved in many ways. This is just what we want: The users can collect many ideas for improvements. Improvements may concern better usage of perception (e.g. by a ball model guiding the search), improved motions (like faster walk), new motions (like kick or dribble), better control (like path planning). It is also possible to have more players on the soccer field such that players can cooperate (e.g. by positioning and passing). This gives room for simple contests during a course.

5.5 Logger

Runtime debugging of programs may be difficult because it affects synchronization with the server. Even simple debug messages printed on System.out may need too much time such that the agent cannot respond in time. It is possible to use the so-called sync mode which lets SimSpark RCSS wait until all agents have sent their messages (see [3]). Alternatively, all debug messages can be collected by the program "Logger" of RoboNewbie. After the agent has finished, the collected messages are printed out. The usage is shown by the programs "Agent_TestPerceptorInput" and "Agent_TestLocalFieldView". Both programs provide also examples for the usage of the getter methods for perceptors.

6 Evaluations

We have tested the RoboNewbie framework at different places. It was used at introductory Robotics courses of about 30 hours during 5-8 days at Ohrid, Warsaw, Novi Sad, and Rijeka, respectively. 20 hours were planned for lectures, 10 hours for introduction and first usages of RoboNewbie. Additional 10-20 hours were used for further experiments by homework.

RoboNewbie served for illustrating experiments and for exercises in connection with the theoretical instructions. The participants of the courses learned to use RoboNewbie during short time and they programmed an improved soccer player at the end. The work with RoboNewbie was helpful to understand the theory, and the final evaluation of the courses by the participants resulted in high marks. Especially the competitions with the improved soccer agent at the end of the courses were motivating.

This was also the case with the participants of a Robotics course at our university, where the students had more time (two months) for their studies and exercises. Students used the time to implement more sophisticated methods and to try out changes of the framework itself (e.g. other interpolation methods for keyframes). But it also turned out, that efforts for more sophisticated controls are limited by the available skills, especially for motions.

7 Conclusion

In contrast to other experiments in Robotics, the RoboNewbie framework can be used without special hardware. It simply needs a computer for simulation of the robot soccer scenario, which is more complex than experiments by many hardware equipments. It is easy to understand and to use after a short introduction. No special knowledge (except basic programming in Java) is required to start with own experiments, and while the users acquire more knowledge, they can work on more challenging tasks.

The "minimalistic approach" is useful especially for short courses and for introductions to longer courses. Later on, the disposability of non-minimalistic more sophisticated methods could be useful for higher level integrative tasks. It is impossible to let students implement all desirable algorithms in the limited time of a course. Joint activities of robots, for example, depend heavily on the available bodily skills and on the capabilities for interaction and coordination.

The practical evaluations have confirmed our expectations on the RoboNewbie project. Beginners in Robotics were able to use the framework after short introductions. They were able to program own methods in parallel to the theoretical concepts and methods provided by classes. Participants have attested the usefulness of own experiences (which again corresponds to our expectations).

Next plans concern the usage of the RoboNewbie framework in Secondary Schools, and the integration into an e-Learning course on Robotics.

We are thankful to the whole RoboCup community, especially to the developers of SimSpark RCSS, to the team magmaOffenburg and to our team NaoTeam Humboldt, and especially to Yuan Xu.

References

1. Taskin Padir and Sonia Chernova (eds.): Special Issue on Robotics Education. IEEE Transactions on Education, Volume:56, Issue: 1, 2013. http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6423944
2. RoboCup - official web page. http://www.robocup.org/. Visited at 20.8.2013
3. SimSpark RCSS Wiki (Documentation of the Simulator). http://simspark.sourceforge.net/Wiki. Visited at 20.8.2013
4. Homepage Team magmaOffenburg. http://robocup.fh-offenburg.de/html/index.htm. Visited at 20.8.2013
5. Aldebaran. http://www.aldebaran-robotics.com/en/. Visited at 20.8.2013
6. RoboNewbie http://www.naoteamhumboldt.de/projects/robonewbie/. Visited at 20.8.2013
7. Russell Smith. Open Dynamic Engine User Guide, 2006. http://www.ode.org. Visited at 20.8.2013
8. Stone, Peter. RoboCup as an Introduction to CS Research. In Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors, RoboCup-2003: Robot Soccer World Cup VII, Lecture Notes in Artificial Intelligence, pp. 28495, Springer Verlag, Berlin, 2004. The material of the course "Autonomous Multiagent Systems" at the University of Texas. (2012) can be found at http://www.cs.utexas.edu/∼todd/cs344m/ Visited at 20.8.2013

Searching for Concepts in Natural Language Part of Fire Service Reports*

Kamil Bąk[1], Adam Krasuski[1], and Marcin Szczuka[2]

[1] Chair of Computer Science, The Main School of Fire Service, Poland
[2] Institute of Mathematics, The University of Warsaw, Poland
krasuski@inf.sgsp.edu.pl,szczuka@mimuw.edu.pl

Abstract. In the article we present the comparison of the information retrieval approaches focused on a searching of specific concepts in a Natural Language part of Fire Service reports. The comparison comprise of searching with use of regular expressions, Latent Semantic Indexing and pre-defined set of search terms. As a case study we selected three concepts which may not be explicitly mentioned in reports, have various meanings (homonymy), or may be replaced by synonyms.
Keywords: natural language processing, information retrieval, search, fire service.

1 Introduction

The Public Security Services in any country are charged with maintaining public safety and emergency assistance. In Poland a large part of public security and safety tasks is the responsibility of the State Fire Service (PSP – from *Państwowa Straż Pożarna* in Polish). As a primary emergency response service the PSP not only deals with fires, but is also charged with technical rescue (e.g., during road collisions, building collapses), chemical emergency response (chemical spills, hazardous material handling), natural disaster response (floods, wildfire, storms and so on) as well as tasks such as removing beehives or inspecting security measures in buildings.

Every time a fire fighting team is dispatched a report of activity shall be created by the commander at the scene. These reports are prepared in a particular, regulated manner and stored in EWID – a computerized incident data reporting system (IDRS) built for this purpose. Each of approximately 500 Fire and Rescue Units (JRG) of the PSP conducts around 3 fire & rescue actions a day. Since after every action a report is created, the total number of reports in EWID is currently around six million.

* This work was partially supported by the Polish National Science Centre grants 2011/01/B/ST6/03867 and 2012/05/B/ST6/03215, and by the Polish National Centre for Research and Development (NCBiR) - grant O ROB/0010/03/001 under Defence and Security Programmes and Projects: "Modern engineering tools for decision support for commanders of the State Fire Service of Poland during Fire&Rescue operations in buildings".

The EWID reporting system is an unparalleled source of information and knowledge about fire&rescue (F&R) operations. Ability to process and analyze this data could help in development of new procedures and protocols as well as aid the optimization of existing ones [1]. The knowledge derived from EWID may be also very helpful in firefighters' training process. For example, if we can retrieve a reference set of descriptions of similar situations from EWID we can apply techniques based on Conversational Case Based Reasoning (CCBR, see [2]) to decide the course of actions for the new situation. In order to use information contained in EWID efficiently and effectively we need to be able to search and summarize reports according to various, possibly changing requirements.

In this paper we focus on one of the particular tasks associated with identification of EWID records that fulfill certain criteria. This corresponds to identification (retrieval) of action reports that describe situation involving a predefined elements (concepts) such as "Hymenoptera insects","mini-bus" or "carbon monoxide". An important factor in that the concept we look for may not be explicitly mentioned in the record. As EWID record comprise of numerical part and Natural Language (NL) description part, we are particularly interested in finding records related to a preset concept even though they do not have corresponding numerical indicators set and the description part is not clearly listing these concepts. We describe a set of techniques that make it possible to cleanse and filter EWID records, most importantly their description part, in such a way that the search/identification is efficient. This involves overcoming typical problems associated with inconsistencies, vagueness and imprecisions that are commonplace in EWID records. Yet another type of problems that we have to overcome is associated with the very nature of NL data. Notions (words) we are looking for may have various meanings (homonymy) or may be replaced by synonyms.

While it is possible to obtain good results using classical search techniques, their application to description part of EWID records is not always viable in practical applications. In a nutshell, they require a person in front of the computer, who is able to resolve inconsistencies (e.g. homonymy), identify meanings and tune filters. In order to ease some of this manual load and extend search scope while retaining acceptable quality of retrieved information we propose to use a combination of language processing and data analysis tools. In our approach texts from the description parts of EWID records are converted to different representation with use of a method known as Latent Semantic Analysis (LSA). Then, a clustering technique is used to find groups of semantically similar concepts. This grouping is then a basis for constructing search and retrieval algorithm. The quality of retrieved result is compared with straightforward manual filtering by means of standard measures from the field of Information Retrieval (IR - see [3]) such as *recall*, *precision*, and *F-measure*[3].

[3] `http://en.wikipedia.org/wiki/F1_score`

In the paper we first introduce the data we work with (Section 2), then we describe the methodology behind our approach (Section 3). The application of the proposed method and results obtained this way are presented in the Section 4. We finish with discussion of results and conclusions in Section 5.

2 Description of Data

Our data set consists of 291 683 F&R reports extracted from the EWID system. They contain information about incidents to which PSP responded in the period between 1992 and 2011. The data is limited to incidents that happened in the City of Warsaw and its surroundings. Out of 291,683 cases in this dataset 136,856 reports represent fires, 123,139 local threats, and 31,688 false alarms.

As already mentioned, each report consists of a numerical attribute section and a natural language *description* part. The attribute section consists of 506 attributes describing various types of incidents. However, depending on the category of incident, the number of attributes that are actually present (have a non-zero value) varies from 120 to 180 per report. Most of the numerical attributes are boolean (True/False), but there are also some numerical values like fire area or amount of water used to extinguish the fire.

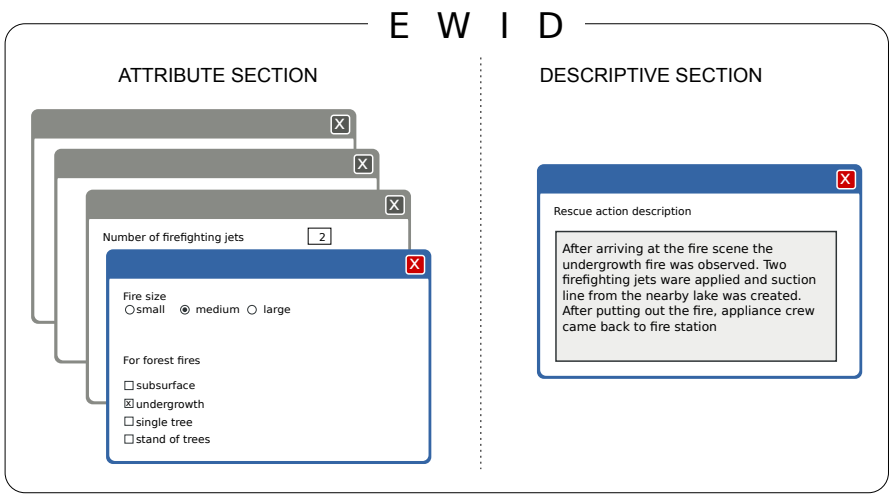


Fig. 1. Representation of a report in EWID database.

The natural language description (NL) part is an extension of the attribute part. It was designed to store information which cannot be represented in the form of a predefined set of attributes. Unfortunately, there are neither clear regulations what should be written in the description part nor any strict guidelines

regarding the format of this part. Therefore, in this part a full spectrum of information can be found. Some descriptions contain detailed information including the precise timeline of events while others are very brief and general. On average, the NL part contains approximately three sentences that describe the situation at the fire ground, actions taken, and weather conditions. Figure 1 depicts the idea of a report representation in EWID database.

In terms of factual aspects the data stored in the EWID contains information about persons, objects involved in the incident, and methods used to eliminate threats that have arisen.

For the purpose of this study we decided to concentrate on three types of incidents that are of some importance to overall management of Fire Service. These are:

1. Incidents where carbon monoxide was present. This mostly concerns fires in residential buildings as carbon monoxide poisoning is one of the major threats in such incidents and the cause of major part of fatalities.
2. Incidents with insects of the Hymenoptera order such as honeybees, hornets, bumblebees, wasps. These incidents fall into the local threats category. Even though they are rarely a major problem, these incidents require relatively high amount of manpower and involvement of specialized equipment.
3. Road collisions involving mini-buses. This category of road accidents is somewhat special. A *mini-bus* in the Polish terminology is a vehicle that is registered to carry between 7 and 12 persons. Such vehicle can be driven by a person with a regular (non-professional) driving permit. In the recent years the accidents involving mini-buses became a major issue in Poland. From security services' point of view they are important, as they may involve many more casualties that "regular" road collisions, and hence require much larger resources to respond to.

In order to perform our experiments we selected from the original data set a sample of 4 135 reports. The records in our subset consist only of NL description part. We extracted this subset using a two-fold procedure. First, using the attribute part we selected the reports which we suspected to contain the kinds of incidents that are of interest to us. Then, using a greedy algorithm based on searching for regular expressions in NL part, we narrowed down the number of previously selected reports to 2 135. In the second step, we selected at random a sample of 2 000 reports, regardless of their kind as a reference sample. Then, we merged this two subsets into one data set for experiments with 4 135 reports in it.

We are fully aware that our data subset may not be sufficiently representative as part of it was not properly, randomly sampled. However, the fully random sample contains too few interesting reports. Therefore, we opted for a compromise combining the fully random sample with the preselected bunch of reports.

The next phase of data preparation involved inspecting (reading) the selected reports one by one and labeling them manually. This step is tantamount to injection of the expert knowledge into the system. We assigned the report to a

category (carbon monoxide, hymenoptera, mini-bus) if it contains the information about a corresponding type of incident. Our final, partly labeled data set contains 82 reports with carbon monoxide intoxication, 167 with road accidents involving mini-buses, and 1557 incidents with Hymenoptera.

3 Methods

Our methodology involves four approaches. In the first approach we adopt a traditional search with use of regular expressions. The user inserts a term or terms which express his information need. He/she defines it using exact or fuzzy search with wild-cards. For example, while searching for reports which describe incidents with *carbon monoxide intoxication* the query can be defined as: "*carbon monoxide*" or "\s CO \s", where *CO* is a chemical symbol for carbon monoxide.

In the second approach the experts define a set of concepts which are related to the defined problem. We transformed these concepts into set of lexemes, i.e., search terms. For example, the problem of finding the reports with carbon monoxide intoxication was defined by the following set of terms: *carbon monoxide, CO, oxide, afterdump, choke-dump, asphyxiate, intoxication.*

In the third approach we we transformed the reports to Latent Semantic Space and performed search using the cosine similarity measure between the query and each of the reports. The fourth approach was similar to the third, but the transformation to LSA was extended by clustering. LSA representations of reports were clustered in order to identify groups of similar incidents.

All the approaches were compared using standard information retrieval measures (recall, precision, and F-measure). In the following subsections we provide some details of our approaches, except for the first one, as it is quite common and simple.

3.1 Search with a set of predefined terms

For all three classes of EWID reports (carbon monoxide, Hymenoptera and mini-buses) we asked domain experts (firefighters) to define the concepts which are related to these problems. They have created a list of concepts which, in their opinion, can express the problem, are associated with it, or occur very often at the emergency scene while responding the particular type of incident. Then, we transformed these concepts into a set of terms. Namely, for the problem of searching carbon monoxide intoxication we defined the following set: *carbon monoxide, co, oxide, afterdump, choke-dump, asphyxiate, intoxicate.* The information need for Hymenoptera-related incidents was defined by the set of terms: *wasp, bee, hornet, bumblebee, insect, cocoon, swarm, ergotizm, gastight clothing.* The terms corresponding to road accidents with mini-buses were: *mini-bus, dostawczy (Polish-specific word), courier.*

As Polish is a fusional language we had to deal with problems posed by inflexion of words. To do that we lemmatized the words in reports from our data

set, creating the non-inflexed form. The lemmatizations were performed with use of the Morfologik software [4].

For each of three incident types we ran a query against the data set using all the terms associated with the given type. The terms were combined in the query using OR operator. The experimental results with this approach are presented in Sections 4 and 5.

3.2 Search using LSA space representation

This approach is based on the transformation of the reports using *Latent Semantic Analysis* (LSA) [5, 6]. The basic idea of LSA is to create the concepts for the given text corpus and then assign each single word from a document (report) to a corresponding concept. The result is that reports can be expressed in Latent Semantic Space as vectors of corresponding concepts' weights. The advantages of LSA representation are that it is considerably more compact than original one and makes it possible to find indirect similarities between reports or between reports and queries.

The reports were lemmatized and transformed into LSA space with use of the R system's [7] library *lsa*. As a result of the transformation we obtained three matrices: *report – concept* matrix (concept in the sense of LSA), *term – concept* matrix and *eigenvalues* matrix. The number of LSA dimensions was established experimentally, based on the values of final measures (precision, recall, F-measure). We found out that the best number of dimensions in this case is 50.

In this approach the search for relevant reports was performed as follows. First, the query (e.g. carbon monoxide) was converted to vector (bag-of-words) form and multiplied by the term-concept matrix in order to obtain its LSA representation. Then, using the cosine similarity measure we find in the report-concept matrix the reports which are similar, as vectors, to the query. The threshold for similarity (cosine between query and report vectors) was established experimentally at 0.7.

3.3 Search using LSA representation and clustering

We have found the results obtained through the usage of the LSA with default settings to be unsatisfactory. In order to improve the results we resorted to cluster analysis. The reports were transformed to LSA representation (report-concept matrix) and then clustered with using the Partitioning Around Medoids (PAM) algorithm [8]. In order to obtain the number of cluster in PAM clustering, we used the silhouette index [9] as a primary measure, complemented with our final performance measures (recall, precision, F-measure). After several repetitions of experiments we have established the desired number of clusters to be 10.

In order to assess the performance of the approach was proceeded as follows. First, we inserted the terms which define our information need, for example "mini-bus". Then, we obtained the names of the clusters which contain such a term. Next, reviewing the reports in these clusters we retained only the clusters

in which the concept (mini-bus) appear in the desired context. In the mini-bus example the desired context would be "road accident with mini-bus". This allows us to eliminate clusters that contain the concept of mini-bus which is not taking part in a road accidents. For example, mini-buses that were burned in parking fire. The similar situation was in the case of carbon monoxide. The clustering helped us in finding the "CO" in the proper context. With carbon monoxide searched as "CO" the situation is tricky because of homonymy. The abbreviation "CO" (uppercase[4]) in Polish is commonly used denote a concept of "central heating" (*Centralne Ogrzewanie*).

4 Results

In Table 1 we show a summary of results obtained from experiments. For each of the methods we calculated the values of three measures: recall precision, and F-measure. The measures were calculated with use of manually labeled reports as the reference.

Table 1. Comparison of search methods.

Method	Measure	Carbon monoxide	Mini-buses	Hymenoptera
Regular expressions	recall	0.451	0.898	0.402
	precision	0.069	0.877	**0.987**
	F-measure	0.120	0.888	0.571
Set of terms	recall	0.671	0.892	**0.990**
	precision	**0.671**	**0.914**	0.981
	F-measure	**0.671**	**0.903**	**0.985**
LSA	recall	0.021	0.347	0.014
	precision	0.011	0.397	0.016
	F-measure	0.012	0.371	0.019
LSA with clustering	recall	**0.768**	**0.928**	0.974
	precision	0.173	0.330	0.557
	F-measure	0.282	0.487	0.709

According to Table 1 the best results were achieved by the approach which used predefined search terms. For each of the classes it obtained the best value of F-measure. Reasonably good results were obtained using the representation of reports in LSA space coupled with clustering.

[4] The situation is even more complicated with a lowercase word "co". It is a common stop word in Polish roughly equivalent – depending of context – to English "what" or "which/that".

5 Discussion and Conclusions

Even though the traditional, term-based search in EWID records returns reasonable results, it is not prefect. In order to get the desired outcome the user of the system must possess some (expert) knowledge of topics from F&R and associated domains. For example, obtaining satisfactory result of search for incidents that involved Hymenoptera requires setting of several filter conditions. Establishing such filtering conditions may be complicated and inconvenient. Similar complications were also symptomatic for other types of searches.

The problems posed by existence of synonyms, homonyms and various elements of specialized jargon had to be overcame. The first attempt was based on analysis of hidden semantic groups derived with use of LSA. Two separate experiments were made. These experiments differ by the operations that were used to transform the (matrix) LSA representation. First of these attempts was made using the *cosine* measure. It measures the angle between the vectors in LSA representation. In this particular case we were interested in measuring the angle between vectors that represent the query and the documents (EWID descriptions). During the experimental evaluation we have determined that this method is inefficient. For most types of queries finding the proper threshold for the value of cosine was problematic. This threshold is used to decide whether a document answers the query or not. Only in the case of querying for incidents involving mini-bus the results were reasonable. In this case we have obtained value of F-measure at 0.37, but the value of precision was merely 0.35. Moreover, the retrieved set of records contained quite high number of incidents involving insects. This may be a result of the two categories (mini-bus-related and insect-related) being identified as semantically close. This semantical closeness is most likely a result of existence of several reports involving both kinds of incidents.

As the results obtained with simple cosine approach were far from satisfactory we had to look for improvements by changing the way the LSA representation was used. In the next attempt we divided the corpus of texts into a pre-set number of clusters. After several experiments we have determined that the best number of clusters in this case is ten. Each cluster was meant to contain reports that share similar context. It was indeed possible to perform clustering in such a way that the clusters were semantically consistent. The best cluster was associated with incidents involving mini-buses and contained 93% of relevant reports. This was, in fact, the best single result we have obtained with any of the methods used. Unfortunately, results obtained with use of this cluster were inferior to manual retrieval attempts since the precision value for this record is only 0.33. This means that a large number of incidents involving types vehicles other than mini-bus was also present in this cluster. The clustering helped to increase the quality of searches involving Hymenoptera. In this case, union of four most prominent clusters contained 97% of all relevant incident reports. The problem with relatively low precision of this approach still remains. For Hymenoptera queries the clusters provided precision at the level of 0.56, which yields the value of F-measure at 0.709. To put this in context, the manual filtering on lemmatized corpus had value of F-measure at 0.985. Clustering approach provided also some

results for queries involving carbon monoxide. One of clusters that were found for this case had the recall 0.768, which is the highest value that we have achieved in all of our experiments. Nevertheless, this cluster is still only marginally useful, as it has very low precision, resulting in value of F-measure that is only 0.282.

To sum up, the best overall results w.r.t. precision, recall, and F-measure were obtained using the semi-manual retrieval with pre-defined set of search terms. This can be further improved by lemmatization which, combined with removal of additional stop words (using Morfologik library), significantly reduces data size and hence the computational effort. The downside of this approach is the necessity of defining and manually entering several search terms. Experiments show that entering search terms one-by-one is ineffective. The have to be several of them in the filter so that they cover a broad range of possible combinations that may occur in incident descriptions. This requires the user to have a good overview of the data corpus and some domain knowledge about incidents stored in EWID.

The problem with requirements for extended users' expertise can be to some reasonable extent – as our initial experiments show – addressed with use of LSA. Conversion of EWID description to LSA representation makes it possible to group (cluster) similar reports. In order to make the demonstrated approach usable we would have to prepare tools that allow user to navigate through the cluster in an intuitive and efficient manner. In particular, the user can be presented with clusters that are represented by a selection of frequent and relevant terms that occur in descriptions that belong to such clusters. Yet another possible extension of our approach could make use of hierarchical clustering. Last, but not the least, we are considering building a search engine that would make it possible for user to perform a faceted search (see [10]) for relevant reports with use of clustering. Facets such as the number and relevance of search terms in aa given cluster may be then used to discriminate between the valuable information and noise.

References

1. Krasuski, A., Kreński, K., Łazowy, S.: A Method for Estimating the Efficiency of Commanding in the State Fire Service of Poland. Fire Technology **48**(4) (2012) 795–805
2. Aha, D.W., Breslow, L.A., Muñoz Avila, H.: Conversational case-based reasoning. Applied Intelligence **14**(1) (2001) 9–32
3. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to information retrieval. Cambridge University Press (2008)
4. Morfologik: About the project. `http://morfologik.blogspot.com/2006/05/about-project.html`
5. Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science **41**(6) (1990) 391–407
6. Landauer, T., Foltz, P., Laham, D.: An introduction to Latent Semantic Analysis. Discourse Processes **25**(2) (1998) 259–284

7. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. (2008)
8. Reynolds, A., Richards, G., De La Iglesia, B., Rayward-Smith, V.: Clustering rules: a comparison of partitioning and hierarchical clustering algorithms. Journal of Mathematical Modelling and Algorithms **5**(4) (2006) 475–504
9. Rousseeuw, P.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics **20** (1987) 53–65
10. Sacco, G.M., Tzitzikas, Y., eds.: Dynamic Taxonomies and Faceted Search. Volume 25 of The Information Retrieval Series. Springer Berlin Heidelberg (2009)

A Rule Format for Rooted Branching Bisimulation

Valentina Castiglioni, Ruggero Lanotte, and Simone Tini

Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria, Como, Italy

Abstract. SOS rule formats are sets of syntactical constraints over SOS transition rules ensuring semantical properties of the derived LTS. Given a rule format, our proposal is to try to relax the constraints imposed by the format on each single transition rule at the price of introducing some reasonable constraint on the form of the whole set of rules, obtaining a new format ensuring the same semantical property and being less demanding than the original one. We demonstrate that this can be done by applying such an idea to a well established rule format ensuring the property of congruence for the rooted branching bisimulation equivalence.

1 Introduction

Structural operational semantics [1] (SOS) is a standard framework to provide process description languages with a semantics. The abstract syntax of a language is given through a *signature*, namely a set of operators together with their arity. The semantics is given through a *labeled transition system* (LTS), namely a set of states that represent processes and that are terms over the signature, together with a set of transitions between states describing computational steps. An LTS is defined by means of a *transition system specification* (TSS), namely a set of *transition rules* of the form $\frac{\text{premises}}{\text{conclusion}}$, which, intuitively, permit to derive transitions between processes from transitions between other processes.

To abstract away from information carried by an LTS that may be considered irrelevant in a given application context, several notions of *behavioral equivalence* were defined (see [2, 3]). Some of these equivalences, like *weak bisimulation* [4] and *branching bisimulation* [5], equate LTS states that incorporate so called *silent steps* representing internal moves by processes that are not observable by the external environment, and are referred to as *weak equivalences*. To ensure compositional modelling and verification, it is crucial that a given behavioral equivalence be a *congruence* w.r.t. all operators of the signature.

Since [6] a *transition rule format* is a set of syntactical constraints on the rules of the TSS, aiming to ensure a given property of the LTS. Several rule formats were developed to ensure the property of congruence for a given behavioral equivalence (see [7] for a survey). The original formats for weak equivalences were proposed in [8]. At present, the standard formats are those in [9–11].

Sometimes the syntactical constraints imposed by the formats on the premises of the transition rules are quite strict. For instance, the rule formats in [9–11]

prohibit the features of *lookahead*, namely the ability to test for two consecutive moves by a process, and *double testing* for running processes, namely the ability to test for two different moves by a process. Our idea is that in some cases it is worth to relax the constraints on the single rules, at the price of introducing some constraints on the form of the whole set of transition rules in the TSS, provided these are not too heavy. To demonstrate how this can be done, we consider the format of [9] for rooted branching bisimulation, we relax the constraints of this format by admitting both lookahead and double testing, and we add the reasonable constraint that lookahead and double testing come together with the ability of testing for an arbitrary number of silent steps, which means introducing a constraint on the set of rules of the TSS since a single rule is required for each number of silent steps. A natural extension of our work is to apply the same strategy to the formats of [10].

The paper is organized as follows: in Section 2, we recall some base notions on SOS, in Section 3 we recall the format of [9], in Section 4 we present our congruence format for rooted branching bisimulation and we end with some conclusions and discussion of related work in Section 5.

2 Preliminaries

In this section we recall some definitions that are standard in the SOS framework.

As usual, we assume that the abstract syntax of a process description language is given by a *signature*, namely a structure $\Sigma = (F, \mathsf{r})$, where (i) F is a set of *function names*, also called *language operators*, and (ii) $\mathsf{r} : F \to \mathbb{N}$ is a *rank function*, which gives the arity of a function name. An operator $f \in F$ is called a *constant* if $\mathsf{r}(f) = 0$. We also assume a set of (process) variables $\mathcal{V}$ disjoint from F, and let x, y, z range over $\mathcal{V}$. Let $W \subseteq \mathcal{V}$ be a set of variables. The set of Σ-terms over W, notation $T(\Sigma, W)$, is the least set satisfying: (i) $W \subseteq T(\Sigma, W)$, and (ii) if $f \in F$ and $t_1, \cdots, t_{\mathsf{r}(f)} \in T(\Sigma, W)$, then $f(t_1, \cdots, t_{\mathsf{r}(f)}) \in T(\Sigma, W)$. $T(\Sigma, \emptyset)$ is the set of all *closed terms*, also called *processes*, and abbreviated as $\mathcal{T}(\Sigma)$. $T(\Sigma, \mathcal{V})$ is the set of all *open terms* and abbreviated as $\mathbb{T}(\Sigma)$. By $\equiv$ we denote the syntactical equality relation between terms. Finally, $Var(t) \subseteq \mathcal{V}$ denotes the set of variables in term t, namely $Var(x) = \{x\}$ and $Var(f(t_1, \ldots, t_{\mathsf{r}(f)})) = \bigcup_{i=1}^{\mathsf{r}(f)} Var(t_i)$.

In SOS framework, the semantic model is that of LTSs.

Definition 1 (Labeled Transition System). *A* Labeled Transition System (LTS) *is a triple $(\mathcal{T}(\Sigma), \mathcal{A}, \rightarrow)$, where i) Σ is a signature; ii) $\mathcal{A}$ is a countable set of actions; and iii) $\rightarrow \subseteq \mathcal{T}(\Sigma) \times \mathcal{A} \times \mathcal{T}(\Sigma)$ is a transition relation.*

Following standard notation, we write $t \xrightarrow{a} t'$ for $(t, a, t') \in \rightarrow$. This represents a computation step of kind a taking process t to process t'.

LTSs are built by means of transition systems specifications, namely sets of transition rules of the form $\frac{\text{premises}}{\text{conclusion}}$. Here we assume that these rules are in the *ntyft*-format [12]. This choice is reasonable since ntyft-format is very general and for transition system specifications that are complete (see Def. 5 below)

it guarantees that bisimilarity equivalence relation is a congruence w.r.t. all operators in F.

Definition 2 (*ntyft*-rule, [12]). *A* ntyft-rule *is of the form*

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \qquad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\}}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t}$$

with J, K at most countable sets of indexes, $t_j, t_k, t \in \mathbb{T}(\Sigma)$, $a_j, b_k, a \in \mathcal{A}$, $y_j \in \mathcal{V}$, $f \in F$, $x_1, \ldots, x_{\mathsf{r}(f)} \in \mathcal{V}$, such that:

- *the $x_1, \ldots, x_{r(f)}$ and the y_j for $j \in J$ are all distinct variables.*

The expressions $t_j \xrightarrow{a_j} y_j$ (resp. $t_k \xrightarrow{b_k} \not\rightarrow$) above the line are called *positive* (resp. *negative*) *premises*. Given a rule ρ, we denote the set of positive (resp. negative) premises by $\mathrm{pprem}(\rho)$ (resp. $\mathrm{nprem}(\rho)$), and the set of all premises by $\mathrm{prem}(\rho) = \mathrm{pprem}(\rho) \cup \mathrm{nprem}(\rho)$. The expression $f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t$ below the line is called *conclusion*, notation $\mathrm{conc}(\rho)$, where $f(x_1, \ldots, x_{\mathsf{r}(f)})$ is called the *source* of ρ, notation $\mathrm{src}(\rho)$, the x_i are the *source variables* denoted by $x_i \in \mathrm{src}(\rho)$, and term t is the *target* of ρ, notation $\mathrm{trgt}(\rho)$. We denote the set of variables in ρ by $\mathit{Var}(\rho)$, *free variables* by $\mathrm{free}(\rho) = \mathit{Var}(\rho) \setminus (\{x_1, \ldots, x_{\mathsf{r}(f)}\} \cup \{y_j \mid j \in J\})$, and *bound variables* by $\mathrm{bound}(\rho) = \mathit{Var}(\rho) \setminus \mathrm{free}(\rho)$.

Definition 3 (*ntyft*-TSS, [12]). *A* ntyft-transition system specification*, ntyft-TSS for short, is a set of ntyft-rules.*

Assigning an LTS to a TSS having rules with negative premises is not trivial. See [7] for a deep discussion. Let us describe the approach we adopt here, namely that of *least three-valued stable model*, introduced in [13] in logic programming.

An expression $t \xrightarrow{a} t'$ (resp. $t \xrightarrow{a} \not\rightarrow$) is called a *positive* (resp. *negative*) *literal* where $t, t' \in \mathbb{T}(\Sigma)$ and $a \in \mathcal{A}$. So, premises and conclusions in rules are literals.

A *substitution* is a mapping $\sigma_{\mathcal{V}} : \mathcal{V} \to \mathbb{T}(\Sigma)$. A substitution is *closed* if it maps each variable to a closed term in $\mathcal{T}(\Sigma)$. A substitution extends to terms by $\sigma_{\mathcal{V}}(f(t_1, \ldots, t_{\mathsf{r}(f)})) = f(\sigma_{\mathcal{V}}(t_1), \ldots, \sigma_{\mathcal{V}}(t_{\mathsf{r}(f)}))$, to literals by $\sigma_{\mathcal{V}}(t \xrightarrow{a} t') = \sigma_{\mathcal{V}}(t) \xrightarrow{a} \sigma_{\mathcal{V}}(t')$ and $\sigma_{\mathcal{V}}(t \xrightarrow{b} \not\rightarrow) = \sigma_{\mathcal{V}}(t) \xrightarrow{b} \not\rightarrow$, and to *ntyft*-rules by $\sigma_{\mathcal{V}}(\rho) = \frac{\bigcup_{\pi \in \mathrm{prem}(\rho)} \sigma_{\mathcal{V}}(\pi)}{\sigma_{\mathcal{V}}(\mathrm{conc}(\rho))}$. A closed substitution instance of a *ntyft*-rule is called a *closed ntyft-rule*. We denote with $\frac{H}{\pi}$ a closed *ntyft*-rule, and with $\frac{N}{\pi}$ a closed *ntyft*-rule having only negative premises (i.e. all elements in N are negative literals).

Given a set of closed positive literals P, a collection of closed negative literals N *holds* for P, denoted $P \models N$, iff for each $t \xrightarrow{b} \not\rightarrow \in N$ we have that $t \xrightarrow{b} t' \notin P$ for any $t' \in \mathcal{T}(\Sigma)$.

Definition 4 (Proof of a closed transition rule). *A* proof *from a TSS T of a closed transition rule $\frac{H}{\pi}$ is an upwardly branching tree in which all upwardly paths are finite, and the nodes are labeled by closed literals such that:*

- *the root is labeled by* π*;*
- *if* K *is the set of the labels of the nodes directly above a node labeled* l*, then:*
  - *either* $K = \emptyset$ *and* $l \in H$*;*
  - *or* $\frac{K}{l}$ *is a closed substitution instance of a transition rule in* T*.*

Given a TSS T, we consider a partitioning of the collection of positive literals to three disjoint sets: i) the set C of positive literals that are *certainly true*; ii) the set U of positive literals for which it is *unknown* whether or not they are true; and iii) the set of remaining literals that are *false*. Such a partitioning, determined by C and U, constitutes a three-valued stable model, denoted $\langle C, U\rangle$, for T if:

- a positive transition π is in C if and only if T proves a closed transition rule $\frac{N}{\pi}$, where N contains only negative literals and $C \cup U \models N$;
- a positive transition π is in $C \cup U$ if and only if T proves a closed transition rule $\frac{N}{\pi}$, where N contains only negative literals and $C \models N$.

Each TSS T allows an (information-)least three-valued stable model $\langle C, U\rangle$, in the sense that the set U is maximal. In [14] *two-valued stable models* were studied, which are three-valued stable models for which the set of unknown positive literals is empty.

Definition 5 (Complete TSS, [15]). *A TSS is* complete *if its least three-valued stable model is a two-valued stable model.*

If a TSS is complete, then it allows only one three-valued stable model, which is taken as the LTS built from the TSS. Only complete TSSs are considered to be meaningful. Notice that a TSS that does not contain transition rules with negative premises is complete for sure.

3 Rooted Branching Bisimulation as a Congruence

Behavioral equivalence relations over processes are usually defined to abstract away information provided by an LTS which is not considered to be relevant for a given application context. Here we consider branching bisimulation, one of those that identify LTS states that incorporate so called silent steps.

In the following we assume that $\mathcal{A}$ contains the special *silent action* τ. The reflexive and transitive closure of relation $\xrightarrow{\tau}$ is denoted with $\xrightarrow{\varepsilon}$. Finally, let us introduce notation $t \xrightarrow{\varepsilon}_n t'$ for $n \in \mathbb{N}$: we have $t \xrightarrow{\varepsilon}_0 t'$ if $t \equiv t'$ and $t \xrightarrow{\varepsilon}_{n+1} t'$ if $t \xrightarrow{\tau} t''$ and $t'' \xrightarrow{\varepsilon}_n t'$ for some $t'' \in \mathcal{T}(\Sigma)$. Hence, $\xrightarrow{\varepsilon} = \bigcup_{n\in\mathbb{N}} \xrightarrow{\varepsilon}_n$.

Definition 6 (Branching bisimulation, [5]). *Take a three-valued stable model* $\langle C, U\rangle$*. A symmetric relation* $\mathcal{B}$ *over* $\mathcal{T}(\Sigma)$ *is a* branching bisimulation *with respect to* C *if whenever* $s\,\mathcal{B}\,t$ *and* $s \xrightarrow{a} s' \in C$ *we have:*

- *either* $a = \tau$ *and* $s'\,\mathcal{B}\,t$*;*
- *or* $t \xrightarrow{\varepsilon} t'', t'' \xrightarrow{a} t' \in C$ *for* $t', t'' \in \mathcal{T}(\Sigma)$ *such that* $s\,\mathcal{B}\,t''$ *and* $s'\,\mathcal{B}\,t'$*.*

We call s and t *branching bisimilar* if there exists a branching bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$. The union of all branching bisimulations over $\mathcal{T}(\Sigma)$ is the greatest branching bisimulation over $\mathcal{T}(\Sigma)$, it is called *branching bisimilarity* and it is denoted with $\leftrightarrows_{bb}$. Branching bisimilarity is an equivalence relation [16].

A crucial property of process description languages to ensure compositional modelling and verification is the compatibility of process operators with the behavioral relation chosen for the application context. In algebraic terms the compatibility of a behavioral equivalence R with operator $f \in F$ is a congruence.

Definition 7 (Congruence). *An equivalence relation R over $\mathcal{T}(\Sigma)$ is a* congruence *if for all $f \in F$, $f(s_1, \ldots, s_{\mathsf{r}(f)})\,R\,f(t_1, \ldots, t_{\mathsf{r}(f)})$ whenever $s_i\,R\,t_i$ for $i = 1, \ldots, \mathsf{r}(f)$.*

Branching bisimulation is not a congruence for the nondeterministic choice operator + defined by rules $\frac{x_1 \xrightarrow{a} y_1}{x_1+x_2 \xrightarrow{a} y_1}$ and $\frac{x_2 \xrightarrow{a} y_2}{x_1+x_2 \xrightarrow{a} y_2}$, which is offered by most of process description languages in the literature. To remedy to this problem the rootedness condition is usually assumed.

Definition 8 (Rooted branching bisimulation). *Take a three-valued stable model $\langle C, U\rangle$. A symmetric relation $\mathcal{R}$ over $\mathcal{T}(\Sigma)$ is a* rooted branching bisimulation *with respect to C if whenever $s\,\mathcal{R}\,t$ and $s \xrightarrow{a} s' \in C$ we have $t \xrightarrow{a} t' \in C$ for t' such that $s' \leftrightarrows_{bb} t'$.*

We call s and t *rooted branching bisimilar* if there exists a rooted branching bisimulation relation $\mathcal{R}$ such that $s\,\mathcal{R}\,t$. The union of all rooted branching bisimulations over $\mathcal{T}(\Sigma)$ is the greatest rooted branching bisimulation over $\mathcal{T}(\Sigma)$, it is called *rooted branching bisimilarity* and it is denoted with $\leftrightarrows_{rb}$. Rooted branching bisimilarity is clearly an equivalence relation.

In the following we recall the rule format RBB-safe [9], which ensures that rooted branching bisimulation is a congruence for all operators in the TSS.

A *patience rule* for the i-th argument of a function symbol $f \in F$ is a *ntyft*-rule of the form

$$\frac{x_i \xrightarrow{\tau} y_i}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{\tau} f(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_{\mathsf{r}(f)})}$$

Following [9], we assume that each argument of each function symbol $f \in F$ is labeled either *tame* or *wild*. A *context*, denoted with $C[\,]$, is an open term in $\mathbb{T}(\Sigma)$ with one occurrence of the *context symbol* $[\,]$.

Definition 9 (w-nested context, [9]). *The collection of* w-nested contexts *is defined inductively by:*

- $[\,]$ *is w-nested;*
- *if $C[\,]$ is w-nested, and argument i of function symbol f is wild, then also $f(t_1, \ldots, t_{i-1}, C[\,], t_{i+1}, \ldots, t_{\mathsf{r}(f)})$ is w-nested.*

Definition 10 (RBB safe TSS, [9]). *A TSS T is called* RBB safe*, with respect to a tame/wild labeling of arguments of function symbols in F, if each of its transition rules is*

1. *either a patience rule for a wild argument of a function symbol,*
2. *or a ntyft-rule ρ with source $f(x_1, \ldots, x_{r(f)})$ and right-hand sides of positive premises $\{y_j \mid j \in J\}$, such that the following requirements are fulfilled:*
   (a) *Variables y_j for $j \in J$ do not occur in left-hand sides of premises of ρ;*
   (b) *If argument i of f is wild and does not have a patience rule in T, then x_i does not occur in left-hand sides of premises of ρ;*
   (c) *If argument i of f is wild and has a patience rule in T, then x_i occurs in the left-hand side of no more than one premise of ρ, where this premise*
      i. is positive,
      ii. does not contain the relation $\xrightarrow{\tau}$, and
      iii. has left-hand side x_i;
   (d) *Variables y_j for $j \in J$ and variables x_i for i a wild argument of f may only occur at w-nested positions in the target of ρ.*

Theorem 1 (Rooted branching bisimulation as a congruence, [9]). *If a complete TSS is RBB safe, then the rooted branching bisimulation equivalence that it induces is a congruence.*

In [9] several counter-examples are given to show that the syntactic constraints of Def. 10 cannot be relaxed in any trivial way. Our aim is to show that the constraints that prohibit *lookahead* (constraint 2a) and *double testing* for wild arguments of operators (constraint 2c) on the single rules of the TSS can be relaxed, provided that suitable and reasonable (i.e non too-demanding) constraints on the whole set of rules of the TSS are introduced. In the following we assume a TSS T containing the CCS-like sequencing operator $\cdot$ defined by the rules $\frac{}{a \cdot x \xrightarrow{a} x}$ for $a \in \mathcal{A}$, the CCS-like nondeterministic choice operator $+$ recalled above, the idle process 0 and the unary operators f_1, f_2 defined below.

Constraint 2a in Def. 10 prohibits *lookahead*, namely the ability of testing for two (or more) subsequent moves by a source argument. An example of rule violating this constraint is the following rule ρ_{f_1} for operator f_1:

$$\frac{x_1 \xrightarrow{a} y_1 \quad y_1 \xrightarrow{b} y_2}{f_1(x_1) \xrightarrow{a} 0}$$

Let us consider processes $s \equiv a \cdot b \cdot 0$ and $t \equiv a \cdot \tau \cdot b \cdot 0$. We have $s \leftrightarrows_{rb} t$. However, we have $f_1(s) \xrightarrow{a} 0$ while $f_1(t) \not\xrightarrow{a}$. Thus $f_1(s) \not\leftrightarrows_{rb} f_1(t)$. In this example the role of the silent action τ in the definition of t is crucial. On one side, the capability of performing τ is not discriminating in the evaluation of branching bisimulation equivalence of processes (see Def. 6). Hence, processes $b \cdot 0$ and $\tau \cdot b \cdot 0$, which are reached through action a by s and t, respectively, are branching bisimilar, thus implying that $s \leftrightarrows_{rb} t$. On the other hand, action τ becomes relevant when we focus on exact process evolution sequences. While process s can immediately perform b after a, process t cannot, namely after performing a it has to do the

action τ to reach a state in which it is able to perform b, and these two different evolutions are discriminated by the premises of ρ_{f_1}. Our proposal is to permit testing for an a move followed by a b move, provided that this comes together with the testing for an arbitrary number of τ-moves between these two moves labeled a and b. This means admitting the following set of rules in the TSS, provided we introduce the constraint that the TSS contains all of them:

$$\left\{ \frac{x_1 \xrightarrow{a} y_1 \qquad y_1 \xrightarrow{\epsilon}_n y_2 \qquad y_2 \xrightarrow{b} y_3}{f_1(x_1) \xrightarrow{a} y_3} \mid n \in \mathbb{N} \right\}$$

Let τ^n denote the sequence $\tau \cdot \ldots \cdot \tau$ of n actions τ, and $s_n \equiv a \cdot \tau^n \cdot b \cdot 0$ (notice $s \equiv s_0$ and $t \equiv s_1$). We have that $f_1(s_n) \xrightarrow{a} 0$ for all $n \in \mathbb{N}$, thus implying that $f_1(s_m) \leftrightarrows_{rb} f_1(s_n)$ for all $m, n \in \mathbb{N}$.

Constraint 2c in Def. 10 prohibits *double testing*, namely the ability of testing for two (or more) moves by a source argument, for arguments labeled as wild. An example of rule violating this constraint is the second of the rules below:

$$\frac{x_1 \xrightarrow{a} y_1}{f_2(x_1) \xrightarrow{a} f_2(y_1)}\ a \in \mathcal{A} \qquad \frac{x_1 \xrightarrow{a} y_1 \quad x_1 \xrightarrow{b} y_2}{f_2(x_1) \xrightarrow{c} 0}$$

where the argument of f_2 has to be wild due to constraint 2d of Def. 10 applied to the first rule. Let us take processes $s \equiv a \cdot (a \cdot 0 + b \cdot 0)$ and $t \equiv a \cdot t'$, with t' defined with the classical recursive construct as $t' \equiv a \cdot 0 + \tau \cdot (b \cdot 0 + \tau \cdot t')$. We have $s \leftrightarrows_{rb} t$. However, we have $f_2(s) \xrightarrow{a} f_2(a \cdot 0 + b \cdot 0) \xrightarrow{c} 0$ while $f_2(t) \xrightarrow{a} f_2(t')$ and neither $f_2(t')$ nor any process reachable from $f_2(t')$ through any sequence of τ-moves can make any c move. Thus $f_2(s) \not\leftrightarrows_{rb} f_2(t)$. Here the processes $a \cdot 0 + b \cdot 0$ and t', which are reached through action a by s and t, respectively, are branching bisimilar. Their difference, sensed by the second rule for f_2, is that $a \cdot 0 + b \cdot 0$ can perform both a and b, whereas t' is not able to reach (through any sequence of τ moves) any state in which both a and b are enabled, despite it can reach through τ actions a state where a is enabled and another state where b is enabled. Our proposal is to permit double testing for moves a and b, provided that these moves may follow an arbitrary number of τ steps. This means admitting the following set of rules in the TSS, provided we add the constraint that the TSS contains all of them:

$$\left\{ \frac{x_1 \xrightarrow{\epsilon}_m y_1 \quad y_1 \xrightarrow{a} y_2 \quad x_1 \xrightarrow{\epsilon}_n y_3 \quad y_3 \xrightarrow{b} y_4}{f_2(x_1) \xrightarrow{c} 0} \mid m, n \in \mathbb{N} \right\}$$

Notice that with these rules we get both $f_2(a \cdot 0 + b \cdot 0) \xrightarrow{c} 0$ and $f_2(t') \xrightarrow{c} 0$, thus implying $f_2(s) \leftrightarrows_{rb} f_2(t)$.

4 Congruence Format for Rooted Branching Bisimulation

As discussed in the previous section, lookahead and double testing can be admitted in the RBB safe format of [9], provided that sets of rules testing for sequences

of τ moves of different length are all introduced in the TSS. Below we introduce the notion of meta transition rule, which denotes a set of transition rules that test for the ability of performing sequences of τ moves of all possible lengths.

Definition 11 (Positive meta premise). *A* positive meta premise *is an expression of the form*

$$t \Longrightarrow \xrightarrow{a} y$$

The meta premise $t \Longrightarrow \xrightarrow{a} y$ represents the set

$$[\![t \Longrightarrow \xrightarrow{a} y]\!] := \left\{ \{ t \xrightarrow{\varepsilon}_n y' \quad y' \xrightarrow{a} y \} \mid n \in \mathbb{N} \right\}$$

of countable many sets of premises. Intuitively, $t \Longrightarrow \xrightarrow{a} y$ holds if there exists an $n \in \mathbb{N}$ and a substitution $\sigma_{\mathcal{V}}$ such that $\sigma_{\mathcal{V}}(t)$ can reach a state able to perform the action a through a sequence of n τ-actions.

Definition 12 (Meta transition rule). *A* meta transition rule*, notation $\tilde{\rho}$, is of the form*

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{z_l \Longrightarrow \xrightarrow{a_l} y_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathrm{r}(f)}) \xrightarrow{a} t}$$

with J, K, L at most countable sets of indexes, $t_j, t_k, t \in \mathbb{T}(\Sigma)$, $a_j, b_k, a_l, a \in \mathcal{A}$, $y_j, z_l, y_l \in \mathcal{V}$, $f \in F$, $x_1, \ldots, x_{\mathrm{r}(f)} \in \mathcal{V}$, such that:

- *the $x_1, \ldots, x_{\mathrm{r}(f)}$, the y_j for $j \in J$ and the y_l for $l \in L$ are all distinct variables.*

A meta transition rule $\tilde{\rho}$ like in Def. 12 represents the set $[\![\tilde{\rho}]\!]$ of all the transition rules of the form

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{\mu_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathrm{r}(f)}) \xrightarrow{a} t}$$

such that $\mu_l \in [\![z_l \Longrightarrow \xrightarrow{a_l} y_l]\!]$.

Definition 13 (Meta TSS). *A* meta TSS *is a set of meta transition rules.*

The meta TSS T represents the TSS $[\![T]\!] = \cup_{\tilde{\rho} \in T} [\![\tilde{\rho}]\!]$. Clearly, $[\![T]\!]$ is a *ntyft*-TSS. So all definitions of Section 2 directly lift to meta TSSs.

Now, we are able to extend the RBB safe format of [9] with lookahead and double testing for running processes.

Definition 14 (Meta RBB safe TSS). *A meta TSS T is called* meta RBB safe*, with respect to a tame/wild labeling of arguments of function symbols in F, if each of its transition rules is*

1. *either a patience rule for a wild argument of a function symbol;*

2. or a meta transition rule $\tilde{\rho}$ of the form

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{z_l \Longrightarrow \xrightarrow{a_l} y_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t}$$

with constraints:

(a) actions a_l, for $l \in L$, are in $\mathcal{A} \setminus \{\tau\}$, namely they can not be action τ;
(b) if $x_i \in [\bigcup_{j\in J} var(t_j) \cup \bigcup_{k\in K} var(t_k)]$ for $i = 1, \ldots, \mathsf{r}(f)$, then i is a tame argument for f;
(c) no y_j for $j \in J$ and y_l for $l \in L$ occurs in $[\bigcup_{j\in J} var(t_j) \cup \bigcup_{k\in K} var(t_k)]$;
(d) variables x_i for i wild argument of f, y_j for $j \in J$ and y_l for $l \in L$ may occur only at w-nested positions in the target t.

Notice that Def. 14 admits lookahead, since for $l \in L$ we may have that $z_l \equiv y_j$ for some $j \in J$ or $z_l \equiv y_{l'}$ for some $l' \in L$. Double testing for a wild argument i of an operation $f \in F$ is admitted since we may have $z_l \equiv z_{l'} \equiv x_i$ for $l, l' \in L$.

Let us remark that meta rules have been already used in [10], called GSOS rules with lookahead, with the purpose of observing a partial form of lookahead, namely a sequence of τ-moves followed by a non silent move.

Notice that in Def. 14 we do not need the constraint 2b of Def. 10, which imposes that testing for a move by a wild argument for an operator f requires that there is a patience rule for it. To explain the reason, let us take the operators

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} g(y_1)} \qquad \frac{x_1 \xrightarrow{b} y_1}{g(x_1) \xrightarrow{b} g(y_1)}$$

that do not respect Def. 10 since the patience rule for the argument of g is missing, and processes $a \cdot b \cdot 0$ and $a \cdot \tau \cdot b \cdot 0$. We have $a \cdot b \cdot 0 \leftrightarrows_{rb} a \cdot \tau \cdot b \cdot 0$ but $f(a \cdot b \cdot 0) \not\leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$ since $f(a \cdot b \cdot 0) \xrightarrow{a} g(b \cdot 0) \xrightarrow{b} 0$ whereby $f(a \cdot \tau \cdot b \cdot 0) \xrightarrow{a} g(\tau \cdot b \cdot 0) \xrightarrow{b} \not\rightarrow$. Definition 10 requires the patience rule for the argument of g, so $g(\tau \cdot b \cdot 0) \xrightarrow{\tau} g(b \cdot 0) \xrightarrow{b} 0$ and, therefore, $f(a \cdot b \cdot 0) \leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$. By adopting the meta rules as in Def. 14, we can write

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} g(y_1)} \qquad \frac{x_1 \Longrightarrow \xrightarrow{b} y_1}{g(x_1) \xrightarrow{b} g(y_1)}$$

and the patience rule for the argument of g is no more needed, since we have $f(a \cdot \tau \cdot b \cdot 0) \xrightarrow{a} g(\tau \cdot b \cdot 0) \xrightarrow{b} 0$ and, thus, $f(a \cdot b \cdot 0) \leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$.

Let us argue that all constraints in Def. 14 cannot be relaxed in any trivial way. Firstly, let us show why, as in [9], some arguments of functions deserve a special treatment. These arguments are labeled as wild. The special treatment consists in constraints 2b and 2d in Def. 14.

Example 1. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} y_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} f(y_1)}$$

and processes $a.\tau.a.0$ and $a.a.0$. We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, we have $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$. In fact, $g(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0)$ and $g(a.a.0) \xrightarrow{a} f(a.0)$, where $f(\tau.a.0) \not\leftrightarrows_{bb} f(a.0)$ since $f(\tau.a.0) \not\xrightarrow{a}$ and $f(a.0) \xrightarrow{a} 0$. The rule for g has $f(y_1)$ as target, where y_1 occurs in the target of the premise $x_1 \xrightarrow{a} y_1$. This implies that it may happen that when the argument x_1 of g is instantiated by two processes p and p' with $p \leftrightarrows_{rb} p'$, we have that the argument y_1 of f is instantiated by two a-derivatives, q and q' respectively, such that $q \leftrightarrows_{bb} q'$ but $q \not\leftrightarrows_{rb} q'$. Arguments of operators that may be instantiated with processes related by $\leftrightarrows_{bb}$ but not by $\leftrightarrows_{rb}$ are labeled wild. This is exactly what is required by constraint 2d in Def. 14. As required by constraint 2b in Def. 14, they cannot be tested by premises of the form $x \xrightarrow{a} y$ since these premises are able to discriminate them. ■

By next example, we show why meta premises cannot test for τ moves (constraint 2a, Def. 14).

Example 2. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} f(y_1)} \qquad \frac{x_1 \Longrightarrow\!\!\xrightarrow{\tau} y_1}{f(x_1) \xrightarrow{\tau} y_1}$$

We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$. In fact we have $g(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0) \xrightarrow{\tau} a.0 \xrightarrow{a} 0$, whereby $g(a.a.0) \xrightarrow{a} f(a.0) \not\xrightarrow{\tau}$. ■

By next example, we show why in meta premises we cannot have an arbitrary term in the left side, and we only allow variable z_l.

Example 3. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} y_1} \qquad \frac{g(x_1) \Longrightarrow\!\!\xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} y_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{h(x_1) \xrightarrow{a} f(y_1)}$$

We have $a.a.0 \leftrightarrows_{rb} a.\tau.a.0$. However, $h(a.a.0) \not\leftrightarrows_{rb} h(a.\tau.a.0)$. In fact we have $h(a.a.0) \xrightarrow{a} f(a.0) \xrightarrow{a} 0$, whereby $h(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0) \not\xrightarrow{a}$. ■

By next example, we show why we cannot allow variables that are targets of premises or meta premises to be source of classic premises (constraint 2c, Def. 14).

Example 4. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1 \quad y_1 \xrightarrow{a} y_2}{f(x_1) \xrightarrow{a} y_2} \qquad \frac{x_1 \Longrightarrow\!\!\xrightarrow{a} y_1 \quad y_1 \xrightarrow{a} y_2}{g(x_1) \xrightarrow{a} y_2}$$

We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, we have that $f(a.\tau.a.0) \not\leftrightarrows_{rb} f(a.a.0)$ since $f(a.\tau.a.0) \not\xrightarrow{a}$ while $f(a.a.0) \xrightarrow{a} 0$. Analogously, $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$ since $g(a.\tau.a.0) \not\xrightarrow{a}$ while $g(a.a.0) \xrightarrow{a} 0$. ■

To prove the congruence result, we have to deal with well-founded rules.

Definition 15 (Well-foundedness). *Let H be a set of premises and meta premises. The* variable dependency graph *of H is a directed graph $G_H = (V, E)$ given by:*

- $V = \bigcup_{h \in H} Var(h)$;
- $E = \{\langle x, y\rangle \mid t \xrightarrow{a} y \in H \text{ and } x \in Var(t) \text{ or } x \Longrightarrow\!\!\xrightarrow{a} y \in H\}$.

We say that H is well-founded *if any backward chain of edges in G_H is finite. A meta transition rule $\tilde{\rho}$ is called* well-founded *if the set of all its premises and meta premises is well-founded. A meta TSS is called* well-founded *if all its meta transition rules are well-founded.*

Theorem 2. *If a complete and well-founded meta TSS T is meta RBB safe, then the rooted branching bisimulation equivalence that it induces is a congruence.*

5 Conclusions

We considered the format of [9], which ensures the congruence property for rooted branching bisimulation, we relaxed the constraints on the single rules by allowing both double testing for wild arguments of operators and lookahead, at the price of constraining these features to come together the testing for an arbitrary number of silent steps. We argued that this means introducing a constraint on the form of the whole set of rules. Our idea can be naturally extended to the formats in [10, 11].

An example of operator that is captured by our format and that is outside the formats in [9–11] is the *copying* operator, originally proposed in [17] for languages that do not consider silent actions, and defined in [11] by the following rules:

$$\frac{x_1 \xrightarrow{a} y_1}{cp(x_1) \xrightarrow{a} cp(y_1)} \quad a \in \mathcal{A} \qquad\qquad \frac{x_1 \xrightarrow{l} y_1 \quad x_1 \xrightarrow{r} y_1}{cp(x_1) \xrightarrow{s} cp(y_1) \parallel cp(y_2)}$$

where $l, r \in \mathcal{A}$ are the *left* and *right forking*, respectively, s is the *split* action, and $\parallel$ is the parallel composition operator. In [11] this operator is admitted in the format thanks to the *two-tiered* approach to SOS proposed in [10, 11]. The idea is to divide function symbols in F into two classes: *principal operators* and *abbreviations*, where an abbreviation can be obtained by grouping together the arguments of a principal operator. Proofs are given that abbreviations are syntactic sugar and do not have to obey the syntactic restrictions of a congruence format, provided they abbreviate principal operators that do so. This is an advantage since if a given equivalence is a congruence w.r.t. an operator $f \in F$ that is outside from a congruence format, one can find an operator f^* that is considered to be principal and abbreviated by f and that obeys the constraints of the format. In [11] an operator for which cp is an abbreviation is provided that is captured by the format. Here we do not need to search for such an abbreviation since cp is already in the format.

Acknowledgements We are grateful to Wan Fokkink for feedback on a preliminary version of this paper.

References

1. Plotkin, G.: A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University (1981)
2. van Glabbeek, R.J.: The linear time - branching time spectrum. In: CONCUR '90. Volume 458 of LNCS., Springer (1990) 278–297
3. van Glabbeek, R.J.: The linear time - branching time spectrum ii. In: CONCUR '93. Volume 715 of LNCS., Springer (1993) 66–81
4. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. Assoc. Comput. Mach. **32** (1985) 137–161
5. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. Assoc. Comput. Mach. **43**(3) (1996) 555–600
6. de Simone, R.: Higher-level synchronising devices in meije-sccs. Theoret. Comput. Sci. **37** (1985) 245–267
7. Aceto, L., Fokkink, W.J., Verhoef, C.: Structural operational semantics. In: Handbook of Process Algebra. Elsevier (2001) 197–292
8. Bloom, B.: Structural operational semantics for weak bisimulations. Theoret. Comput. Sci. **146** (1995) 25–68
9. Fokkink, W.J.: Rooted branching bisimulation as a congruence. J. Comput. Syst. Sci. **60**(1) (2000) 13–37
10. van Glabbeek, R.J.: On cool congruence formats for weak bisimulations. Theoret. Comput. Sci. **412**(28) (2011) 3283–3302
11. Fokkink, W.J., van Glabbeek, R.J., de Wind, P.W.: Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity. Inf. Comput **214** (2012) 59–85
12. Groote, J.F.: Transition system specifications with negative premises. Theoret. Comput. Sci. **118**(2) (1993) 263–299
13. Przymusinski, T.C.: The well-founded semantics coincides with the three-valued stable semantics. Fundam. Inform. **13**(4) (1990) 445–463
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: 5th Conference on Logic Programming, MIT Press (1988) 1070–1080
15. van Glabbeek, R.J.: The meaning of negative premises in transition system specifications ii. In: ICALP'96. Volume 1099 of LNCS., Springer (1996) 502–513
16. Basten, T.: Branching bisimilarity is an equivalence indeed! Inform. Proc. Lett. **58**(3) (1996) 141–147
17. Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. J. Assoc. Comput. Mach. **42**(1) (1995) 232–268

A Rewriting Based Monitoring Algorithm for TPTL*

Ming Chai and Bernd-Holger Schlingloff

Humboldt University Berlin, Berlin D-10099, Germany
{ming.chai, hs}@informatik.hu-berlin.de

Abstract. In this paper, we present a rewriting based monitoring algorithm for time propositional temporal logic (TPTL), which is a classic time extension of linear temporal logic (LTL). TPTL has been shown to be more expressive than other real-time extensions of LTL, e.g., metric temporal logic (MTL). We first describe the syntax and semantics of TPTL on finite time-traces. Using Maude, which is an executable environment for various logics, we give rewriting clauses to check whether a finite time-trace satisfies a TPTL formula. We use our algorithm to test a concrete example from the European Train Control System (ETCS), and evaluate it on several benchmarks. The results show the feasibility of our approach.

1 Introduction

Runtime verification is proposed for checking whether a run of a system satisfies or violates a given correctness property [1]. It is seen as a lightweight verification technique when compared to model checking and testing. Runtime verification is able to avoid the following problems of model checking: *i)* when checking a high complexity system, model checking could suffer from the so-called state explosion problem; *ii)* when checking a black-box system, a model of the system may not be available for model checking; *iii)* the object of model checking is a model of the system, not the system itself.

Runtime verification is performed by using a *monitor*. This is a device or a piece of software that reads a behavior of the system under monitoring and gives a certain verdict (*true* or *false*) as the result. A behavior of the system is presented by its *trace*, which is an observable execution sequence of the system. Unlike model checking, runtime verification does not check all executions of the underlying system, but a finite trace. Hence it does not suffer from the state explosion problem when dealing with a large system. Furthermore, runtime verification does not need a model of the system. Therefore, it is well suited to check black-box systems. Finally, the checking object of runtime verification is the system itself. Thus, the possibility of introducing additional errors in the modeling is excluded.

* This work was supported by the State Key Laboratory of Rail Traffic Control and Safety (Contract No.: RCS2012K001), Beijing Jiaotong University

One of the most interesting problems in runtime verification is how to build a monitor from a high level specification. Havelund et al. [2] propose a formula rewriting based runtime verification approach, constituting part of a project named Java PathExplorer (JPAX). Their work aims at monitoring Java programs and has been used in Mars Exploration Rover missions. Feng et al. [3] propose an MOP framework for software development and analysis, in which the satisfaction/violation of properties can be detected by executing the code. Barringer et al. [4] propose a rule-based system for trace analysis RuleR. They also propose the LOGSCOPE system, which is an extension of RuleR with a simple, user-friendly temporal logic. d'Amorim et al. present a modified Büchi automata, which is used for monitoring a system [5].

For checking time-relevant properties, real-time logics have been introduced into runtime verification. Bauer et al. [6] work on TLTL based runtime verification for monitoring real-time properties. They define TLTL by introducing two operators ($\triangleright_a \in I$) and ($\triangleleft_a \in I$) with a being an event, and I being a time interval. They build a monitor for a TLTL property, and use event-clock automata to detect whether a trace is accepted or rejected.

Metric temporal logic (MTL) [7] is a well studied real-time logic. It is obtained by extending standard LTL with a time bounded temporal operator $\mathcal{U}_{[a,\ b]}$, where a, b are natural numbers. Several MTL based monitoring approaches have been proposed. Thati et al. [8] propose a formula rewriting based monitoring algorithm for MTL. Nickovic et al. propose monitoring algorithms for a restricted version of MTL, named MITL. Basin et al. [9] propose a monitoring algorithm for metric first-order logic. Their approach can cope with variables ranging over infinite domains. They also develop algorithms for MTL with discrete events and continuous states [10].

Alur et al. [11] propose a "more temporal" real-time logic, named time propositional temporal logic (TPTL). It is obtained from LTL by introducing a freeze quantifier "$x.$". A TPTL formula can "reset" a formula clock at some point by assigning variables in the formula to the time value when the formula is evaluated. The expressiveness of TPTL and MTL is studied in [12, 13]. It has been proven that TPTL is strictly more expressive than MTL. Although the verification and model checking problem for TPTL has been studied intensely, the number of TPTL based runtime verification approaches is quite limited. One example is Kristoffersen et al. [14], who give a monitoring algorithm for LTL_t, which also extends LTL by a freeze quantifier. The difference between TPTL and LTL_t is that the latter needs an extra clock variable r for expressing time.

In this paper, we propose a formula rewriting based runtime verification approach for TPTL. The monitor consists of a TPTL formula and a formula rewriting algorithm, where the formula is generated from a high level specification. The monitor receives a time-trace, which is generated from the underlying system. It detects failures through checking whether this time-trace violates the formula. The process is shown in Fig. 1. Our algorithm is developed directly based on the syntax and semantics of TPTL.

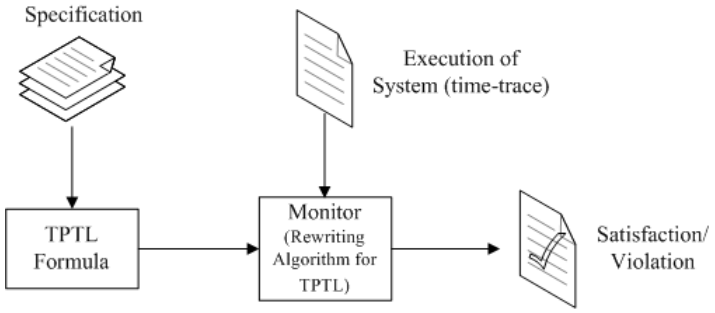


Fig. 1. The runtime verification process

Our algorithm is based on Maude [15], which is a high performance system for model checking, theorem proving, and programming. It can be used for runtime verification implementation. We use the Maude rewriting logic, in the style of the LTL rewriting program proposed by Havelund [16]. Additionally, we present a case study of a concrete example in the railway domain. We translate several properties contained in the specifications of a signaling system to TPTL formulae, and abstract some executions of the system to time-traces. Then we monitor these time-traces in Maude. The results show that our approach is feasible for monitoring time-traces.

The rest part of the paper is organized as follows. Section 2 introduces the definition of TPTL, including the syntax and semantics. Section 3 presents the Maude-based program for TPTL based monitor. Section 4 shows a case study with a concrete example from the railway domain. Section 5 contains the conclusion and future work.

2 Preliminaries

2.1 Time-events and Time-traces

Given a (finite) set of atomic propositions AP and a (finite) alphabet $\Sigma = 2^{AP}$, an *event* is defined as any single element of Σ, i.e. $e = \{p_1, \cdots, p_m\}$ with $p_1, \cdots, p_m \in AP$. If e is a singleton, we omit the curly brackets in the denotation. If we denote the set of natural numbers by $\mathbb{N}_{\geq 0}$ and $t \in \mathbb{N}_{\geq 0}$, then a *time-event* is defined as a pair $te = (e, t)$ from the set $\Sigma \times \mathbb{N}_{\geq 0}$. The natural number t in a time-event te is a discrete *time stamp*, to identify the time of the event emitted by a running real-time system. Given a time-event $te = (e, t)$, we define $Event(te) \triangleq e$ and $Time(te) \triangleq t$. A *time-trace* is defined as a (possibly infinite) sequence of time-events, i. e. $tt = (te\ [0], te\ [1], \cdots, te\ [n])$, where for each $i < n$ with $i \in \mathbb{N}_{\geq 0}$, it holds that $Time\ (te\ [i]) < Time\ (te\ [i+1])$ (strict monotonicity). The length of tt is denoted by $|tt|$.

2.2 Syntax and Semantics of TPTL

LTL is a widely-accepted logic for specifying properties of infinite traces. TPTL is an extension of LTL to express real-time properties. It contains a freeze quantifier "$x.$", which assigns the time value when the formula is evaluated to the variable x. A TPTL formula $x.\ \varphi(x)$ is satisfied by a time-trace tt iff $\varphi(time(tt[0]))$ is satisfied by tt. For instance, a TPTL formula

$$(\Box\ x.\ (Request \rightarrow \Diamond\ y.\ (Ack \wedge y < 5 + x)))$$

expresses the property "whenever an event *Request* occurs, then the acknowledgment event *Ack* must occur within 5 time units". This formula is satisfied, e.g., by the time-trace $(\cdots, (Request, 7), \cdots, (Ack, 11), \cdots)$, since $11 < 5 + 7$. More precisely, TPTL is defined as follows.

Definition 1. ***(Syntax for TPTL)*** *Given the finite set AP of atomic propositions and a set V of free variables, the terms π and formulae φ of TPTL are inductively formed according to the following grammar, where $x \in V$, $r \in \mathbb{N}_{\geq 0}$, $p \in AP$ and $\sim \in \{\leq, <, =, >, \geq\}$:*

$\pi ::= x + r \mid r$

$\varphi ::= \bot \mid p \mid (\varphi_1 \rightarrow \varphi_2\ ) \mid (\varphi_1\ \mathcal{U}\ \varphi_2) \mid \pi_1 \sim \pi_2 \mid x.\ \varphi.$

The following shorthands are used in TPTL as in LTL: $\Diamond\ \varphi$ stands for $\top\ \mathcal{U}\ \varphi$, $\Box\ \varphi$ stands for $\neg\Diamond\ \neg\varphi$, and $\bigcirc\ \varphi$ stands for $\bot\ \mathcal{U}\ \varphi$.

Assume that $\mathcal{E}$ is a function $\mathcal{E}$: $V \rightarrow \mathbb{N}_{\geq 0}$ for assigning free variables in $\mathbb{N}_{\geq 0}$ (time value) such that $\mathcal{E}(x + r) = \mathcal{E}(x) + r$ and $\mathcal{E}(r) = r$. Given a variable x and a natural number r, we denote $\mathcal{E}[x := r]$ for the evaluation $\mathcal{E}'$ such that $\mathcal{E}'(x) = r$, and $\mathcal{E}'(y) = \mathcal{E}(y)$ for all $y \in V \backslash \{x\}$. In runtime verification, the time-traces to be checked are finite. Hence, we give TPTL finite semantics as follows.

Definition 2. ***(Semantics for TPTL)*** *Let tt be a finite trace with $i \in \mathbb{N}_{\geq 0}$ being a position, p a proposition, and φ_1 and φ_2 any TPTL formulae. The satisfaction relation $(tt, i, \mathcal{E}) \models \varphi$ is defined inductively as follows:*

$(tt, i, \mathcal{E}) \not\models \bot$;

$(tt, i, \mathcal{E}) \models p$ *iff* $p \in Event(tt[i])$;

$(tt, i, \mathcal{E}) \models (\varphi_1 \rightarrow \varphi_2)$ *iff* $(tt, i, \mathcal{E}) \models \varphi_1$ *implies* $(tt, i, \mathcal{E}) \models \varphi_2$;

$(tt, i, \mathcal{E}) \models (\varphi_1\ \mathcal{U}\ \varphi_2)$ *iff there exists* $i < j < |tt|$ *with* $(tt, j, \mathcal{E}) \models \varphi_2$ *and for all* $i < j' < j$ *it holds that* $(tt, j', \mathcal{E}) \models \varphi_1$;

$(tt, i, \mathcal{E}) \models \pi_1 \sim \pi_2$ *iff* $\mathcal{E}(\pi_1) \sim \mathcal{E}(\pi_2)$;

$(tt, i, \mathcal{E}) \models x.\ \varphi$ *iff* $(tt, i, \mathcal{E}[x := Time(tt[i])]) \models \varphi$.

As is proven in [13], TPTL is strictly more expressive than MTL. The property "whenever an a-event occurs, then a b-event will occur in the future and, later a c-event will occur within 3 time units" can be expressed by a TPTL formula as: $\Box\ x.\ (a \rightarrow \Diamond\ (b \wedge \Diamond\ y.\ (c \wedge y < x + 3)))$. This property cannot be expressed in MTL.

3 The Rewriting Algorithm for TPTL in Maude

Subsequently, we develop an algorithm for checking whether a finite time-trace satisfies a TPTL formula. More specifically, when checking the satisfaction relation between a finite time-trace and a TPTL formula, the formula is continuously transformed to another formula by consuming the first time-event in the time-trace. This procedure processes iteratively, until the last time-event is consumed. It will output a boolean value in $\mathbb{B} = \{true, false\}$. Our algorithm is implemented in Maude, which provides an executable environment for various logics. Here we informally describe some of Maude's features which are related to the algorithm, more details can be found in the manual [17].

3.1 Basic Rewriting Operators and Logic Connectives

In Maude, we use the functional modules following the pattern

fmod <name> is <body> emdfm.

The body of a functional module consists of a collection of declarations, of which we will use sorts (*sort* and *sorts*), subsorts (*subsort* and *subsorts*), operations (*op* and *ops*), variables (*var* and *vars*) and equations (*eq*).

We first need to define all necessary data types involved in the program, including atomic proposition (*Atom*), event (*Event*), time-event (*TimeEvent*), time-trace (*TimeTrace*) and free variable (*FreeV*). These types are defined according to their definition shown above. The following Maude program defines operators "__", "_:-_ ", "_,_" and "_ of _" for generating an event, a time-event, a time-trace and a free variable, respectively. Every operator has a priority feature, which is declared through "[prec n]" with $n \in \mathbb{N}_{\geq 0}$.

```
op __ : Atom Event -> Event [prec 23] .
op _:-_ : Event Nat -> TimeEvent [prec 23] .
op _,_ : TimeEvent TimeTrace -> TimeTrace [prec 25] .
op _ of _ : Nat Atom -> FreeV [prec 23]. //receive a Nat
   (stands for the value of the variable) and an Atom
   (stands for the name of the variable), and generate a
   FreeV as the result.
op nil : -> Event . //an emptyset is an event
```

We also define *Atom* to be a subsort of *Event*, *TimeEvent* to be a subsort of *TimeTrace*, and *FreeV* and *Nat* to be subsorts of *Atom*.

Based on the syntax and semantics of TPTL described above, we define several operators, "_{_}", "_{_}′" and "_⊨_", for checking whether a time-trace satisfies a formula. The operator "_{_}" receives a formula and an event. It yields the formula $\top/\bot$ depending on whether the event satisfies the formula or not. The operator "_{_}′" is defined on basis of "_{_}" for checking the satisfaction relation between a time-event and a formula. A time-event te satisfies a formula φ iff $\varphi\{Event(te)\}$ returns $\top$. By extending "_{_}′", the operator

"_⊨_" is defined for checking whether a time-trace satisfies a formula. This operator receives a time-trace and a formula, and generates a boolean value in $\mathbb{B}$. Given a formula φ and a time-trace (te, tt) consisting of a time-event te and its suffix tt, then (te, tt) $\models \varphi$ returns *true*/*false* iff $\varphi\{\text{te}\}'$ returns $\top/\bot$ as the result.

The calculation rules of logic connectives $\rightarrow$ (implication), $\wedge$ (and), $\vee$ (or), ++ (exclusive or), ! (negation) and $\leftrightarrow$ (equivalence) are declared as usual [16].

In our program, the comparison operators ($\leq$, $<$, $=$, $>$ and $\geq$) and the primitive operator (+) in TPTL are denoted by $\leq'$, $<'$, $='$, $>'$, $\geq'$ and $+'$ respectively, to distinguish the original definition of these operators in Maude. See $<$ as an example of comparison operators, the declaration for $<'$ is shown as follows.

```
vars R R' N N' : Nat .
vars A A' : Atom .
op _<'_ : Formula Formula -> Formula [prec 40] .
ceq R <' R' = true if R < R' .
ceq R <' R' = false if R > R' or R == R' .
ceq ( N of A ) <' R = true if N < R .
ceq ( N of A ) <' R = false if N > R or N == R .
ceq ( N of A ) <' ( N' of A' ) = true if N < N' .
ceq ( N of A ) <' ( N' of A' ) = false if N > N' or N == N' .
```

3.2 Temporal Operators and Freeze Quantifiers

In this part we describe the Maude program for temporal operators and freeze quantifiers in TPTL. Let TE be a time-event, TT be a time-trace, X and Y be formulae, and U' be an operator, which receives two formulae and generates a formula. The rewriting rules for the temporal operator $\mathcal{U}$ is presented as follows.

```
eq TE |= X U Y = false .
eq TE, TT |= X U Y = TT |= X U' Y .
eq TE, TT |= X U' Y = TE, TT |= Y or TE, TT |= X and TT |= X
   U' Y .
eq TE |= X U' Y = TE |= Y .
```

In Maude, we denote the formula $x.\ \varphi$ by $(R\ of\ x)$ @ φ with $x \in AP$ being the name of the quantifier, $R \in \mathbb{N}_{\geq 0}$ being the value of the quantifier, and φ being a TPTL formula. In addition, we define an operator "@@" for assigning free variables in φ. The rewriting process of $tt \models (R\ of\ x)$ @ φ is separated into two steps as follows.

1. The variable x of $x.\ \varphi$ is set to the time when the formula is evaluated. Hence, the formula $(R\ of\ x)$ @ φ is rewritten to another formula $((Time(tt[0])\ \text{of}\ x)$ @@ $\varphi)$, where $(Time(tt[0])$ is the initial time value from the given time-trace;
2. The operator @@ assigns all occurrences of variable x in φ to the value $(Time(tt[0])$, and proceeds with the $tt \models \varphi$ checking process. The Maude program is as follows.

```
/* the value of a freeze quantifier (R of A) equals to
   T, which is the time of the first time-event in the
   time-trace */
eq E :- T, TT |= (R of A) @ X = E :- T , TT |= ((T of A )
   @@ X) .
eq E :- T |= (R of A) @ X = E :- T |= ( T of A ) @@ X .
ceq (M of A) @@ (M' of A') = (M of A' ) if A == A' . // a
   FreeV (M' of A') is assigned to the value of the freeze
   quantifier (M of A) if they have the same name
ceq (M of A) @@ (M' of A') = (M' of A' ) if A =/= A' . //
   a FreeV (M' of A') is not assigned to the value of the
   freeze quantifier (M of A) if they have different names
/* the value assignment rule for an algebraic formula. */
ceq (N of A) @@ (N' of A' +' R) = N + R if A == A' .
ceq (N of A) @@ (N' of A' +' R) = (N' of A' +' R) if A =/=
   A' .
```

In addition, we introduce the following equivalences into the program for the operator @@. These equivalences are declared in the module FREE-QUAN, where N, N', M and M' are natural numbers; A, A', B and B' are atomic propositions; E is an event; and X, Y, $true$ and $false$ are formulae.

```
eq (N of A) @@ (X /\ Y) = (N of A) @@ X /\ (N of A) @@ Y
eq (N of A) @@ (X ++ Y) = ((N of A) @@ X) ++ ((N of A) @@ Y)
eq (N of A) @@ E = E
eq (N of A) @@ (X <' Y) = ((N of A) @@ X) <' ((N of A) @@ Y)
eq (N of A) @@ true = true . eq ( N of A ) @@ false = false
eq (N of A) @@ ((N' of A') @ X) = (N' of A') @ ((N of A) @@ X)
eq (N of A) @@ (<> X) = <> ((N of A) @@ X)
eq (N of A) @@ ([] X) = [] ((N of A) @@ X)
eq (N of A) @@ (X U Y) = ((N of A) @@ X) U ((N of A) @@ Y)
eq (N of A) @@ (o X) = o ((N of A) @@ X)
```

4 Case Study: the RBC/RBC Handover Process

In this section, we apply our TPTL runtime verification implementation to a concrete example from the European Train Control System (ETCS). ETCS is a signaling, control and train protection system that is replacing the national, incompatible safety systems within Europe. ETCS consists of the on-board subsystem (composed of ERTMS/ETCS on-board equipment, the on-board part of the GSM-R radio system and specific transmission modules for existing national train control systems), and the track-side sub-system (composed of balise, lineside electronic unit, GSM-R, radio block center (RBC), euroloop and radio infill unit) [18]. In ETCS, the RBC is responsible for providing movement authorities to allow the safe movement of trains. A movement authority is generated by computing messages to be sent to the trains, where the messages are on the

basis of information received from external track-side systems and information exchanged with the on-board sub-system. A route is divided into several RBC supervision areas. Here we consider the RBC/RBC handover specification. When a train approaches the border of an RBC supervision area, an RBC/RBC handover process takes place (see Fig. 2). The RBC/RBC handover specification specifies how a train moves from one RBC supervision area to an adjacent one.

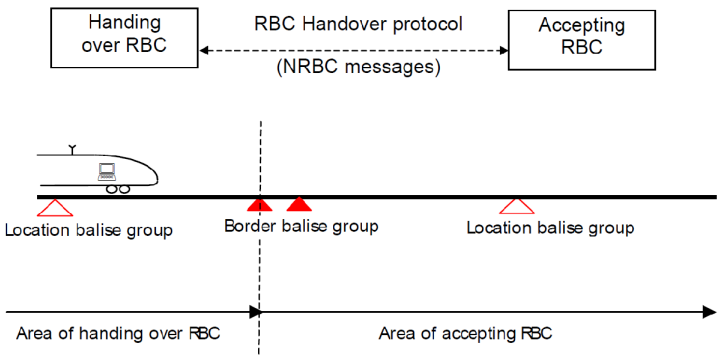


Fig. 2. The RBC/RBC handover process

We consider properties on basis of the two different specifications: FIS for the RBC/RBC Handover [19] and RBC-RBC Safe Communication Interface [20]. An execution of the system refers to the following properties in the FIS for the RBC/RBC Handover.

- Property 1: "the handing over RBC is responsible to send information about an approaching train to the accepting RBC area (i.e. pre-announcement)" (4.2.2.1);
- Property 2: "the handing over RBC must send Acknowledgment after receiving route related information" (5.2.2.5);
- Property 3: "if the Acknowledgment for route related information is missing, the accepting RBC must send route related information again" (5.2.3.5).

Based on the specification of the Safe Communication Interface, we assume that the time to take into account an incoming message and produce an answer is between 30 and 60 time units. We also assume that the tolerance window for the messages transition time is between 0 and 50 time units. Table 1 shows the abbreviations used in our case.

Let *Mess* be any message. We write "*sendMess*" for the *Mess* which is sent by a component, and "*recvMess*" for the *Mess* which is received by a component. The above properties can be expressed by the following TPTL formulae.

Table 1. Abbreviations in case study

Abbreviation	Definition
HOVcond	Handover condition detected
PreANN	Pre-announcement
RRI	Route related information
Ackn	Acknowledgment
AcknMissing	The Acknowledgement is missed
RRIReq	Route related information request
MAReq	Movement authority request
PosRep	Position report
Ann	Announcement
TOR	Taking Over Responsibility
BPSRE	Position report: "Border passed by safe rear end"
BPFE	Position report: "Border passed by max safe front end"

- Property 1: $\varphi_1 = \Box\, x.(\text{sendPreANN} \rightarrow \Diamond\, y.\ (\text{recvPreANN} \wedge (y \leq x + 50)))$.
- Property 2: $\varphi_2 = \Box\, x.(\text{recvRRI} \rightarrow \Diamond\, y.(\text{sendAckn} \wedge (y \geq x + 30) \wedge (y \leq x + 60)))$.
- Property 3: After an RRI message is sent by the accepting RBC, three time intervals must be considered: the transition time of RRI ($0 < r_1 \leq 50$), the time for producing acknowledgment ($30 \leq r_2 \leq 60$) and the transition time of the message acknowledgment ($0 < r_3 \leq 50$). Hence, if the accepting RBC does not receive the acknowledgment between 30 and 160 ($= 50 + 60 + 50$) time units after sending an RRI, an AcknMissing message should occur. The accepting RBC should resend an RRI after the AcknMissing message occurs, within 50 time units. Now property 3 can be expressed by the TPTL formula φ_3, :
  - $\varphi_{31} = \Box\ (x.(\text{sendRRI} \rightarrow \Diamond\, y.(\text{recvAckn} \wedge (y \leq x + 160) \wedge (y \geq x + 30)))$ $++\ x.(\text{sendRRI} \rightarrow \Diamond\, y.(\text{AcknMissing} \wedge (y > x + 160))))$;
  - $\varphi_{32} = \Box\, x.(\text{AcknMissing} \rightarrow \Diamond\, y.(\text{sendRRI} \wedge (y < x + 50)))$;
  - $\varphi_3 = \varphi_{31} \wedge \varphi_{32}$.

We assume that the handing over RBC and the accepting RBC have a synchronized clock, beginning at time 0. An example of RBC/RBC handover process is given in the FIS for the RBC/RBC Handover specification. Based on the RBC-RBC Safe Communication Interface specification, we design a potential time stamp for each event, get an example of real-time executions of this process, shown in Fig. 3. A corresponding time-trace is as follows.

tt_1 = (sendPreANN, 0), (sendRRIReq, 20), (recvPreANN, 35), ({sendRRIReq, recvRRIReq}, 50), (sendRRI, 90), (recvRRIReq, 97), (recvRRI, 115), (sendAckn, 157), (sendRRI, 180), (recvAckn, 191), (AcknMissing, 350), (sendRRI, 360), (recvRRI, 373), (sendAckn, 403), (recvAckn, 437), (recvMAReq, 492), (sendRRIReq, 536), (recvRRIReq, 542), (sendRRI, 583), (recvRRI, 592), (send Ackn, 639), (recvAckn, 652), (recvBPFE, 700), (sendTOR, 738), (sendAnn, 741), (recvAnn, 752), (recvTOR, 759), (recvCBPRE, 800).

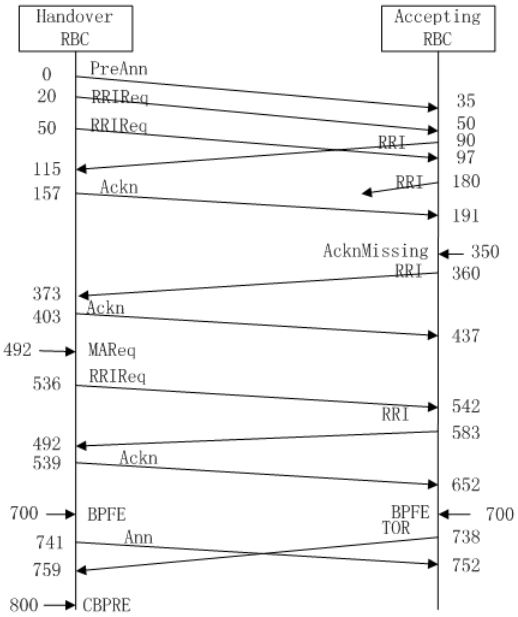


Fig. 3. An example of message sequence

The calculation results of $tt_1 \models \varphi_1$, $tt_1 \models \varphi_2$ and $tt_1 \models \varphi_3$ in Maude are all *true*. It means that this execution satisfies all the three properties.

Time-trace tt_2 represents an execution in which some errors occur: *i*) the accepting RBC receives the pre-announcement 60 time units after it is sent; *ii*) the handing over RBC does not send the acknowledgment after reception of an RRI; *iii*) when missing the acknowledgment of an RRI, the accepting RBC does not resend it.

tt_2 = (sendPreANN, 0), (sendRRIReq, 20), (recvPreANN, 60), ({sendRRIReq, recvRRIReq}, 65), (sendRRI, 90), (recvRRIReq, 97), (recvRRI, 115), (sendRRI, 180), (recvMAReq, 492), (sendRRIReq, 536), (recvRRIReq, 542), (sendRRI, 583), (recvRRI, 592), (sendAckn, 639), (recvAckn, 652), (recvBPFE, 700), (sendTOR, 738), (sendAnn, 741), (recvAnn, 752), (recvTOR, 759), (recv CBPRE, 800).

The calculation results of $tt_2 \models \varphi_1$, $tt_2 \models \varphi_2$ and $tt_2 \models \varphi_3$ are all *false*, which means that this execution of the system violates the properties.

We repeated similar experiments several times with difference traces. The checking efficiency is shown in Fig. 4. The case study shows that our TPTL based runtime verification implementation is feasible to detect failures in the executions of a system.

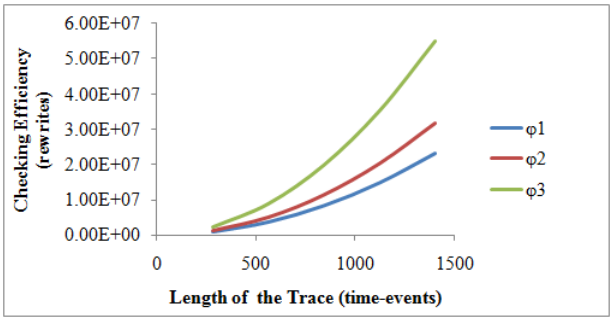


Fig. 4. The monitoring efficiency in Maude

5 Conclusion

In this paper, we have proposed a runtime verification method for TPTL. We developed a formula rewriting based algorithm, and implemented the algorithm in Maude. This makes it possible to check the satisfaction relation between a long time-trace and a complex TPTL formula automatically. Furthermore, we have presented a case study with a concrete example from the railway domain. The results show the feasibility of our implementation.

There are several interesting topics for future work. Firstly, as is well known, LTL with two truth values gives misleading results when checking finite traces. For this reason, we want to develop a three-valued TPTL, introducing a third truth value "*inconclusive*". This truth value means the satisfaction relation between a time-trace and a TPTL formula is decided by the potential suffix of the given initial fragment of the time-trace. Secondly, the clock reset principle in a TPTL formula $x.\varphi$ is to freeze the variable x in φ when the formula is evaluated. This makes TPTL unintuitive in the cases when a property contains a "clock-reset" condition. Hence an extension of TPTL with modifying the freeze quantifier "$x.$" to "$\psi.$" is worth to be studied, where ψ is any formula. Last but not least, to solve the difficulty of writing formal specifications in runtime verification, we are going to study specification techniques. The long-term goal is to develop a methodology to semi-automatically translate system specifications from the railway domain into temporal formulae.

References

1. Leucker, M., Schallhart, C.: A Brief Account of Runtime Verification. Journal of Logic and Algebraic Programming 78, 293-303 (2009)
2. Havelund, K., Roşu, G.: Monitoring Java Programs with Java PathExplorer. Electronic Notes in Theoretical Computer Science 55, 200-217 (2001)
3. Chen, F. and G. Roşu.: Mop: an efficient and generic runtime verification framework. ACM SIGPLAN Notices, pp. 569-588 (2007)
4. Barringer, H., Havelund, K., Rydeheard, D., Groce, A.: Rule Systems for Runtime Verification: A Short Tutorial. In: Runtime Verification, pp. 1-24. Springer, (2009)

5. d'Amorim, M., Roşu, G.: Efficient Monitoring of ω-languages. In: Computer Aided Verification, pp. 364-378. Springer, (2005)
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) 20, (2011)
7. Koymans, R.: Specifying Real-time Properties with Metric Temporal Logic. Real-time systems 2, 255-299 (1990)
8. Thati, P., Roşu, G.: Monitoring Algorithms for Metric Temporal Logic Specifications. Electronic Notes in Theoretical Computer Science 113, 145-162 (2005)
9. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime Monitoring of Metric First-order Temporal Properties. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 49-60. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2008)
10. Basin, D., Klaedtke, F., Zălinescu, E.: Algorithms for Monitoring Real-time Properties. In: Runtime Verification, pp. 260-275. Springer, (2011)
11. Alur, R., Henzinger, T.A.: A Really Temporal Logic. Journal of the ACM (JACM) 41, 181-203 (1994)
12. Alur, R., Henzinger, T.A.: Real-time Logics: Complexity and Expressiveness. Information and Computation 104, 35-77 (1993)
13. Bouyer, P., Chevalier, F., Markey, N.: On the Expressiveness of TPTL and MTL. Information and Computation 208, 97-116 (2010)
14. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Runtime Verification of Timed LTL Using Disjunctive Normalized Equation Systems. Electronic Notes in Theoretical Computer Science 89, 210-225 (2003)
15. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and Programming in Rewriting Logic. Theoretical Computer Science 285, 187-243 (2002)
16. Havelund, K., Rosu, G.: Monitoring Programs Using Rewriting. In: Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on, pp. 135-143. IEEE, (2001)
17. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (version 2.6). University of Illinois, Urbana-Champaign 1, 4.6 (2011)
18. UNISIG: SUBSET-026: System Requirements Specification. (2008)
19. UNISIG: SUBSET-039: FIS for the RBC/RBC Handover. (2005)
20. UNISIG: SUBSET-098: RBC-RBC Safe Communication Interface. (2007)

Sound Recoveries of Structural Workflows with Synchronization

Piotr Chrząstowski-Wachtel, Paweł Gołąb, and Bartosz Lewiński

Institute of Informatics, Warsaw University,
Banacha 2, PL 02-097 Warszawa, Poland
pch@mimuw.edu.pl, pawel.golab@mimuw.edu.pl, bartosz.lewinski@mimuw.edu.pl

Abstract. We consider communication places in workflow nets, where the connected transitions lie in parallel threads. When a workflow Petri net is constructed structurally, by means of refinements, such places cannot be modeled structurally. They are added after the net is constructed. Workflow nets constructed in a structural manner are sound, but addition of such communication places can result in losing the desired *soundness* property. However, there is a method to avoid such misplacement of communication places. We should limit the pairs of connected transitions to the ones that lie in truly parallel threads and to avoid cycles introduced by communication places.
Recovery transitions — special kind of transitions used as a powerful tool in dynamic workflow modeling — allow the manager to move tokens arbitrarily, when some unexpected situation happens. They were extensively studied and proved to be a useful tool in the workflow management [HA00]. They can be modeled as a kind of *reset transitions* with additional feature of depositing tokens taken from a specified region to particular places in this region, like it was proposed in [Ch06]. Moving tokens arbitrarily by the manager requires a lot of attention, since soundness of the net can easily be destroyed. In this paper we present a sufficient and necessary condition of soundness for a marking in a structured net with communication places. Verifying the condition turns out to be fast. The cost is linear with respect to the total number of places and transitions.

1 Introduction

Workflow management is an area, where workflow designers can prove correctness and flexibility of their models. It has been studied in numerous papers, like [WS09], [BPS09]. One of the major problems is how to organize communication between parallel tasks performed by communicating agents [BPM05]. Making communication pattern wrongly can easily lead to deadlocks, mutual waiting or leaving messages as trash, when they are deposited somewhere, and not consumed by anyone. The danger of bad design increases, when we talk about management that lasts in time and needs rearrangement due to some unexpected situations.

Composing workflow nets in a structural way was proposed in [ChBHO03]. A number of basic node refinement rules has been introduced. These rules include sequential split, parallel split, choice and loop splits. They reflect typical programming constructs like sequence of actions, an invocation of parallel threads, instruction of choice and a loop statement. The control of the workflow runs can be hence guarded by these constructs. Restricting the nets to the nets obtained from a single node by these structural constructs was proven in to guarantee soundness, as defined by [vdAtH00].

As it was already recognized in [ChBHO03], these constructs are not sufficient for typical needs of a workflow designer. In the cited paper a number of non-structural synchronization rules were proposed. By non-structural we mean here adding of new nodes and edges, which do not result as a refinement of existing nodes. Among them the most important was the synchronization of two parallel actions. When in two parallel threads A and B we want an action b from B to wait until an action a from A has been executed, we can model it by introducing a new communication place s with arcs leading from a to s and from s to b. Such construct we call a *synchronization* or *communication*, depending on whether we emphasize the fact that b must wait for a or that a has something important to communicate to b. Introducing such synchronization places can result in a possible deadlock or other unsound constructs, like trash tokens generation. In order to avoid misintroduction of such places, a criterion was proposed, which is a sufficient condition for soundness. The condition was based on the idea of the *refinement tree*, reflecting the history of refinements. It has been proven that refinement trees are unique up to an isomorphism for a given structural WF-net. In other words, if a WF-net is constructed structurally, then all the histories creating this net differ only in unimportant details (like the order of refinements in disjoint areas), resulting in the same refinement tree.

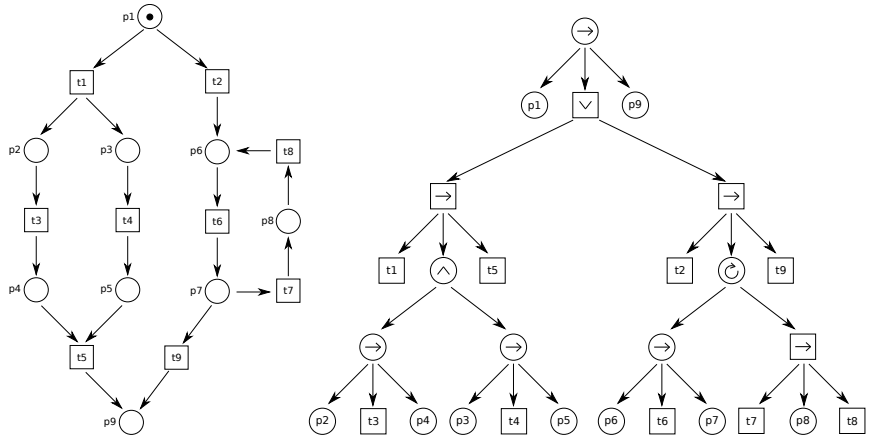


Fig. 1. Example *WF*-net and an corresponding refinement tree

In dynamical workflows it is often desired that the control is being changed during the lifetime of the workflow execution. Such situations are quite normal, especially when workflows describe processes that last for a long time (months or even years). Sometimes the manager decides to detour from the anticipated control flow and would like to "correct" the flow manually moving tokens around. Situations of this kind can happen in particular, when for instance under some time pressure we decide to skip several actions or when we decide that some part of the workflow should be repeated due to unexpected failures, which were not foreseen during the design. In such cases we would like to support the workflow management by allowing the manager to perform only sound rearrangements of the flow. When no such restriction would be set, the manager, quite possibly without understanding side effects, could make undesirable changes. This can lead to unwanted behavior, making the net unsound.

One of the main problems with such on-the-fly changes of the markings is to determine the *impact area*, which is the least part of the net, called *region*, which is affected by the rearrangement of tokens. The refinement tree gives us precise information — in order to define the impact area caused by any changes in the net, it suffices to find the latest common predecessor of the nodes, where the changes were made. The nodes which are not descendants of this node are not affected by these changes.

The changes we consider are of two kinds. First of them is the addition of places or transitions in an unstructured manner. An example of such useful addition is the introduction of a place joining two transitions, which are in (different) parallel threads. If such a place connects transition t with transition r, then the intention is to suspend the execution of r until t is executed. Clearly the introduction of such a place can result in a deadlock. In [ChBHO03] a strong sufficient condition was presented guaranteeing net soundness after such insertion of a communication place. It turns out that if an inserted place connects such two transittion-type leaves t and r in the refinement tree that no choice-split or loop-split node is found on the path from t to r and if no cycle can be detected in the net after such insertion, then the resulting net is sound.

The second kind of change is of dynamical matter. We allow the manager to modify the marking by arbitrary moving tokens around some *region*. A region is understood as the net unfolded from a single place-type node in the refinement tree. Inside a region we consider the so-called *recovery transitions*, which remove tokens from the whole area and restore them in arbitrarily chosen places. Our goal is to find conditions, which would protect the manager from depositing tokens on such places, that the resulting marking would be unsound, hence not properly terminating.

2 Refinement Rules

This section is a short reminder of *WF*-nets refinement rules introduced in [ChBHO03]. The idea behind is that having defined the refinement rules preserving certain network properties, we are able to analyse *WF*-networks that were

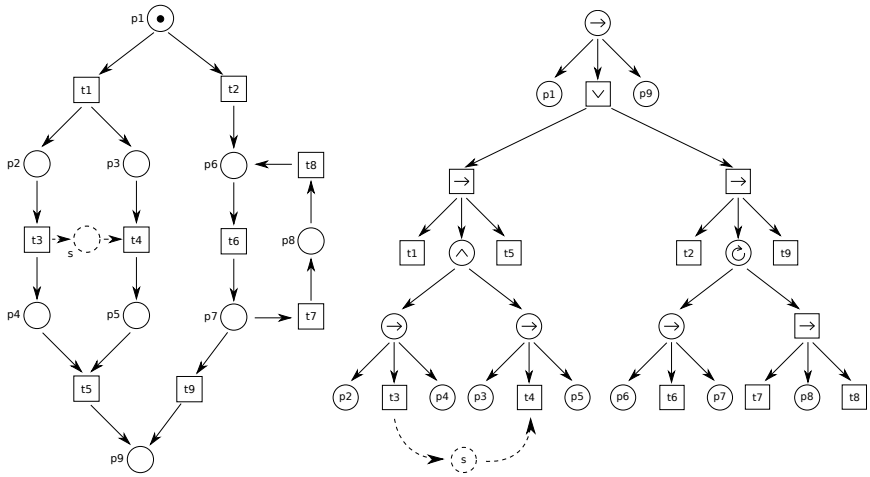


Fig. 2. Example of net synchronization with corresponding refinement tree

constructed by using those rules. To construct such network one starts with a single place, and then repeatedly applies context-free rules on network elements. One of biggest advantages of using refinement rules approach is construction trace called *refinement tree* that contains much information about the network structure. An example of such tree and corresponding network can be found on figure 1. The rest of this section covers basic rules presented in [ChBHO03]. All described rules are depicted on figure 3.

The first two rules are called *sequential splits*. They are used to create linear sequence of places and transitions, like $p2 \rightarrow t3 \rightarrow p4$ on figure 1. It's an example of splitting a single place. There are two kinds of sequential splits depending on the node type they are applied to. We call them *sequential place split* and *sequential transition split* respectively. Splits of this type introduce partial order of transitions in sound transition firing sequences. Splits replace the chosen node with three other nodes: the first and the last are of the same type and have either the same inputs or outputs as the original node, respectively. The third node is the one in the middle that connects two other nodes, so is of opposite type.

The next two rules are equivalent to logical AND and OR gates respectively. The first of them applies to places and the second one to transitions and both are replacing node with two copies of it.

The first split called *parallel split* introduces two parallel threads that will be executed simultaneously. In sound transition firing sequences transitions from different parallel paths can be safely swapped (if partial order defined by other splits is preserved). Examples of such paths are $p2 \rightarrow t3 \rightarrow p4$ and $p3 \rightarrow t4 \rightarrow p5$ on figure 1.

The second split, called *choice split*, defines two alternative paths that the process can follow. During a single process run, transitions of only one of the

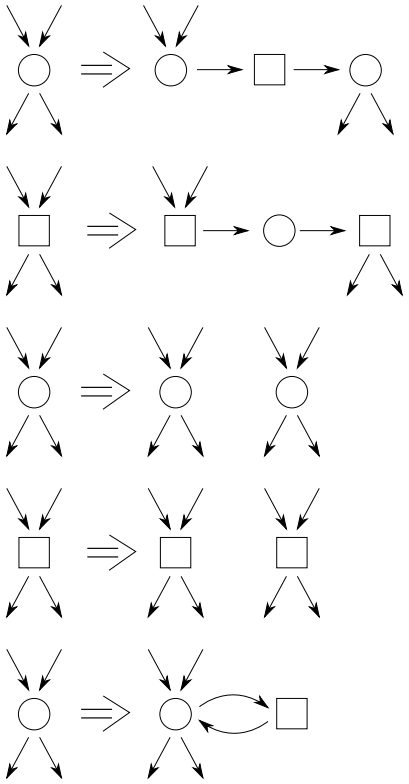


Fig. 3. Basic *WF*-nets refinement rules. Starting from top: *sequential place split*, *sequential transition split*, *parallel split*, *choice split* and *loop split*.

paths can be enabled. Examples of such paths are $t1 \to \ldots \to t5$ and $t2 \to \ldots \to t9$ on figure 1.

The last split type introduces loops and therefore is called a *loop split*. A loop example with nodes later splitted by sequential split rules can be found on figure 1.

3 Definitions

In this section we'll present some definitions and notation conventions that will be used in the rest of the article.

Siblings. For the node v that is a child of sequential split type node p, let right siblings be defined as siblings that occur after v in p's children order. The definition of left siblings of v is analogous.

Prenodes. We define *Prenodes*(v) set as follows. Let p_i be the i-th node on the path from *root* to v. Then for each p_i of sequential split type, *Prenodes*(v) includes p_i, left siblings of p_{i+1} and their subtrees.

This definition corresponds to the original refinement tree, without using additional edges provided by synchronisations.

Existential marking function. Below we present notation and definition for existential marking function which returns information if in a given set of nodes any places are marked.

$$\mathrm{M}^{?}(V) = \begin{cases} 0 & \text{if } \mathrm{M}(v) = 0 \text{ for each } v \in V \\ 1 & \text{otherwise} \end{cases}$$

Structured *SWF*-net (Synchronised WorkFlow network) is an extension of *WF*-net defined as a 7-tuple $\langle P, T, F, s, e, S, C \rangle$, where

1. $\langle P, T, F, s, e \rangle$ is a standard structured *WF*-net with the set of places P, transitions T, flow function F the source place s, and the exit place e.
2. S — the set of synchronisation places (semaphores)
3. C — the set of edges joining semaphores and synchronised transitions. It is easy to see that $C \cap F = \emptyset$

When two transitions t_1 and t_2 are synchronised via place s, we denote t_1 as $in(s)$ and t_2 as $out(s)$.

4 Soundness Characterisation

The main goal of this chapter is to find properties of marking M in *SWF*-net that guarantee soundness. Before introducing these properties we'll define two auxiliary sets *Before* and *After*, that will help us in those properties formulation and we'll explore some important properties of them.

4.1 *Before* and *After* sets

Intuitively, *Before*(v) and *After*(v) sets include places and transitions, that are over or under vertex v in *WF*-net graph respectively. In the case of *SWF*-nets we consider only such nodes that are not synchronisation places during *Before* and *After* sets construction.

We define *Before*(v) set as follows. A leaf l is in *Before*(v) if and only if there exists a predecessor q of l being a left-sibling of some node lying on the path from the root to v inclusively. Similarly for $After(v)$ we consider right siblings instead. We ignore the synchronisation places.

An important feature of *Before* and *After* sets is that *Before*(v) and *After*(v) sets are not containing v itself, so immediate conclusion from *Before* and *After* sets definitions is that *Before*$(v) \cap$ *After*$(v) = \emptyset$ for each node v.

It is worth to explore, how *Before* and *After* sets are constructed in the case of loops. We can distinguish two cases depending on whether loop contains the node for which these sets are constructed or not.

Let us consider loops from the first case. Let l be the loop that was created by splitting place p_l and transition t_l. In this case *Before* and *After* sets will either contain leaves only from p_l subtree or only from t_l subtree.

In the second case the *Before* and *After* sets can either contain all the leaves of the given loop or none of its nodes.

In the forthcoming text it is important to have clarity about *Before* and *After* sets containment rules. Let v, v_b, v_a be the vertices such that $v_b \in \mathit{Before}(v)$ and $v_a \in \mathit{After}(v)$. Clearly, $\mathit{Before}(v_b) \subset \mathit{Before}(v)$ and $\mathit{After}(v_a) \subset \mathit{After}(v)$. We also have $v \in \mathit{After}(v_b)$ and $v \in \mathit{Before}(v_a)$. And so, finally, $\mathit{After}(v) \subset \mathit{After}(v_b)$ and $\mathit{Before}(v) \subset \mathit{Before}(v_a)$.

The *Before* and *After* sets have some interesting properties in the context of sound marking, that we will formulate in the following proposition. We say that a node (place or transition) x in a Petri net is reachable from a given marking M if it is not dead in the case x is a transition or it can be marked by some marking reachable from M, in the case x is a place.

Proposition 1 *Let M be a sound marking of WF-net with synchronisations. For each place or transition x we have:*

1. $M(x) \geq 1$ *implies* $M(\mathit{After}(x)) = 0$
2. $M(\mathit{Before}(x)) > 0$ *implies* $M(\mathit{After}(x)) = 0$
3. $M(\mathit{Before}(x)) > 0$ *implies* $M(x) = 0$ *and x is reachable.*

Proof. Let $\langle P, T, F, s, e, S, C\rangle$ be a structured SWF-net and M a sound marking on this network.

We begin the proof with some observations. When constructing the *Before* and *After* sets, we take into account only subtrees related with nodes that are of sequential split type. The sequential split type nodes determine the partial order of transitions in possible transition sequences transforming any sound marking M (in particular M_s^1) into M_e^1. Some transitions are incomparable in this order because of different types of nodes, for example AND nodes that introduce concurrency in nets. This partial order results in important properties of *Before* and *After* sets.

For a transition $t \in T$ that is not a part of any loop, the set $\mathit{After}(t)$ contains all transitions that can fire after t occurs in a sequence transforming a sound marking M into M_e^1 and that this firing is directly dependent on t. If the transition t is in a loop, the same condition holds, except for some other transitions from this loop — not all of them are included in $\mathit{After}(t)$ set. But still, all transitions from the loop that belong to the $\mathit{After}(t)$ set in order to fire, need transition t to fire before them.

It is important to stress out here, that we only consider transition sequences that contain t when writing about firing dependences. In the case of transitions that come after AND nodes, there are at least two independent paths which can lead to those transitions firing, so there are possible situations, when t won't

occur in such sequences but the considered transitions will still fire. Nevertheless, if t occurs in such a sequence, it determines the path that the process went through and we know that t firing is necessary in order to make the next transitions firings possible. We have a similar situation in the case when t results from loop main transition fragmentation (as $t7$ and $t8$ on figure 1).

With this observation, we can move to the main part of the proof.

1. The first step is a direct result of our observation. If $x = e$ then this point is obvious. If $x \neq e$ then $\mathrm{M}(x) \geq 1$ means that x is a place and x is an input for some transitions $T_x \subset T$.

Firstly, let's consider nets that contain no loops. Each transition in any sequence transforming M_s^1 into M can occur only once. Since x isn't empty, we know that none of the T_x transitions will occur in possible transition sequences transforming M_s^1 into M. Taking into account our observation this also means that none of the transitions from *After* sets of T_x transitions will occur in such sequences. It means that none of outputs of T_x transitions or outputs of transitions from their *After* sets is marked. We also know, that all the transitions from T_x, all their outputs and their *After* sets are in $After(x)$. Moreover it is easy to find that these are the only items in $After(x)$. So we have $\mathrm{M}(After(x)) = 0$.

The case of loops is very similar. The only problem is that x can be a loop element. It is possible in this case, that not all of T_x transitions will be in $After(x)$ set — some of them can be transitions starting new loop iteration. However, it is easy to recognize that this makes no problem, and reasoning for transitions from T_x that are in $After(x)$ is still valid.

2. This property is a simple consequence of 1. We know that for all places from $Before(x)$ that marking M is greater than zero and their *After* sets markings is zero too. We also know that $After(x)$ set is a subset of those *After* sets, so $\mathrm{M}(After(n)) = 0$.

3. First part of this property is also a consequence of point 1. We know, that for all places from $Before(x)$ for which the marking M is greater than zero, their *After* sets markings equal zero. We also know that x is an element of those places *After* sets, so $\mathrm{M}(x) = 0$.

The fact that x is reachable is a result of our observation. Let t_y be such a transition that is in *After* set of some place from $Before(x)$ that the marking M is greater than zero, and for which this place in an input. We know that x is in t_y's *After* set, so there exists a transition t_x for which x is an output and either $t_x \in After(t_y)$ or simply $t_x = t_y$. Because marking M is sound, it is possible for t_y to fire. In the case when $t_x \neq t_y$, we know from the observation and from the fact that the marking M is sound [ChBHO03], that it is also possible for t_x to fire, so x is reachable.

4.2 Properties of sound *SWF*-net markings

In this section we'll give a characterisation of *SWF*-net sound markings in the form of a short lemma. Before formulating the lemma, it is worth noticing, that as in the case of standard *WF*-nets, if M is a sound marking of *WF*-net with synchronisations, then M is 1-bounded (there can't be two tokens at a one place).

Lemma 1 *Let* $\mathcal{N} = \langle P, T, F, f, e\rangle$ *be an* WF*-net and* $\mathcal{N}_s = \langle P, T, F, f, e, \{s\}, C\rangle$ *be the same net with added synchronisation place* s*. Given* $\mathcal{N}_s$ *marking* M *is sound if and only if:*

1. $M|_{\mathcal{N}}$ *is sound in* $\mathcal{N}$
2. *Exactly one of following holds:*
   (a) $M(Before(in(s))) > 0$
   (b) s *is marked*
   (c) $M(After(out(s))) > 0$
   (d) *Synchronisation was skipped, so* $M(Before(in(s)) \cup After(in(s))) = 0$ *and* $M(Before(out(s)) \cup After(out(s))) = 0$

Basically, the lemma describes intuitions about how synchronisation place should work. It says, that we have four different states, that our process can be in. First three cases are straightforward: it can be either before synchronisation, during synchronisation or after synchronisation. The last one is the case, when the synchronisation occurs in one of the branches that resulted from an choice split, and an active branch is not the synchronised one. An example of such situation is depicted below on figure 4.

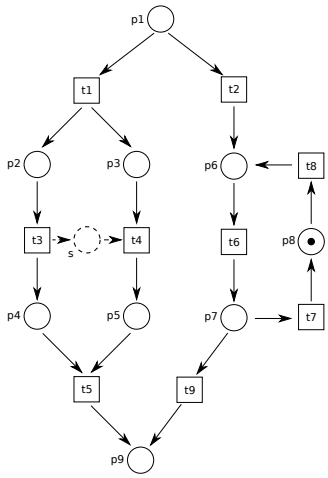


Fig. 4. Example of synchronisation skipping marking

Therefore, for the sake of precision, we can describe the first three states as: synchronisation place is active, process is during synchronisation and synchronisation is inactive, but either synchronisation has occurred, or the process branch without synchronisation is at the evaluation point, when marked places are in $After(out(s))$ set. The last one describes situation when process finished considered fragment with alternative). Nevertheless it is more convenient to use definitions presented in the previous paragraph.

To show that the above intuitions are right, we will now prove our lemma.

Proof. It is important to remember that synchronization of items in loops is not allowed, so in every valid transition sequence $in(s)$ and $out(s)$ can occur only once.

Firstly let us assume that a marking M is sound in $\mathcal{N}_s = \langle P, T, F, f, e, \{s\}, C \rangle$ net.

What synchronisation point does, is prohibiting certain transition sequences that allow obtaining marking $\mathrm{M}|_{\mathcal{N}}$ from M_s^1 in $\mathcal{N}$, by enforcing firings of one transition group before another. It is important that synchronisation is not adding any new possible transition sequences. This means, that any transition sequence that leads to the marking M in $\mathcal{N}_s$ is also valid in $\mathcal{N}$, leading to the marking $\mathrm{M}|_{\mathcal{N}}$. This means that $\mathrm{M}|_{\mathcal{N}}$ is sound in $\mathcal{N}$, so 1. holds.

For the case when there exists such OR place that allows to skip synchronisation, let p_{in} be the place before such fork, and p_{out} first place after it. From the rules of synchronization we know that in this case p_{in} is in $\mathit{Before}(in(s))$ and p_{out} is in $\mathit{After}(out(s))$.

It is easy to see, that each of the conditions a), b) and c) exclude d) and vice versa. Now we'll show that a), b) and c) exclude each other. If $\mathrm{M}(\mathit{Before}(in(s))) > 0$, then we know from the proven proposition, that there is no transition sequence that leads to a marking M containing $in(s)$, so s will be empty for each valid subsequence of those transitions, which means that there was no possibility for $out(s)$ to fire, so again from the proposition, $\mathrm{M}(\mathit{After}(out(s))) = 0$.

If s is marked, than $in(s)$ has already fired. We know that $\mathrm{M}|_{\mathcal{N}}$ is valid so, $\mathit{Before}(in(s))$ marking must be empty. Again from the proposition we have $\mathrm{M}(\mathit{After}(out(s))) = 0$, because there was no possibility for $out(s)$ to fire.

If $\mathrm{M}(\mathit{After}(out(s))) > 0$ then it means that transitions $in(s)$ and $out(s)$ have already fired, so s is empty and from the validity of $\mathrm{M}|_{\mathcal{N}}$, either $\mathrm{M}(\mathit{Before}(in(s)))$ is empty or the process was run following a path where no synchronisation occurs. In the second case we have $\mathrm{M}(p_{out} \cup \mathit{After}(p_{out})) > 0$, we have that $\mathrm{M}(s) = 0$ and $\mathrm{M}(\mathit{Before}(in(s))) = 0$, because $\mathrm{M}(p_{in} \cup \mathit{Before}(p_{in})) = 0$ according to the proposition.

If none of the conditions a), b) or c) holds, we have $\mathrm{M}(\mathit{Before}(in(s))) = 0$, $\mathrm{M}(s) = 0$ and $\mathrm{M}(\mathit{After}(out(s))) = 0$. There are two possibilities. Firstly, we may consider situation when $\mathrm{M}(\mathit{After}(in(s))) > 0$ and $\mathrm{M}(\mathit{Before}(out(s))) > 0$, but in this case s can't be empty, because it would make impossible for $out(s)$ to fire and we know from the proposition that it has to fire in order to clean up tokens from $\mathit{Before}(out(s))$.

The only valid option is $\mathrm{M}(\mathit{After}(in(s))) = 0$ and $\mathrm{M}(\mathit{Before}(out(s))) = 0$, which means that all tokens are in $\mathit{After}(p_{in}) \cap \mathit{Before}(p_{out})$ on process path disjoint with the synchronisation. There are no other options because of the synchronisation rules, which allow to synchronize only these transitions, which are not separated by choice or loop split nodes in the refinement tree. So we know now that a), b), c) and d) exclude each other and in each case one of them must hold, so soundness of $\mathcal{N}_s$ means that 1. and 2. hold.

Now lets assume that 1. and 2. from the lemma hold.

$\mathrm{M}|_{\mathcal{N}}$ is sound, so let $\langle x_i \rangle$ be a sequence of events leading to M_e^1 in $\mathcal{N}$. It's easy to see, that in case when one of b), c) or d) holds, the same sequence can be fired in $\mathcal{N}_s$. In the case when s is not empty, this sequence will clean s. A bit harder is the case when a) holds, but one can easily see, that it is possible to rearrange $\langle x_i \rangle$ so that $in(s)$ will occur before $out(s)$ and still all transitions from $\mathit{Before}(in(s))$ will be before $in(s)$ and all transitions from $\mathit{After}(out(s))$ will be after $out(s)$. Rules of synchronisation guarantee that such operations will lead to valid transition sequence, which will also be valid in $\mathcal{N}_s$ and will lead to the same result as the original sequence.

This proves that if 1. and 2. from the lemma hold for a given marking M that the marking is sound.

We have proved implications in both directions, so the lemma is valid.

5 Soundness Checking

The algorithm for checking the soundness of a marking will be a modification of the soundness checking algorithm (theorem) from presented in previous work [Ch06].

As a reminder, two functions w and W were defined as follows:

$$w(v) = \begin{cases} 1 & \text{if } v = \mathit{root} \\ w(\mathit{parent}(v)) & \text{if } v \text{ is a child of a node which is not a parallel split node} \\ \frac{w'(\mathit{parent}(v))}{c} & \text{if } v \text{ is a child of a node of the parallel split node, with } c \text{ children} \end{cases}$$

$$W(v) = \begin{cases} \mathrm{M}(v)\, w'(v) & \text{for each place-leaf } v \\ \sum_{y \in Ch(v)} W(y) & \text{for all internal nodes } x \\ 0 & \text{for each transition-leaf } t \end{cases}$$

The theorem stated that a marking is sound if and only if for each tree node x either $W^s(w)(x) = 0$ or $W^s(w)(x) = w(s)$. We will now define new functions w^s, W^s and S such that W^s and S will have the same signature as W and w^s will have the same signature as w. Note that the values in W and S are determined only by leaves — for the internal nodes we take sum of the values of their children. For both $in(s)$ and $out(s)$, we will add additional weight to the path from the node to the root, and propagate this change downwards for the pre-nodes (nodes occurring to the left of it). Let c_s be a constant such that for every node x the condition $w(x) > c_s$ is satisfied.

We will apply the following transformation two times to W presented in [Ch06] for both $in(s)$, $out(s)$, with c equal to c_s, $-c_s$ respectively.

For every node v on the path from t to root we add the weight c

$$w'(v) = w(v) + c$$

For every node $v \in Prenode(in(s))$ (nodes on the left of the path from $in(s)$ to root)

$$w^s(v) = \begin{cases} 1 & \text{if } v = root \\ w^s(parent(v)) & \text{if } v \text{ is a child of a node which is not a parallel split node} \\ \frac{w^s(parent(v))}{c} & \text{if } v \text{ is a child of a node being a parallel split node, with } c \text{ children} \end{cases}$$

For the remaining nodes $w^s(v) = w(v)$. The function W remains unchanged, except for the fact that now it depends on w^s.

$$S(M)(t) = \begin{cases} c_s \cdot (\mathrm{M}(s) + \mathrm{M}^?(After(out(s)))) & \text{if } t = in(s) \\ -c_s \cdot (\mathrm{M}^?(After(out(s)))) & \text{if } t = out(s) \\ 0 & \text{for every other case} \end{cases}$$

We will use this transformation two times to nodes $in(s)$, $out(s)$ with c being inverses of each other. Notice that this pair of transformations together changes nodes only inside the smallest subtree containing them.

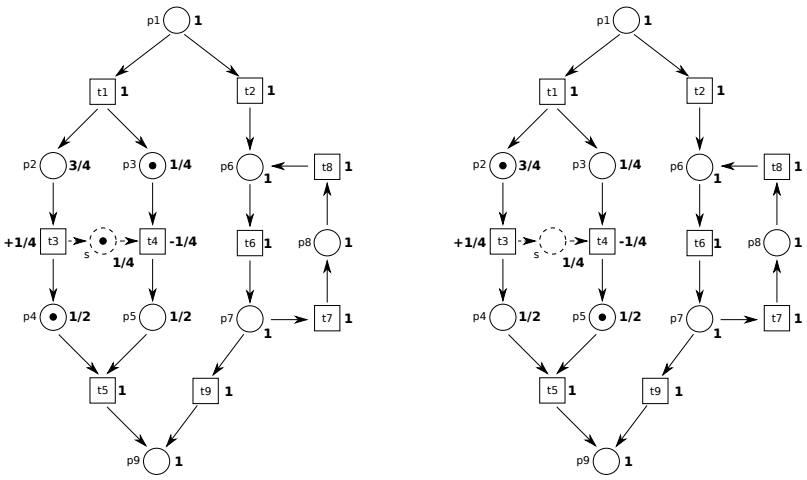


Fig. 5. Example of *SWF*-net with marked weights defined by functions from the theorem. The marking on the left is sound, while the marking on the right isn't, which results in weight explosion in $t1$.

Theorem 1 *A marking M of the* SWF*-net $\mathcal{N}$ is sound if and only if for each node v in the refinement tree either $W^s_M(v) + S(v) = w(v)$ or $W^s_M(v) = 0$*

Proof. We will prove it by using the sound characterisation lemma.

Let M be a sound marking. By the above lemma, exactly one of following holds:

1. $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) > 0$
2. s is marked.
3. $\mathrm{M}\,(\mathit{After}\,(\mathit{out}(s))) > 0$
4. Synchronisation was skipped, so $\mathrm{M}\,(\mathit{After}\,(\mathit{in}(s)) \cup \mathit{Before}\,(\mathit{in}(s))) = 0$ and $\mathrm{M}\,(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}\,(\mathit{out}(s)))$

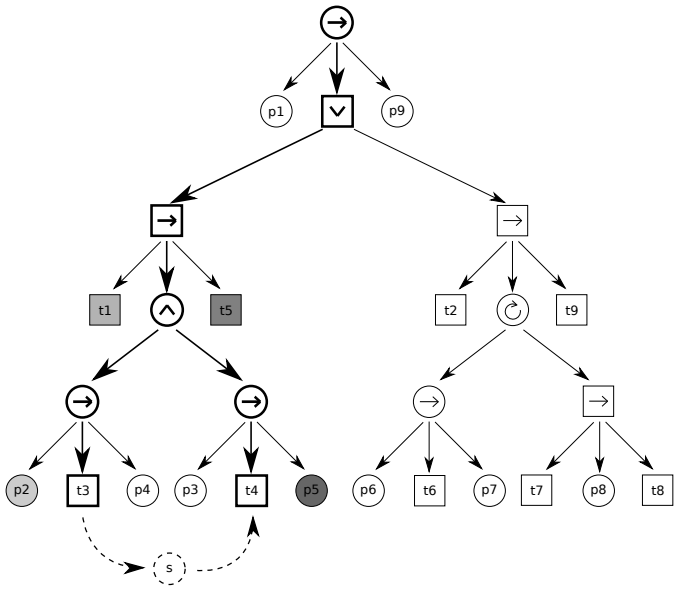


Fig. 6. Derivation Tree with paths highlighted

Lets reason case by case, we will prove that in each of these cases our algorithm will correctly verify the marking:

1. If $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) > 0$: Both $\mathit{in}(s)$ and $\mathit{out}(s)$ are inactive (which means $S(\mathit{in}(s)) = S(\mathit{out}(s)) = 0$). The marking M is sound, so $M(\mathit{After}\,(\mathit{out}(s))) = 0$ and $M(\mathit{After}\,(\mathit{in}(s))) = 0$. This means that all the marked nodes are in modified left sides ("Before") of trees, so the verification behaves just as in the W without synchronisation, because all the weights were just decreased by some amount.

2. If synchronization place s is marked: $\mathit{in}(s)$ is active and $\mathit{out}(s)$ is inactive. Since $\mathrm{M}(\mathit{After}\,(\mathit{in}(s)) \cup \mathit{Before}\,(\mathit{in}(s))) > 0$ and $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) = 0$, we have $\mathrm{M}\,(\mathit{After}\,(\mathit{in}(s))) > 0$ and because the marking is sound, it meets $S(\mathit{in}(s))$ in some node in the path to root, and then $S(\mathit{in})$ modifies it to the correct amount

3. If $\mathrm{M}(\mathit{After}(\mathit{out}(s))) > 0$: both $\mathit{in}(s)$ and $\mathit{out}(s)$ are active. Of course in this situation $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) > 0$ and $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) = 0$, so we have $\mathrm{M}(\mathit{After}(\mathit{in}(s)))) > 0$ The marking is sound, so it meets the $S(\mathit{in}(s)$ in some node in the path to root, and then $S(\mathit{in}(s))$ modifies it to the correct amount. Similarly, since $\mathrm{M}(\mathit{After}(\mathit{out}(s))) > 0$ and marking is sound, the verification succeeds until it meets at the path from in to root and here it is corrected by S from $\mathit{in}(s)$

4. If synchronisation was skipped, so $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) = 0$ and $\mathrm{M}(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}(\mathit{out}(s)))$: The modifications to weights we made apply only to nodes in the path from $\mathit{in}(s)$ to $\mathit{out}(s)$ and they are all empty in this case.

This proves that in every case, our verification succeeds.

Let us prove the opposite implication now. Let M be such a marking that for each node v in the refinement tree either $W^s_M(v) + S(v) = w(v)$ or $W^s_M(v) = 0$. We will show that the cases from the lemma are exclusive:

1. excludes (2. or 3.) Let's assume $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) > 0$. Then if 2. or 3. then $\mathit{in}(s)$ is active, but since $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) > 0$, there is an active node that is on the left hand side from the path from $\mathit{in}(s)$ to root, but it was already modified by c, so when S will meets, there will be c added two times.

2. and 3. Easy case, $W(\mathit{in}(s)) = w^s(\mathit{in}(s)) \cdot \left(\mathrm{M}(s) + \mathrm{M}^?(\mathit{After}(\mathit{out}(s)))\right) = 2w^s(\mathit{in}(s)) > w^s(\mathit{in}(s))$

It is easy to see that 4. is disjoint from the other cases too. We need only to distinguish 4. from 2.: If $\mathrm{M}(s) > 0$ then $W(\mathit{in}(s)) > 0$, and since 4. holds, $W(\mathit{out}(s)) = 0$, so in the place where paths from $\mathit{in}(s)$ to root and $\mathit{out}(s)$ to root meet there will be an unbalance.

Hence the cases from the lemma are disjoint. We need to prove now that at least one of them holds. Let's assume none of them holds. Then $\mathit{in}(s)$ and $\mathit{out}(s)$ are not active. Assume that $\mathrm{M}(\mathit{Before}(\mathit{out}(s)))$ isn't empty. Because of 1., $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) = 0$, so there are nodes enabled to the right from the path from $\mathit{out}(s)$ to the root node, that are below least common ancestor of $\mathit{in}(s)$ and $\mathit{out}(s)$. When the weight of those active nodes gets passed to the path, they need S to have $W^s_M(v) + S(v) = w(v)$, but $\mathit{in}(s)$ is inactive, so we came to a contradiction.

Similarly, assume $\mathrm{M}(\mathit{After}(\mathit{in}(s))) > 0$. Because of 1., $\mathrm{M}(\mathit{After}(\mathit{out}(s))) = 0$, so there are enabled nodes to the left from path from $\mathit{out}(s)$ to the root node, that are below least common ancestor of $\mathit{in}(s)$ and $\mathit{out}(s)$. When the weight of those enabled nodes get passed to the path, they need S to have $W^s_M(v) + S(v) = w(v)$, but $\mathit{in}(s)$ is disabled, so we came to a contradiction.

These two arguments imply that 4., $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) = 0$ and $\mathrm{M}(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}(\mathit{out}(s)))$, so at least one of the cases from the lemma holds.

Now we need to prove the first part of the lemma, that $\mathrm{M}|_{\mathcal{N}}$ is sound in $\mathcal{N}$. The proof of this fact is identical to first part of this proof. We show that if one of 1., 2. or 3. occurs, the rest of checking behaves identical to checking done in w.

6 Conclusion

We have proven that the important construction of creating a channel between two transitions (like links in BPEL4WS) can be done in a semi-structured manner with the preservation of soundness. We have discovered a condition that is sufficient and necessary for a marking to preserve soundness. The condition, based on the structure of the refinement tree is fast to verify; in fact it is linear with respect to the number of nodes of the net (so even better than the size of the net: the edges, with possible quadratic number of them, do not count). This condition allows us to determine soundness of an arbitrary marking and allow on-the fly changes of markings during the execution of a workflow. Such changes are considered a powerful tool for a manager to change a marking in an arbitrary manner in case of an unexpected detour from the normal workflow run. Support by automatic verification, if such changes can cause an undesired behavior (like deadlock or creation of trash tokens) is an important improvement of the technology.

References

[vdAtH00] W.M.P.van der Aalst, A.H.M. ter Hofstede, *Verification of Workflow Task Structures: A Petri-net-based Approach*, Information Systems, 25(1): 43-69, March 2000.

[ChBHO03] Piotr Chrząstowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, Adi Susanto, *Top-down Petri Net Based Approach to Dynamic Workflow Modelling*, Lecture Note in Computer Science. v2678. 336-353., 2003.

[Ch06] P.Chrząstowski-Wachtel, *Determining Sound Markings in Structured Nets*, Fundamenta Informaticae, 72, 2006.

[BPM05] M. Laugna, J. Marklund. *Business Process Modeling, Simulation, and Design.* Prentice Hall, Upper Saddle River, New Jersey, 2005.

[HA00] C.Hagen, G.Alonso, *Exception Handling in Workflow Management Systems*, IEEE Trans. of Soft. Eng. vol. 26 No 10, Oct 2000.

[BPS09] W.M.P. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. *Business Process Simulation: How to get it right?* In J. vom Brocke and M. Rosemann, editors, International Handbook on Business Process Management, Springer-Verlag, Berlin, 2009.

[WS09] A. Rozinat, M. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge., *Workflow Simulation for Operational Decision Support.*, Data and Knowledge Engineering, 68(9):834-850, 2009.

Floating Channels between Communicating Nets

Ludwik Czaja[1,2]

[1] Academy of Finance and Business Vistula, Warsaw
[2] Institute of Informatics, The University of Warsaw, Poland
lczaja@mimuw.edu.pl

Abstract. A network system is given as a set of Petri net-like structures called agents. Each agent has a singled out place interpreted as a communication port with ingoing edges labelled with $send(p_1, ..., p_n)$ and $receive(q_1, ..., q_m)$ commands, where p_i, q_j are names of ports of its interlocutors. Every such edge exits a transition emiting a request for send or receive message. A transmission channel between the agent and its intelocutors is established when its port holds a send or receive command, while ports of its interlocutors hold respective (matching) communication commands. This gives rise to communication between the agent and its interlocutors, after which the channel is disrupted: hence *floating channels*. Some behavioural properties of such network system are examined, their decision complexity, deadlock and fairness in their number.

1 Introduction

A system of communicating agents here is a collection of Petri net-like structures [Rei 1985], such that in every net there is a singled out place serving for communication and called a port. Each arrow entering the agent's port is labelled with a *send* or *receive* communication statement with parameters being names of ports the agent sends a message to, or receives from. Firing a transition the arrow outgoes, results in putting the arrow's label in the port. If it is a send (receive) statement and all ports - its parameters - hold matching receive (send) statements, then a communication channel between senders and receivers is set up ("matching" in the sense of "hand-shaking" [Hoa 1978], [Hoa 1985], [OCCAM 1984]). The channel is realized as a special transition, called a *transmission*, with sending and receiving ports as its preset and postset respectively. Firing such transition represents a message transfer between the ports involved. After firing, this transition disappears, thus the channel is disrupted. That is why we say that the channels are floating. Such systems are defined, examples shown and some behavioural properties investigated. If the 1-safe Petri nets are taken as the agents, then complexity of a number of decision problems for systems with floating channels become straightforward conclusions from known results, collected e.g. in [ESP 1998]. Some problems, namely deadlock and two kinds of fairness is analysed in the framework of the proposed model and their set-theoretic characteristics are given.

2 Communicating Agents

Let $A = \{A_{p_1}, A_{p_2}, ..., A_{p_d}\}$ be a set of agents, each agent A_{p_i} $(i = 1, 2, ..., d)$ equipped with a single communication port p_i, their set $P = \{p_1, p_2, ..., p_d\}$. A is treated as a distributed system whose agents are capable of intercommunicating through their ports. Suppose that the agents are autonomous, i.e. do not share any of their constituents. Let $!(p_{k_1}, p_{k_2}, ...)$ and $?(p_{l_1}, p_{l_2}, ...)$ be a shorter notation of $send(p_{k_1}, p_{k_2}, ...)$ and $receive(p_{l_1}, p_{l_2}, ...)$ operations respectively, i.e. sending a message by an agent to ports $p_{k_1}, p_{k_2}, ...$ and receiving a message from ports $p_{l_1}, p_{l_2}, ...$ Here $k_1, k_2...$ and $l_1, l_2...$ are subsequences of the sequence $1, 2, ..., d$. These *communication operations* may assume a varying number of parameters and are executed in the synchronous, i.e. hand-shaking mode. Let C denote a set of all possible communication operations of all the agents, along with the empty (no communication) operation Θ. Since apart from communication, other computational activity of the agents is inessential, such fragments of their activity are not taken into consideration. That is why we assume that agent A_p with port p is represented as a single place net with a specific firing rule (semantics):

$A_p = (\{p\}, T_p, F_p)$ for $p \in P$ where:

T_p is a set of transitions, i.e. actions inserting send or receive operations in the port p,

$F_p : T_p \times \{p\} \to C_p$ is a set of arrows from transitions to place p, each arrow labelled with a send or receive operation the agent A_p can issue, i.e. $C_p \subseteq C - \{\Theta\}$. Suppose no agent can send/receive message to/from itself. That is:

$F_p(t, p)$ is either $!(p_{k_1}, p_{k_2}, ...)$ or $?(p_{l_1}, p_{l_2}, ...)$ with $p_{k_i} \neq p \neq p_{l_j}$ $(i = 1, 2, ...;$ $j = 1, 2, ...)$.

The *local communication state* (for short: a *local state*) of the agent A_p is a function $M_p : \{p\} \to C_p \cup \{\Theta\}$.

The set of all states of the agent A_p is $\mathbb{S}_p = (C_p \cup \{\Theta\})^{\{p\}}$

Semantics of transition $t \in T_p$ is a relation $[[t]] \subseteq \mathbb{S}_p \times \mathbb{S}_p$ defined by $(M_p, M_p^{'}) \in [[t]]$ iff $M_p(p) = \Theta \wedge M_p^{'}(p) = F_p(t, p)$ ($M_p^{'}$ is the next state following M_p obtained in effect of firing transition t)

Semantics of agent A_p : $[[A_p]] = \bigcup_{t \in T_p} [[t]]$

Fig.1 depicts agent A_p capable of communicating with agents $A_{p_1}, A_{p_2}, A_{p_3}, A_{p_4}$, A_{p_5} and passing from the state $M_p = \{(p, \Theta)\}$ to the state $M_p' = \{(p, !(p_1, p_4)\}$

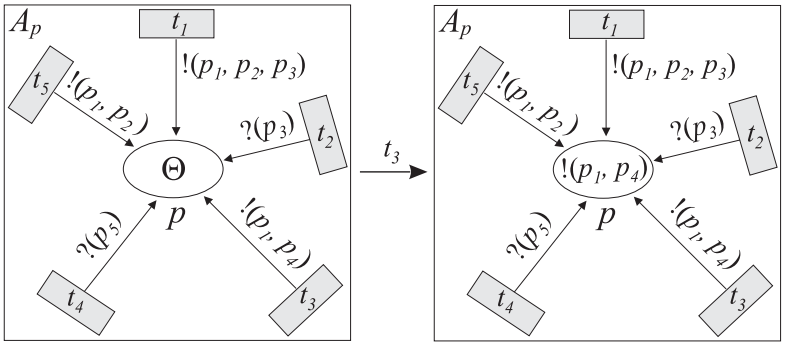


Fig. 1. Example of agent A_p and result of firing transition t_3

as a result of firing transition t_3. This means that A_p issued a request for sending a message to A_{p_1} and A_{p_4}.
The *global communication state* (for short: a *global state*) of the system A is a function
$M : P \to C$, their set $\mathbb{S} = C^P$, thus the local state M_p is a restriction of M to $\{p\}$. The state (global and local) will be treated as a set of pairs of the form $(p, !(p_{k_1}, p_{k_2}, ...))$ and $(p, ?(p_{l_1}, p_{l_2}, ...))$ for $p, p_{k_1}, p_{k_2}, p_{l_1}, p_{l_2} ... \in P$.

2.1 Transmissions

For $n, m \geq 1$, let $a_1, ..., a_n$ and $b_1, ..., b_m$, pairwise distinct, be ports of agents $A_{a_1}, ..., A_{a_n}$ and $A_{b_1}, ..., A_{b_m}$. Let $a_i{:}!(b_1, ..., b_m)$ denote the pair $(a_i, !(b_1, ..., b_m))$ meaning "agent A_{a_i} sends a message to agents $A_{b_1}, ..., A_{b_m}$" and $b_j{:}?(a_1, ..., a_n)$ the pair $(b_j, ?(a_1, ..., a_n))$ meaning "agent A_{b_j} receives a message from agents $A_{a_1}, ..., A_{a_n}$". A *transmission* (matching send and receive operations) is a pair $\boldsymbol{t} = ({}^\bullet\boldsymbol{t}, \boldsymbol{t}^\bullet)$ of sets of the form:
${}^\bullet\boldsymbol{t} = \{a_1{:}!(b_1, ..., b_m), ..., a_n{:}!(b_1, ..., b_m)\}$ (pre-set of transmission $\boldsymbol{t}$)
$\boldsymbol{t}^\bullet = \{b_1{:}?(a_1, ..., a_n), ..., b_m{:}?(a_1, ..., a_n)\}$ (post-set of transmission $\boldsymbol{t}$)
Let ${}^\bullet\boldsymbol{t}^\bullet = {}^\bullet\boldsymbol{t} \cup \boldsymbol{t}^\bullet$ and ${}^\bullet\boldsymbol{t}^\bullet \downarrow P$ be a projection of ${}^\bullet\boldsymbol{t}^\bullet$ onto the set P, i.e. ${}^\bullet\boldsymbol{t}^\bullet \downarrow P = \{a_1, ..., a_n, b_1, ..., b_m\}$, that is, the set of ports the transmission $\boldsymbol{t}$ is involved in. Note that ${}^\bullet\boldsymbol{t}^\bullet$ is of the same type as the global state M: both are sets of pairs of the form $(x, !(...))$ or $(x, ?(...))$.

Expressions $a_i{:}!(b_1, ..., b_m)$ and $b_j{:}?(a_1, ..., a_n)$ denote *matching labelled communication operations.*

Note that a transmission depends on a state: it may come into existence in a certain global state and disapear in another. Such emerging and disappearing during system's activity transmissions are typed in bold letters, to distinguish them from the static transitions of the agents.

Let $\boldsymbol{TR}$ denote the set of all possible transmissions in the system. If a transmission $\boldsymbol{t} \in \boldsymbol{TR}$ exists in a state M (i.e. ${}^\bullet\boldsymbol{t}^\bullet \subseteq M$) then its semantics is a relation $[[\boldsymbol{t}]] \subseteq \mathbb{S}\times\mathbb{S}$ defined by $(M, M') \in [[\boldsymbol{t}]]$ iff $M' = M - {}^\bullet\boldsymbol{t}^\bullet \cup \{(x, \Theta) |\ x \in {}^\bullet\boldsymbol{t}^\bullet \downarrow P\}$. This means that M' is M in which all pairs $(x, !(...))$ and $(x, ?(...))$ belonging to ${}^\bullet\boldsymbol{t}^\bullet$ are replaced with pairs (x, Θ), i.e. M' is the result of "firing" transmission $\boldsymbol{t}$ at the state M. This models the transfer of a message from senders to receivers and disruption of the communication channel.

In Fig.2 a collection of 6 ports of agents $A_p, A_{p_1}, A_{p_2}, A_{p_3}, A_{p_4}, A_{p_5}$ are depicted. The global state of this system is

$M =$

p	p_1	p_2	p_3	p_4	p_5
$!(p_1, p_4)$	$?(p)$	Θ	Θ	$?(p)$	Θ

Transmission $\boldsymbol{t} = (\{(p{:}!(p_1, p_4))\}, \{(p_1{:}?(p), (p_4{:}?(p))\})$ transforms M

into $M' =$

p	p_1	p_2	p_3	p_4	p_5
Θ	Θ	Θ	Θ	Θ	Θ

in effect of sending

simultaneously a message from agent A_p to A_{p_1} and A_{p_4}

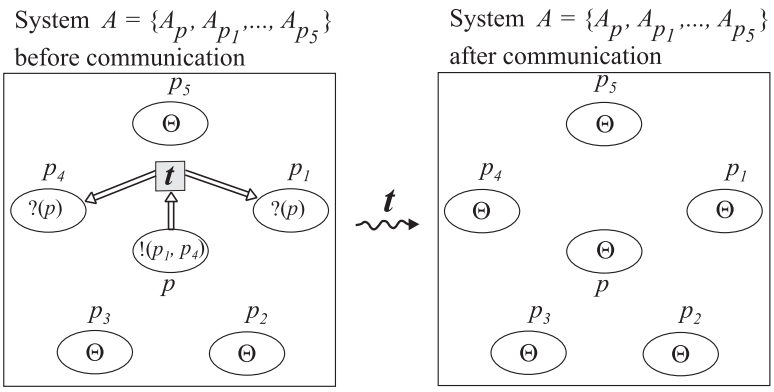


Fig. 2. Transmission of a message from agent A_p to A_{p_1} and A_{p_4} through channel $\Longrightarrow$. Here, ${}^\bullet\ \boldsymbol{t}^\bullet \downarrow P = \{p, p_1, p_4\}$

2.2 Existence of transmissions

A transmission $\boldsymbol{t} \in \boldsymbol{TR}$ exists in a global state M of the system A iff ${}^\bullet\boldsymbol{t}^\bullet \subseteq M$. Given a global state M_0 and a transmission $\boldsymbol{t}$, the realizability of $\boldsymbol{t}$ starting computation from M_0 is expressed as: does there exist a state M reachable from M_0 such that ${}^\bullet\boldsymbol{t}^\bullet \subseteq M$? Thus, the existence problem for $\boldsymbol{t}$ reduces to some versions of state reachability and inclusion problems. Their solution in the

form of yes/no decision and, possibly, their complexity, depends obviously on the formal description of agents. For example, let us assume that agents are described as 1-safe finite Petri nets (places are valued in the set {0,1}) obtained by replacing labels of arrows entering ports by weights 1. Then, the existence of $\boldsymbol{t}$ reduces to the problem "For a given marking M_0 and place p, is there a reachable from M_0 marking with a token in p?", which is known to be the PSPACE-hard (PSPACE - the set of all decision problems solvable by a Turing machine with a polynomial amount of space), see e.g. [ESP 1998]. Indeed, after the replacement of arrow labels, the whole system becomes one disconnected 1-safe Petri net. Denote by m_0 the marking of it, such that each place (port) $x \in {}^\bullet\boldsymbol{t}^\bullet \downarrow P$ holds a token (i.e. $m_0(x) = 1$) iff $M_0(x) \neq \Theta$. In such system net each $x \in {}^\bullet\boldsymbol{t}^\bullet \downarrow P$ has no outgoing arrow, thus, if a token enters this place at a certain marking reachable from m_0, then it will stay there indefinitely. Now, decide if there is a marking m reachable from m_0 and satisfying $m(x) = 1$ for all $x \in {}^\bullet\boldsymbol{t}^\bullet \downarrow P$. If yes (and only if), then in the original system (before replacement of the labels of arrows entering ports) there exists the transmission $\boldsymbol{t}$, because ${}^\bullet\boldsymbol{t}^\bullet \subseteq M$, where M is m restricted to ports $x \in {}^\bullet\boldsymbol{t}^\bullet \downarrow P$ holding communication operations !(...) and ?(...) instead of tokens.

Note that the assumption on agents' internal (i.e. without communication) activity as specified by Petri nets, corresponds to the concept of self-modifying nets ([B-D 1997], [Val 1978], [Val 1981], [Cza 2013]). Indeed, transmissions are in fact a special kind of transitions appearing and disappearing, so the system changes its structure in the course of its performance.

3 Semantics of the System A and some PSPACE-hard Decision Problems of its Behaviour

Let $T = \bigcup_{p\in P} T_p$ and $F = \bigcup_{p\in P} F_p$ i.e. the set of all transitions and arrows in the system A respectively. The triple $A = (P, T, F)$, denoted also by A, is a *net representation* of the system. Its semantics is the union of semantics of the transitions $t \in T$ and message transmissions $\boldsymbol{t}\in\boldsymbol{TR}$: $[[A]] = \bigcup_{\tau\in T\cup\boldsymbol{TR}} [[\tau]]$. If $(M, M') \in [[A]]$ then M' is the next to M state evoked by a transition $t \in T$ or a transmission $\boldsymbol{t}\in\boldsymbol{TR}$. For $\tau \in V = T \cup \boldsymbol{TR}$ denote $M \xrightarrow{\tau} M'$ iff $(M, M') \in [[\tau]]$. A *run* starting at M_0 is a chain $M_0 \xrightarrow{\tau_1} M_1 \xrightarrow{\tau_2} M_2 \xrightarrow{\tau_3} ...$, finite or infinite, but if finite $M_0 \xrightarrow{\tau_1} M_1 \xrightarrow{\tau_2} M_2 \xrightarrow{\tau_3} ... \xrightarrow{\tau_n} M_n$ then none M satisfies $(M_n, M) \in [[A]]$. A finite or infinite word $v = \tau_1\tau_2\tau_3... \in V^\omega = V^* \cup V^\infty$ occurring at this run is a *path* starting at M_0. If finite $v = \tau_1\tau_2\tau_3...\tau_n \in V^*$ then $M \xrightarrow{v} M'$ denotes $M \xrightarrow{\tau_1} M_1 \xrightarrow{\tau_2} M_2 \xrightarrow{\tau_3} ... \xrightarrow{\tau_n} M'$. The set of all finite and infinite runs starting at M is $RUN_*(M)$ and $RUN_\infty(M)$ respectively and $RUN(M) = RUN_*(M) \cup RUN_\infty(M)$. The set of respective paths is $PATH(M) = PATH_*(M) \cup PATH_\infty(M)$, thus $PATH(M) \subseteq V^\omega$.

Assuming, as above, that agents are described by 1-safe Petri nets obtained by replacing labels of arrows entering ports by weights 1, one can simulate behaviour

of the system by a 1-safe net as follows. Let a state M be given. For each transmission $\boldsymbol{t} \in \boldsymbol{TR}$ create a transition $t \notin T$ defined as $t = ({}^\bullet t, t^\bullet)$ with ${}^\bullet t = {}^\bullet \boldsymbol{t}^\bullet \downarrow P$, $t^\bullet = \emptyset$, and make arrows from ports $p \in {}^\bullet \boldsymbol{t}^\bullet \downarrow P$ to t. The extended net is a triple $\overline{A} = (P, \overline{T}, \overline{F})$, where $\overline{T} = T \cup$ *set of newly created transitions*, and $\overline{F} = F \cup$ *set of newly created arrows weighted with* 1. A marking of $\overline{A}$ is obtained from marking of A by replacing operations !(...), ?(...) with tokens wherever such operations are in some ports and removing Θ from remaining ports. Fig.3 depicts a simulation of transmission $\boldsymbol{t}$ from Fig.2 by the newly created transition t and result of its firing. Remember: while $\boldsymbol{t}$ appears and disapears in the course of the system activity, the transition $t \in \overline{T}$ is the ordinary transition of the Petri net $\overline{A}$ simulating system A, thus a unchangeable member of the $\overline{A}$'s static structure.

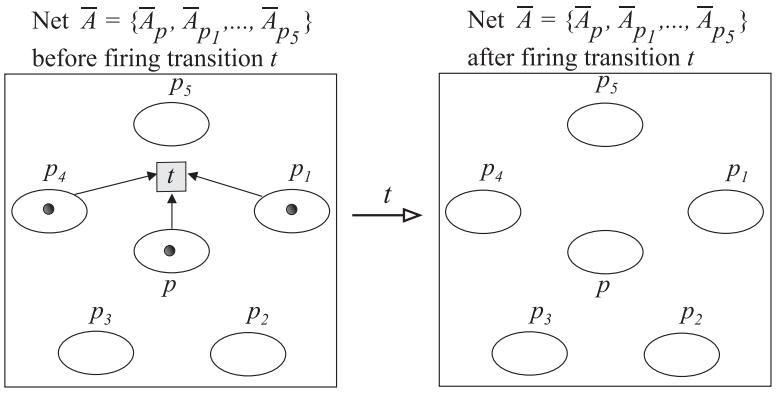


Fig. 3. Transition $t = (\{p, p_1, p_4\}, \emptyset)$ with empty post-set simulates behaviour of message transmission $\boldsymbol{t}$ in Fig.2

Some problems concerning behaviour of the system A may be reduced to problems concerning behaviour of the Petri net $\overline{A}$. To mention a few (suppose runs start from a given marking):

a. Existence of run with a given message transmission occurrence
b. Existence of reachable dead marking (no transition can fire at it)
c. Existence of finite run (equivalent to b)
d. Existence of infinite run
e. Existence of run with infinite number of a given message transmission occurrence
f. Existence of run with never accomplished a given request for communication

All these problems are PSPACE-hard for 1-safe Petri nets ([ESP 1998]) and $\overline{A}$ is such net. Therefore, by virtue of the obviously polynomial simulation procedure described above, the problems for systems specified like A in this paper, are PSPACE-hard provided that internal activity of agents is specified by 1-safe Petri nets.

4 Deadlock and Fairness: Emptiness and Finiteness of Sets of Paths

Out of several concepts and kinds of deadlock and fairness found in diverse models of distributed computing, let us consider those arising from communication and described in terms of the model pursued here.

4.1 Deadlock

System A is deadlock-free at a state M if for each agent requesting for communication there is a finite path starting at M, such that the agent will be permitted to accomplish the request on this path. A deadlock is a negation of this property. For an agent $A_p \in A = (P, T, F)$, with port p and for a state $M \in \mathbb{S}$ define:

$$D_p(M) \overset{def}{\iff} \neg[\exists M'.\exists v.(M \xrightarrow{v} M' \wedge \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet))]$$

where $\boldsymbol{t} \overset{*}{\in} v$ means "transmission $\boldsymbol{t} \in \boldsymbol{TR}$ occurs on the path v" and $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$ means "$\boldsymbol{t}$ accomplishes request for communication issued by agent A_p and pending at the state M".
In words: never agent A_p requesting for communication at the state M will be permitted to accomplish the request by a certain transmission occurring on whichever finite path starting at M.
The system is subject to a deadlock at the state M iff:
$\exists p.M(p) \neq \Theta \wedge D_p(M)$.

Proposition 4.1.1 (set-theoretic characterization)

$D_p(M)$ if and only if $PATH(M) \cap V^* \boldsymbol{t} V^* = \emptyset$ for each $\boldsymbol{t}$ satisfying $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$

Proof
$D_p(M) \iff$
$\neg\exists M'.\exists v.(M \xrightarrow{v} M' \wedge \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$ (swapping quantifiers)
$\neg\exists v.\exists M'.(M \xrightarrow{v} M' \wedge \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$ (De Morgan law)
$\forall v.\neg\exists M'.(M \xrightarrow{v} M' \wedge \underbrace{\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)}_{\text{no } M' \text{ in this formula}}) \iff$
$\forall v.\neg((\exists M'.M \xrightarrow{v} M') \wedge \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$ (De Morgan law)
$\forall v.(\neg(\exists M'.M \xrightarrow{v} M') \vee \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$
$\forall v.((\exists M'.M \xrightarrow{v} M') \Rightarrow \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$
$\forall v.(v \in \{u \mid \exists M'.M \xrightarrow{u} M'\} \Rightarrow \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$ (definition of $PATH(M)$)
$\forall v.(v \in PATH(M) \Rightarrow \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \iff$
$\{v \mid v \in PATH(M)\} \subseteq \{v \mid \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)\} \iff$

$PATH(M) \subseteq \{v|\, \neg\exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)\} \Longleftrightarrow$
$PATH(M) \subseteq V^* - \{v|\, \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)\}$

Therefore:
$PATH(M) \subseteq V^* - V^*\boldsymbol{t}V^*$ for each $\boldsymbol{t}$ satisfying $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$ where $V^*\boldsymbol{t}V^*$ is the set of all finite words over V where $\boldsymbol{t}$ occurs. Thus:
if $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$ then $PATH(M) - (V^* - V^*\boldsymbol{t}V^*) = \emptyset$.
Since $X - (Y - Z) = (X - Y) \cup (X \cap Z)$ for any sets X, Y, Z then
$PATH(M) - (V^* - V^*\boldsymbol{t}V^*) = (PATH(M) - V^*) \cup (PATH(M) \cap V^*\boldsymbol{t}V^*) = \emptyset$
(Because $PATH(M) - V^* = \emptyset$). Finally:
$D_p(M)$ iff $\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow PATH(M) \cap V^*\boldsymbol{t}V^* = \emptyset)$ □

Theorem 4.1.1

A deadlock at a state M occurs iff:

$$\exists p.[M(p) \neq \Theta \wedge (\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow PATH(M) \cap V^*\boldsymbol{t}V^* = \emptyset))] \quad \square$$

Thus decidability of such deadlocks reduces to deciding whether transmission $\boldsymbol{t}$ does not occur on any path starting from M (provided that there are a finite number of agents, thus also transmissions), which depends on algebraic structure of the set $PATH(M)$.

4.2 Weak fairness

System A is *weakly* fair at a state M if each agent requesting for communication at M will be permitted to accomplish the request on every infinite path starting from M. This is expressed by the formula:
$\forall p.[M(p) \neq \Theta \Rightarrow \forall v.(v \in PATH_\infty(M) \Rightarrow \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet))]$

Theorem 4.2.1

System A is weakly fair at a state M iff

$$\forall p.[M(p) \neq \Theta \Rightarrow (\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow PATH_\infty(M) - V^*\boldsymbol{t}V^\infty = \emptyset))]$$

Proof
$\forall v.(v \in PATH_\infty(M) \Rightarrow \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)) \Longleftrightarrow$
$\{v|\, v \in PATH_\infty(M)\} \subseteq \{v|\, \exists \boldsymbol{t}.(\boldsymbol{t} \overset{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)\} \Longleftrightarrow$
$PATH_\infty(M) \subseteq V^*\boldsymbol{t}V^\infty$ for each $\boldsymbol{t}$ satisfying $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$. Thus
$\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow PATH_\infty(M) - V^*\boldsymbol{t}V^\infty = \emptyset)$ □

4.3 Strong fairness

System A is *strongly* fair at a state M if each agent requesting for communication at M will be permitted to accomplish the request on every finite path starting at M if all these paths are "sufficiently long", i.e. of the length at least k, for a certain k. So, all these paths may be jointly ("uniformly") bounded in lentgh. This is expressed by the formula:
$\forall p.[M(p) \neq \Theta \Rightarrow \exists k.F_p(M,k)]$ where
$F_p(M,k) \stackrel{def}{\Longleftrightarrow} \forall v.((v \in PATH_*(M) \wedge |v| \geq k) \Rightarrow \exists \boldsymbol{t}.(\boldsymbol{t} \stackrel{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)$

Theorem 4.3.1

System A is strongly fair at a state M iff

$\forall p.[M(p) \neq \Theta \Rightarrow (\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow |PATH_*(M) - V^*\boldsymbol{t}V^*)| < \infty)]$

Proof
$F_p(M,k) \Longleftrightarrow$
$\{v|\, v \in PATH_*(M)\} \cap \{v|\, |v| \geq k\} \subseteq \{v|\, \exists \boldsymbol{t}.(\boldsymbol{t} \stackrel{*}{\in} v \wedge p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet)\} \Longleftrightarrow$
$PATH_*(M) \cap V^kV^* \subseteq V^*\boldsymbol{t}V^*$ for each $\boldsymbol{t}$ satisfying $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$
where V^kV^* is the set of finite words of the length at least k and $V^*\boldsymbol{t}V^*$ the set of finite words where $\boldsymbol{t}$ occurs. Thus:
$\forall \boldsymbol{t}.(p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet \Rightarrow PATH_*(M) \cap V^kV^* - V^*\boldsymbol{t}V^* = \emptyset)$.
Now, let $H_p(M,k) = PATH_*(M) \cap V^kV^* - V^*\boldsymbol{t}V^*$ for each $\boldsymbol{t}$ satisfying $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$.
We show that $|H_p(M,0)| < \infty$ if and only if $\exists k.H_p(M,k) = \emptyset$
($|X|$ is cardinality of the set X).
($\Rightarrow$) Let $\forall k.H_p(M,k) \neq \emptyset$. Then $\lambda(H_p(M,k)) < \lambda(H_p(M,k+1))$ where $\lambda(L)$ is the length of a shortest word in the set L.
Thus $\lim\limits_{k\to\infty} \lambda(H_p(M,k)) = \infty$, which implies
(since $H_p(M,k) \supset H_p(M,k+1)$) that $H_p(M,0)$ contains words of arbitrary length, hence $|H_p(M,0)| = \infty$.
($\Leftarrow$) Let $|H_p(M,0)| = \infty$. Then $H_p(M,0)$ contains words of arbitrary length, thus, for any k it contains a word w with $|v| \geq k$. Because $v \in V^kV^*$ and $H_p(M,k) = H_p(M,0) \cap V^kV^*$ we have $v \in H_p(M,k)$ hence $H_p(M,k) \neq \emptyset$. □

4.4 Equivalence of weak and strong fairness

To demonstrate the equivalence between the two kinds of fairness in the model considered here, let us recall a version of the:

König's Lemma [Kön 1927]:
Let Σ be a set and Υ a tree of the properties:

- the number of sons of every node in Υ is finite;
- for any $k \geq 0$ there is a finite branch b in Υ with $|b| \geq k$ and $b \subseteq \Sigma$.

Then there exixts a infinite branch B in Υ with $|B| \subseteq \Sigma$.

Theorem 4.4.1
The weak and strong fairness are equivalent.

Proof
By the Theorem 4.2.1 and 4.3.1 it suffices to demonstrate that $|PATH_*(M) - V^*\boldsymbol{t}V^*| < \infty \iff PATH_\infty(M) - V^*\boldsymbol{t}V^\infty = \emptyset$ for each transmission $\boldsymbol{t}$ such that $p{:}M(p) \in {}^\bullet\boldsymbol{t}^\bullet$ for every port p with $M(p) \neq \Theta$.
Implication "$\Rightarrow$" is evident, it remains to show "$\Leftarrow$". Suppose $|PATH_*(M) - V^*\boldsymbol{t}V^*| = \infty$. Note that the set of paths starting at M is prefix-closed: each prefix of $v \in PATH(M)$ belongs to $PATH(M)$. To each v assign a unique element $node(v)$ in this way that $v_1 \neq v_2 \Rightarrow node(v_1) \neq node(v_2)$ and let $NODE(M) = \{node(v) | \, v \in PATH_*(M)\}$. This set with internodal relation defined by "$node(u)$ is father of $node(v)$ iff $v = u\tau$ for a certain $\tau \in V$" is a tree Υ with $node(\varepsilon)$ as the root (ε is the empty path) and finitely many sons of each father. So, every path $v \in PATH_*(M)$ is a branch in Υ. By assumption $|PATH_*(M) - V^*\boldsymbol{t}V^*| = \infty$ there are infinitely many finite paths, thus branches in Υ on which no $\boldsymbol{t}$ exists. Therefore there must be an arbitrarily long branch in the tree. Setting $\Sigma = PATH_*(M)$ and applying the König's Lemma, we come to contradition. □

Summing up the results obtained above, the set-theoretic characteristics of the deadlock and fairness at a state M are in the following table:

Deadlock	$PATH(M) \cap V^*\boldsymbol{t}V^* = \emptyset$
Weak fairness	$PATH_\infty(M) - V^*\boldsymbol{t}V^\infty = \emptyset$
Strong fairness	$\lvert PATH_*(M) - V^*\boldsymbol{t}V^*\rvert < \infty$

for every transmission $\boldsymbol{t}$.

5 Counting States

If the agents do not send and receive messages to/from themselves then the total number of (global) states of n-agent system is $(2^n - 1)^n$. Indeed, each agent may issue $\frac{2^n}{2} - 1$ send !(...) requests and the same number of receive ?(...) requests, that is $2^n - 2$ requests for communication. Since the agent may assume Θ as its local state, the number of local states it may assume is $2^n - 1$. The set of global states is the Cartesian product of sets of the local states of all agents. Therefore the number of global states is $\underbrace{(2^n - 1) \cdot \ldots \cdot (2^n - 1)}_{n \text{ times}} = (2^n - 1)^n$. For instance, for agents p_1, p_2, p_3:
the set of local states of $p_1 = \{\Theta, !(p_2), ?(p_2), !(p_3), ?(p_3), !(p_2, p_3), ?(p_2, p_3)\}$
the set of local states of $p_2 = \{\Theta, !(p_1), ?(p_1), !(p_3), ?(p_3), !(p_1, p_3), ?(p_1, p_3)\}$
the set of local states of $p_3 = \{\Theta, !(p_1), ?(p_1), !(p_2), ?(p_2), !(p_1, p_2), ?(p_1, p_2)\}$
Thus, the system of three agents has $7^3 = 343$ global states.

References

[B-D 1997] Badouel E., Darondeau P., *Stratified Petri Nets*, FCT'97, Lecture Notes in Computer Science vol. 1279 (1979), pp. 117-128

[Cza 2013] Czaja L., *Self-Modifying Nets for Synchronous, Connection-Oriented, Multicast Communication*, Fundamenta Informaticae, to appear

[ESP 1998] Esparza J., *Decidability and Complexity of Petri Net Problems - An Introduction*, Lecture Notes in Computer Science, Vol. 1491, 1998, pp. 374-428

[Hoa 1978] Hoare C.A.R., *Communicating Sequential Processes*, Comm. of the ACM, Vol 21, pp. 666-677, 1978

[Hoa 1985] Hoare C.A.R., *Communicating Sequential Processes*, Prentice-Hall, 1985

[Kön 1927] König D., *Über eine Schlussweise aus dem Endlichen in Umendliche*, Acta Litt. Ac. Sci. Hung. Fran. Josep. 3 (1927), pp. 121-130

[OCCAM 1984] INMOS Limited: *OCCAM Programming Manual*, Prentice-Hall, 1984

[Rei 1985] Reisig W., *Petri Nets, An Introduction*, EATC Monographs on Theoretical Computer Science, Springer-Verlag, 1985

[Val 1978] Valk R., *Self-Modifying Nets, a Natural Extension of Petri Nets*, Icalp'78, Lecture Notes in Computer Science vol. 62 (1978), pp. 464-476

[Val 1981] Valk R., *Generalization of Petri Nets*, MFCS'81 Lecture Notes in Computer Science vol. 118 (1981), pp. 140-155

The Mathematical Model for Interference Simulation and Optimization in 802.11n Networks

Iwona Dolińska, Antoni Masiukiewicz, and Grzegorz Rządkowski

Vistula University, Warsaw, Poland, http://www.vistula.edu.pl

Abstract. One of the key problems in 802.11 standard networks are interferences. It is not possible to avoid the influence of other wireless systems. One can only minimize the power level of unwanted signals. Typically the designer should find the best localization of access points (AP), but there is no planning and coordination between different private networks. To reduce the level of interferences, the transmitting power reduction is applied. The mathematical model was built to analyze the relationship between the coverage and the level of interferences. The results of these simulations are presented in this article.

Keywords: Wi-Fi, interferences, 802.11n standard, WLAN, throughput

1 Introduction

The 802.11 standard networks are the most popular solution of wireless communication today, besides the mobile telephony networks. One of the key problems in such networks is the issue of interferences (see [6]). The main sources of interferences are various radio systems or devices, which operate on the same or similar frequency range. These networks produce both, adjacent and inter channel interferences. The reduction of internal system interferences is crucial for obtaining the proper QoS of the transmissions (see [1]). Basic methods of the interference limitation implement proper planning, which means the proper arrangement of the access point localizations. The next step is a selection of the transmission frequency dedicated for each channel. Such planning is not possible in any network. One of the important features of the 802.11 networks is the fact, that they operate at the public free frequency range (ISM Industrial, Scientific, Medical), so the high number of different devices can operate at the same time on a similar area. These devices could be elements of home or office networks. There is no coordination between such networks.

Another method of interference reduction is the diminish of the transmitted power (see [7]). In the authors opinion this method is not very efficient especially, if the coverage is an important issue. The authors built the theoretical model and carried out several calculations to show both advantages and disadvantages of such solution.

2 The Structure of 802.11n Physical Layer

The structure of the Physical Layer has a great influence on the internal system interference level. For 2.4 GHz transmission frequency range, only three channels (numbered 1, 6 and 11) are the so called not overlapping channels (see [9]). The standard deviation between the central frequencies of these three channels is 25 MHz. The level of signal within a channel is limited by the mask. The mask is a filter with specially developed characteristic. The characteristics of filters for 1, 6 and 11 channels in the 802.11n standard are presented in Fig. 1.

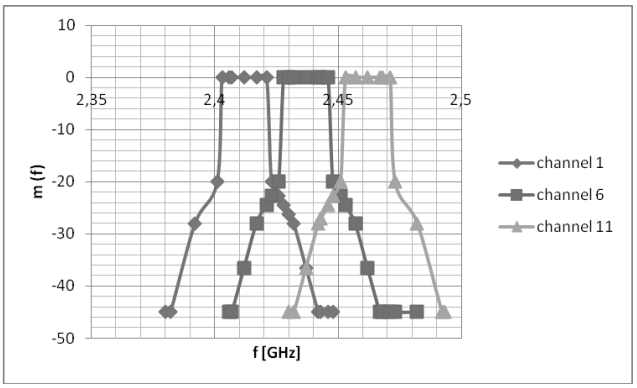


Fig. 1. Filter's (masks) characteristics for 1, 6, 11 channels in the 802.11n standard. Source: own preparation.

The channel masks overlap partly, even for non overlapping channels. Some interchannel interferences are always present in the system, when more than one network is operating on the same area. The final level of the interference signal power depends strongly on many parameters. The distance plays an important role, because the level of the received power decreases while increasing the distance between Wi-Fi stations. Two disadvantages are produced by interferences (see [2]). The first is the diminish of signal to noise ratio, because the interference power is treated as noise within the transmission channel. The thermal noise and the interference power are produced by uncorrelated sources, so we can calculate the summarized noise as the sum of power density of the thermal noise and the interference power (see [8]):

$$P_{noise} = P_{int} + P_{white\_noise}\,. \tag{1}$$

The noise power diminishes the channel throughput. The throughput is the most important parameter determining the QoS of the transmission. The channel throughput could be described by the following formula (see [3]):

$$C = B \ln\left(1 + \frac{P_{signal}}{P_{noise}}\right), \tag{2}$$

where B represents the bandwidth of the transmission channel. The second disadvantage of interferences especially, when their signal power is relatively high, is the effect of the spurious carrier detection. The high level of interference power blocks the transmission channel. Some methods of interference level reduction are discussed in the next section.

3 The Methods of Interference Level Reduction

A high level of interference power could be reduced by a proper arrangement of access points (see [4]). It is possible only in some networks eg. private networks, company networks. On the other hand, in some networks, the access points are arranged in a totally chaotic way. There is no coordination of AP localization and no coordination of utilized channels. An example of a set of private networks is shown in Fig. 2. Such a situation happens very frequently, especially in the multi-family or office buildings.

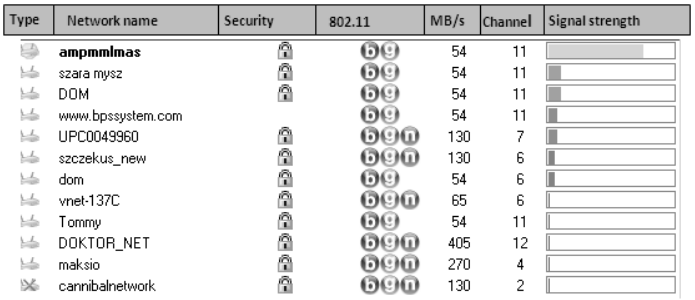


Fig. 2. The sample set of private 802.11 networks. Source: own measurement (Card WLAN Monitor–Dell Wireless 1450).

Many devices use the same channel (5 devices - channel nr 11), some devices use channels other then 1, 6, 11, so the choice of a channel is random. Network planning let us achieve capacity, range and QoS (see [7], [4], [5]). There are several methods of WiFi network planning described in the literaturte, e.g. Neldeare-Mead direct planning (see [4]). This method enables the optimal determination of localization of AP stations. The coefficient of channel frequency reuse could be calculated (co channel interference reduction factor). This factor is the function of the number of available channels/frequencies especially those not overlapping and could be expressed by following formula:

$$Q = \sqrt{3}N, \tag{3}$$

where the N is the number of available channels.

The second solution suggests [7] reduction of transmitted power, but as a side effect a decrease of coverage occurs. This solution reduces the interference

power level, but on the other hand leads to dead areas with no coverage, what is shown in Fig. 3.

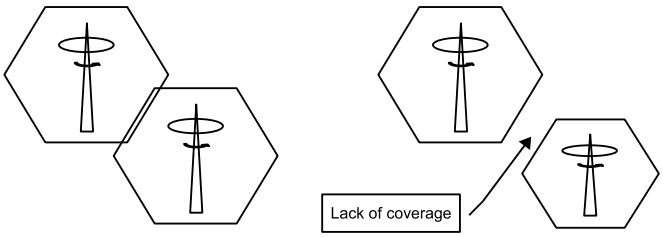


Fig. 3. Effects of transmitted power reduction. Source: own preparation.

The smaller is the coverage of one cell, the more cells we have to produce to obtain the full coverage. This means more APs and in the end, more transmissions at the same time, but it does not mean that we reduce the interference power level. The authors present some proof in Sec. 5 and Sec. 6.

4 Correlation Between Coverage, Transmitted Power and the Interference Power Level

The basic equation, which describes the radio wave distribution in a free space is the Friis formula (see[3]):

$$P_{rx}\left(r\right) = \frac{P_{tx}G_{rx}G_{tx}\lambda^2}{\left(4\pi r\right)^2} = \frac{P_{tx}G_{rx}G_{tx}c^2}{\left(4\pi r\right)^2 f^2}. \tag{4}$$

This formula allows us to calculate the received power (P_{rx}) depending on the transmitted power (P_{tx}), the gains of receiving and transmitting antennas (G_{tx}, G_{rx}), the channel frequency f and the distance between the transmitter and receiver (in so called free space, the r power is equal 2):

$$P_{rx}(r) = \frac{k}{f^2 r^2}. \tag{5}$$

Using the formula (4) we can calculate the attenuation of radio signal in a free space:

$$P_{odb}\left(r\right) = P_{tx}G_{rx}G_{tx}L_{fspl}, \tag{6}$$

$$L_{fspl} = \frac{c^2}{\left(4\pi r\right)^2 f^2} = \frac{\left(3\cdot 10^8\right)^2}{\left(4\pi r\cdot 10^3\right)^2 f^2\cdot\left(10^6\right)^2} = \left(\frac{40\pi fr}{3}\right)^{-2}. \tag{7}$$

L_{fspl} could be presented in the logarithmic scale:

$$L_{fspl}[dB] = -10\log L_{fspl}, \tag{8}$$

and finally we obtain the following formula [6]:

$$L_{fspl}[dB] = 32,44 + 20\log r[km] + 20\log f[MHz], \tag{9}$$

where r is in [km] and f in [MHz]. The more general formula takes the following form (see [6]):

$$L_{fspl} = \frac{c^2}{16\pi^2 r^\alpha f^2} \tag{10}$$

This formula for frequencies f in [Hz] or [MHz] can be rewritten respectively as:

$$L_{fspl}[dB] = -147.6 + 10\alpha\log r[m] + 20\log f[Hz], \tag{11}$$

$$L_{fspl}[dB] = -27.56 + 10\alpha\log r[m] + 20\log f[MHz]. \tag{12}$$

The α coefficient is rather unstable in time and very sensitive to the environment e.g. it changes strongly in rooms.

The coverage in the 802.11n standard is determined by the minimal received power (received signal sensitivity), which is necessary for obtaining required level of throughput. The set of minimal received power in the case of a single spatial transmission in 802.11n standard is presented in Table 1.

Table 1. Minimal received signal sensitivity for the station operating in SISO mode

MCS Index	**Modulation/coding**	**Data Rate [Mbit/s] 20MHz channel**	**Received signal sensitivity [dBm]**
0	BPSK/1:2	6.5	-82
1	QPSK/1:2	13.0	-79
2	QPSK/3:4	19.5	-77
3	16QAM/1:2	26.0	-74
4	16QAM/3:4	39.0	-70
5	64QAM/2:3	52.0	-66
6	64QAM/3:4	58.5	-65
7	64QAM/5:6	65.0	-64

The minimal sensitivity is respectively -82 dBm for throughput of 6.5 Mbit/s and -64 dBm for 65 Mbit/s. It is difficult to correlate these values with a specific distance, because in practice this distance could vary in a broad range due to a lot of factors.

5 The Analysis Assumptions

The authors tried to verify the assumption that the decreasing of transmission power and reduction of coverage help to diminish the internal interferences in

802.11n networks (see [7]). The analysis was reduced to a model with one spatial stream in 802.11n standard. One spatial stream means the use of the SISO (Single Input Single Output) antenna solution. The isotropic characteristic of transmission power is also the assumption. The analysis was carried out for three non overlapping channels 1, 6 and 11. The Tx and Rx configurations are presented in Fig. 4.

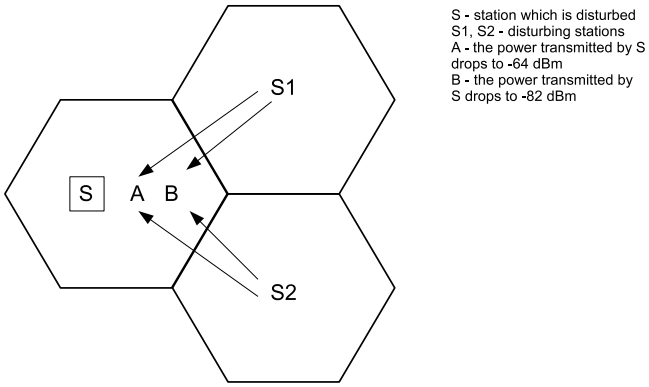


Fig. 4. Station configurationw, where the S station is disturbed by S_1 or S_2 or both the stations at the same time. Source: own preparation.

The Table 2 includes the channel allocation, which is used in simulation.

Table 2. Channel number allocation for stations S, S_1 and S_2

Channel Number		
S1	**S**	**S2**
11	6	-
11	6	11
6	6	11
6	6	6
-	6	6
1	6	1

We assume that all stations are the transmitters. The localization of a station within the cell (coverage area) could vary from the center of the area to its edge. We analyze the 802.11n standard with 20 MHz channel bandwidth. The center frequencies for channels 1, 6, 11 are shown in Table 3.

The interference power level was calculated as the sum of interferences from stations S_1 and S_2 and white noise and the noise figure representing the noise

Table 3. Center frequencies for channels 1, 6, 11

Channel number	Center frequency [GHz]
1	2.412
6	2.437
11	2.462

of electronic circuits (mainly electronic amplifiers).

$$P_{int} = \sum_{x=1}^{2} P_{intx} + (P_{white\_noise} + P_{NF}) \ . \tag{13}$$

White noise or thermal noise [3] within the channel bandwidth could be described as:

$$P_{white\_noise}(f) = kT\,[W/Hz] \ , \tag{14}$$

T denotes the environment temperature in K degree, while k is a Boltzman constant. Threshold of white noise in 1 Hz bandwidth at 0 Kelvin degree is -228.6 dBW. White noise in B bandwidth can be calculated as:

$$P_{white\_noise}[dBm] = 10\log(kTB)\,. \tag{15}$$

The white noise in 20 MHz channel at 17 C degree could reach the following level:
$P_{white\_noise}(T = 17°C, B = 20MHz) = -174 + 10\log B = -131dBm$.
The following formula was developed by the authors to calculate the received power:

$$\begin{aligned} P_{received}(r) = {} & M(f - 2412 - 5(K-1)) + P_{transmitted} \\ & -(-27,56 + 10\alpha \log r[m] + 20\log f[MHz]) + G_{sum}. \end{aligned} \tag{16}$$

We will denote $P_{received}$ and $P_{transmitted}$ by P_{rx} and P_{tx} respectively. The $M(f)$ function represents the mask (filter) of the relevant channel. The signal outside the mask is eliminated while the one below the mask characteristics passes. The authors assume that the mask characteristic determines the maximal internal level of interferences. G_{sum} is equivalent to the additional gain of the system including the influence of the antennas of the receiver and the transmitter and respectively the gain connected with modulation, coding and different types of signal dispersions. P_{tx}, α and G_{sum} are the parameters of the simulation and their values are presented in Table 4.

The parameters of $m(f)$ function correspond to the mask of 802.11n standard. We assume that the function (for f in MHz) is continuous, piecewise linear and

Table 4. Simulation parameters values range

Transmitted power [dBm]	**Alpha parameter**	**Additional gain [dB]**
-10 to 20	2 to 8	0 to 15

is described by the following formula:

$$m(f) = \begin{cases} 0 & \text{for } f \in (-\infty, -30] \\ 2f + 60 & \text{for } f \in [-30, -20] \\ \frac{5}{9}f + \frac{280}{9} & \text{for } f \in [-20, -11] \\ 10f + 135 & \text{for } f \in [-11, -9] \\ 45 & \text{for } f \in [-9, 9] \\ -10f + 135 & \text{for } f \in [9, 11] \\ -\frac{5}{9}f + \frac{280}{9} & \text{for } f \in [11, 20] \\ -2f + 60 & \text{for } f \in [20, 30] \\ 0 & \text{for } f \in [30, \infty) \end{cases} \tag{17}$$

The stations are placed on Cartesian plane. The S transmitter has the (x, y) coordinates and S_1 respectively (x_1, y_1). The distance d_1 between the above stations is equal:

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}\,. \tag{18}$$

The practical formula for interference power level, which influences the S station (operating on channel 6) in d_1 distance from the station S_1, producing interferences while operating on channel 11, will be as follows:

$$P_{rx}(d_1) = M(f - 2462) + P_{tx} - (-27.56 + 10\alpha \log d_1[m] + 20 \log 2462) + G_{sum}. \tag{19}$$

The average interference power within the whole channel is the integral from P_{rx} over f within the proper channel (6th in our case):

$$P_{rx}(d_1)average = \int_{2.427}^{2.447} P_{rx} df\,. \tag{20}$$

The authors correlate the distance d_1 with the minimal received signal sensitivity (see Table 3).

6 The Simulation Results

In the first simulation the station S, operating on channel 6, was disturbed by S_1 station, operating on channel 11. The transmitting power distribution was firstly simulated. The figure 5 shows the distribution of points correlated with

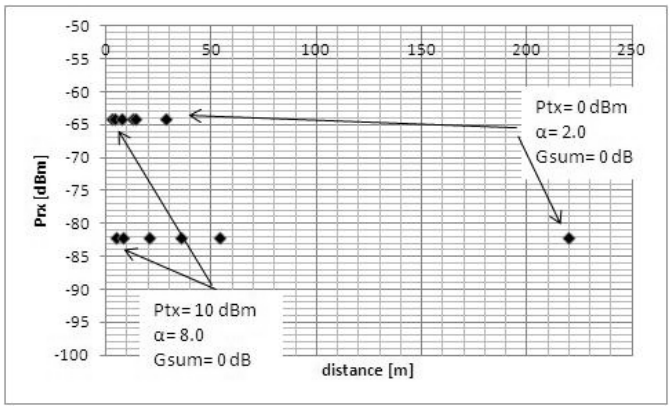


Fig. 5. Change of coverage in channel 6 versus P_{rx}, α and G_{sum}. Source: own preparation.

respectively -64 dBm and -82 dBm of received power (upper and lower lines). This analysis concerns the transmission in channel 6. We assume that the power transmitted by the S station is equal to receiver sensitivity.

The points corresponding to the -64dBm received signal are within the range from single meters to about 25 meters, while these corresponding with -82 dBm are within the range from a few meters to more than 200 meters. The coverage diminishes especially for the higher value of α. Figure 6 presents the characteristics of the diminish of transmitted power with distance for different values of P_{rx}, α and G_{sum}. The critical parameter is the α. The highest slope is for $\alpha = 8$.

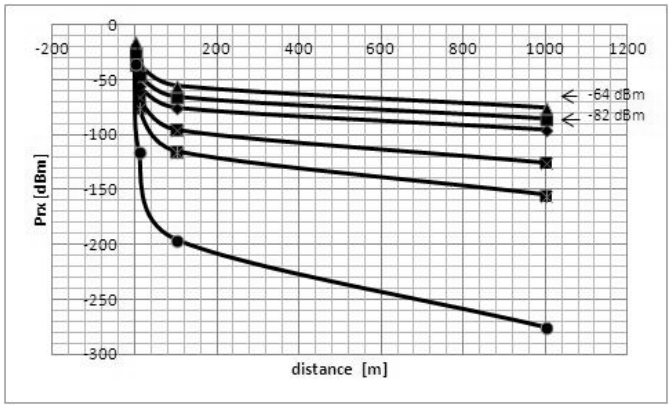


Fig. 6. The characteristics of received power in channel 6 versus P_{rx}, α and G_{sum}. Source: own preparation.

The characteristics of the interference power for the fixed distance beetwen S and S_1 are presented in Fig. 7. The P_{int} characteristics are versus the level of the disturbances source power. The following assumptions are made: the S is transmiiting in the channel nr 6, P_{tx} is 10 dBm, G_{sum} is 0 [dBm] and $\alpha = 3$; while the S_1 station (disturbing) is transmitting in the channel nr 11, G_{sum} is 0 [dBm], α is 3 and the P_{tx} change in the range from -10 to 20 dBm. The point of the interference level calculation coresponds with the maximum coverage of the S station, where the P_{rx} is -82dBm. This distance is 53,38 m while the distance between stations is 106.76m. The results are shown in Fig. 7.

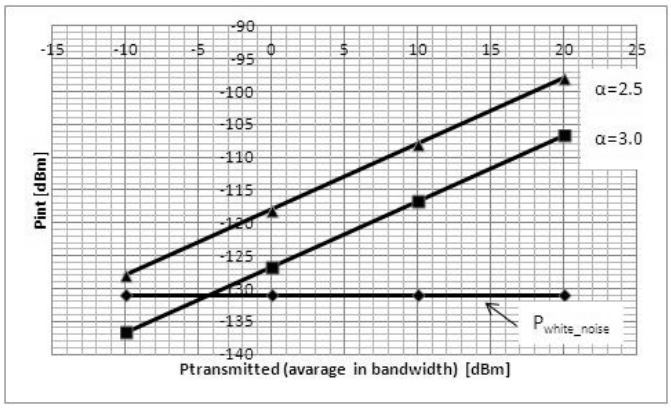


Fig. 7. The characteristics of the interferences power level for d1=53.38 m versus P_{rx}. Source: own preparation.

The interference power level diminishes, when the P_{tx} of S_1 diminish, but at the same time the coverage area is reduced, so the dead zone arises with no possibility of transmission. The white noise could have higher level than interference power for low P_{tx} of disturbing station and for non convenient transmission conditions (high value of the α coefficient- eg. rooms, halls etc.).

The characteristics of interference level with another assumption is presented in fig. 8. In this case together with the change of the P_{tx} we change the point of S_1 localization (x_1,y_1) to reduce the dead zone. The point of the interference level calculation (distance from S) is constant and its value is 53.38 m, but the distance between S and S_1 stations (d_1) diminishes relatively to P_{tx} (S_1) reduction. The dead zone is minimized. The level of interference power is constant versus P_{tx} and α values (Fig. 8). The next simulation concern the situation when the P_{tx} power is increased for both S and S_1 and the interference power level is calculated for maximum coverage points corresponds with received power equal -82 dBm (Fig. 9). The results of the simulation are the same as previously. The inteference power level is higher than the thermal noise, but if we take into account the the electronic circuits noise figure, then the total noise could be above interference power level.

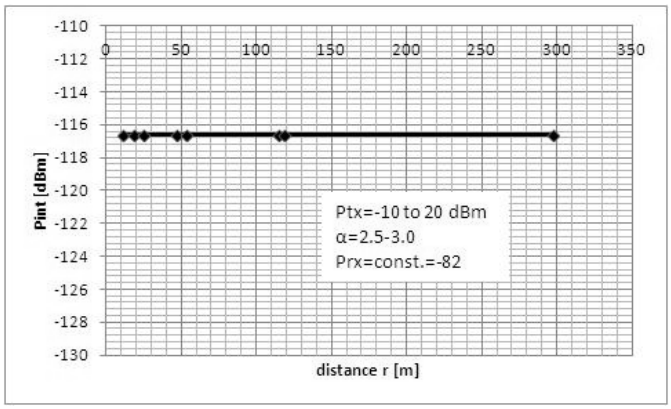


Fig. 8. The interferences level characteristics, where the distance from S is constant (53.38 m) while S_1 changes its position (r). The r distance each time corresponds to -82 dBm power level. Source: own preparation.

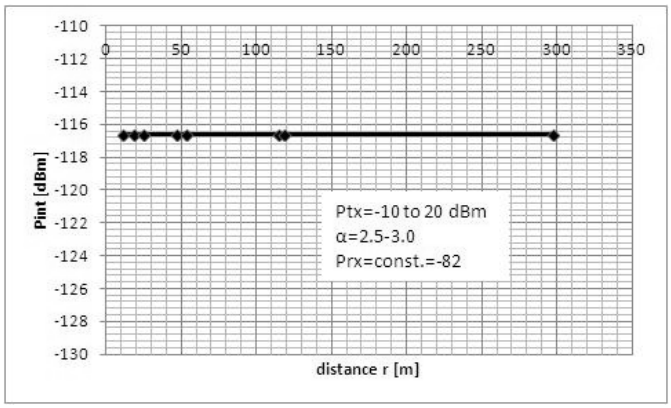


Fig. 9. The interferences power level versus changes of P_{tx}, $P_{int} = f(P_{transmitted})$ for r [m] relative to $P_{received}$ =-82 [dBm], with the assumption that both the P_{tx} and the distance between stations is reduced. Source: own preparation.

7 Conclusions

The model for interference power level simulations from one or two disturbing sources (S_1 /S_2) was developed. The model includes the efects of the mask and several other parameters such as antennas gain, modulation coding and dispersion gain (G_{sum} coefficient). The following simulations based on this model were carried out :

1. the characteristic of interference power level, when the P_{tx} of source of disturbances is reduced, but the localisation remains the same (see Fig. 7) ,

2. the characteristic of interference power level, when the P_{tx} of source of disturbances is reduced, but the localisation is changed to avoid the dead zone (see Fig. 8),
3. the characteristic of interference power level, when the P_{tx} of source of disturbances is changed as well as the power of disturbed station S and the localisation of both stations is also changed to avoid the dead zone (see Fig. 9).

Taking into account the mentioned above simulations, we can conclude:

1. for the 1st simulation: the power level of interferences decrease, but the coverage diminishes at the same time,
2. for the 2nd simulation: the interference level is constatnt,
3. for the 3rd simulation: the interference level is constatnt.

The achieved results let us make a conclusion, that reduction of P_{tx} could reduce the interference power level, but at the same time cause the dead zone to arise. This solution may be applied, if the station is close to AP, so we can temporary (for one or more sessions) reduce transmission power, keeping reasonable throughput. This solution requires communication between different APs to establish the most efficient transmission power level. Such solutions are not available nowadays.

References

1. Dolińska I., Masiukiewicz A.: Quality of service providing in WLAN networks, possibilities, challenges and perspectives. In: Information Systems in Management, Wydawnictwo SGGW, Warsaw (2012)
2. Fuxjager P., Valerio D., Ricciato F.: The Myth of Non-Overlapping Channels: Interference Measurements in IEEE 802.11. IEEE (2007)
3. Freeman R.L.: Radio system design for telecommunication, Wiley (2007)
4. Gajewski P., Wszelak S.: Optymalizacja wyboru punktow dostepowych w sieciach WLAN metoda bezposredniego poszukiwania. Przeglad Telekomunikacyjny nr 8-9, Warsaw (2007)
5. Hereman F., Joseph W., Tanghe E., Plets D. and Martens L.: Prediction of Range, Power Consumption and Throughput for IEEE 802.11n in Large Conference Rooms. In: Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP) (2011)
6. Hotgkinson T.G.: Wireless communication–the fundamentals. In: BT Technology Journal, Vol 25 No 2 (2007)
7. Juniper Networks White Paper, Coverage or Capacity–Making the Best Use of 802.11n. (2011)
8. Masiukiewicz A., Analysis and optimization of quartz crystal oscillators for minimal phase noise. PhD Thesis, Warsaw University of Technology (1997)
9. Roshan P., Leary J.: Bezprzewodowe sieci LAN 802.11–Podstawy. PWN, Warsaw (2007)

A Domain View of Timed Behaviors *

Roman Dubtsov[1], Elena Oshevskaya[2], and Irina Virbitskaite[2]

[1] Institute of Informatics System SB RAS,
6, Acad. Lavrentiev av., 630090, Novosibirsk, Russia;
[2] Institute of Mathematics SB RAS,
4, Acad. Koptyug av., 630090, Novosibirsk, Russia;
dubtsov,eso,virb@iis.nsk.su

Abstract. The intention of this paper is to introduce a timed extension of transition systems with independence, and to study its categorical interrelations with other timed "true-concurrent" models. In particular, we show the existence of a chain of coreflections leading from a category of the model of timed transition systems with independence to a category of a specially defined model of marked Scott domains. As an intermediate semantics we use a model of timed event structures, able to properly capture causality, conflict, and concurrency among events which arise in the presence of time delays of the events.

1 Introduction

The behaviour of concurrent systems is often specified in terms of states and transitions between states, the labels on the transitions represent the observable part of system's behaviour. The simplest formal model of computation able to express naturally this idea is that of labelled transition systems. However, they are a representative of the interleaving approach to concurrency and hence do not allow one to draw a natural distinction between interleaved and concurrent executions of system's actions. Two most popular "true concurrent" extensions of transition systems, aiming to overcome limitations of the interleaving approach, are asynchronous transition systems, introduced independently by Bednarczyk [1] and Shields [2], and transitions systems with independence, proposed by Winskel and Nielsen [3].

Category theory [4] has been successfully exploited to structure the tangled world of models for concurrency. Within this framework, objects of categories represent processes and morphisms correspond to behavioural relations between the processes, i.e. to simulations. The category-theoretic approach allows for natural formalization of the fact that one model is more expressive than another in terms of an "embedding", most often taking the form of a coreflection, i.e. an adjunction in which the unit is an isomorphism. For example, Hildenbrandt and

* The second author is supported in part by the RFBR (grant 12-01-00873-a), by the President Program "Leading Scientific Schools" (grant NSh-7256.2010.1), and by the Federal Program "Research and educational personnel for innovative Russia" (grant 8206).

Sassone [5] have constructed a full subcategory of a category of asynchronous transition systems and have shown the existence of a coreflection between the subcategory and a category of transition systems with independence. In their next paper [6], the authors have enriched the model of transition systems with independence by adding multi-arcs and have yielded a precise characterization of the model in terms of (event-maximal, diamond-extensional) labeled asynchronous transition systems, by constructing functors between categories of the models.

It is generally acknowledged that time plays an important role in many concurrent and distributed systems. This has motivated the lifting of the theory of untimed systems to the real-time setting. Timed transition system like models have been studied thoroughly within the two last decades (see [7,8] among others), while timed "true concurrent" extensions have hitherto received scant attention.

The aim of this paper is to introduce a timed extension of transition systems with independence, and to study its categorical interrelations with other timed "true-concurrent" models. In particular, we show the existence of a chain of coreflections leading from a category of the model of timed transition systems with independence to a category of a specially defined model of marked Scott domains. As an intermediate semantics we use a model of timed event structures, able to properly capture causality, concurrency, and conflict among events which arise in the presence of time delays of the events.

The paper is organized as follows. In Section 2, the notions and notations concerning the structure and behaviour of timed transition systems with independence are described. Also, an unfolding of timed transition systems with independence is constructed, and it is shown that together with the inclusion functor the unfolding functor defines a coreflection. Section 3 establishes the interrelations in terms of the existence of a coreflection between timed occurrence transition systems with independence and timed event structures. In Section 4, using the equivalence of the categories of timed event structures and marked Scott domains, stated in [9], functors between the categories of timed transition systems with independence and marked Scott domains are constructed to constitute a coreflection. Section 5 provides a direct translation from timed transition systems with independence to marked Scott domains, established in the categorical setting. In section 6, we conclude with a short summary of the discovered relationships.

2 Timed Transition Systems with Independence

In this section, we first describe the basic notions and notations concerning the structure and behaviour of timed transition systems with independence.

We start with untimed case. A *transition system with independence* is a tuple $TI = (S, s^I, L, Tran, I)$, where S is a countable set of *states*, $s^I \in S$ is the *initial state*, L is a countable set of *labels*, $Tran \subseteq S \times L \times S$ is the *transition* relation, and $I \subseteq Tran \times Tran$ is the irreflexive, symmetric *independence relation*,

such that, using $\prec$ to denote the following relation on transitions $(s,a,s') \prec (s'',a,u) \iff \exists (s,b,s''),(s',b,u) \in Tran$ s.t. $(s,a,s')\ I\ (s,b,s'') \wedge (s,a,s')\ I\ (s',b,u) \wedge (s,b,s'')\ I\ (s'',a,u)$, and $\sim$ for the least equivalence relation containing $\prec$, we have:

1. $(s,a,s') \sim (s,a,s'') \Rightarrow s = s''$,
2. $(s,a,s')\ I\ (s,b,s'') \Rightarrow \exists (s',b,u),(s'',a,u) \in Tran\ .\ (s,a,s')\ I\ (s',b,u) \wedge (s,b,s'')\ I\ (s'',a,u)$,
3. $(s,a,s')\ I\ (s',b,u) \Rightarrow \exists (s,b,s''),(s'',a,u) \in Tran\ .\ (s,a,s')\ I\ (s,b,s'') \wedge (s,b,s'')\ I\ (s'',a,u)$,
4. $(s,a,s') \sim (s'',a,u)\ I\ (w,b,w') \Rightarrow (s,a,s')\ I\ (w,b,w')$.

Let $\mathrm{Diam}_{a,b}(s,s',s'',u) \iff \exists (s,a,s'),(s,b,s''),(s',b,u),(s'',a,u) \in Tran\ .\ (s,a,s')\ I\ (s,b,s'') \wedge (s,a,s')\ I\ (s',b,u) \wedge (s,b,s'')\ I\ (s'',a,u)$. We say that the transitions above form an *independence diamond*, and denote the $\sim$-equivalence class of a transition $t \in Tran$ as $[t]$.

A transition system with independence functions by executing transitions from one state to another. A possibly infinite sequence $\pi = t_0\ t_1\ \ldots$ with $t_i = (s_i, a_i, s_{i+1}) \in Tran$ $(i \geq 0)$ is called a *path*. The starting state of π is denoted as $\mathrm{dom}(\pi)$, and the ending state as $\mathrm{cod}(\pi)$ if π is a finite path. A *computation* is a path π such that $\mathrm{dom}(\pi) = s^I$. Let $\mathrm{Comp}(TI)$ $(\mathrm{Comp}^0(TI))$ be the set of all (finite) computations of TI. A transition t is said to be *reachable*, if there exists a computation $\pi \in \mathrm{Comp}^0(TI)$ such that t appears in π. From now on, we consider only those transition systems with independence in which all transitions are reachable. Let $\simeq \subseteq \mathrm{Comp}(TI) \times \mathrm{Comp}(TI)$ be the least equivalence relation such that $\pi_s(s,a,s')(s',b,u)\pi_v \simeq \pi_s(s,b,s'')(s'',a,u)\pi_v \iff \mathrm{Diam}_{a,b}(s,s',s'',u)$, and let $[\pi]$ stand for the $\simeq$-equivalence class of a computation π.

We now incorporate time into the model of transition systems with independence. By analogy with the paper [8], we assume a global, fictitious clock, whose actions advance time by nonuniform amounts and whose value is set to zero at the beginning of system's functioning. All transitions are associated with timing constraints represented as minimal and maximal time delays, and happen "instantaneously", while timing constraints restrict the times at which transitions may be executed. Unlike the paper [8], in our timed model the time domain is changed to the integers, and the maximal delays associated with transitions are always equal to ∞, therefore they are not specified explicitly.

Let $\mathbb{N}$ be the set of non-negative integers.

Definition 1. *A* timed transition system with independence *is a tuple* $TTI = (S, s^I, L, Tran, I, \delta)$*, where* $[\![TTI]\!] = (S, s^I, L, Tran, I)$ *is the underlying transition system with independence, and* $\delta : Tran \to \mathbb{N}$ *is the* delay function *such that* $\delta(t) = \delta(t')$ *for any* $t, t' \in Tran$ *such that* $t \sim t'$.

A *timed computation* of a timed transition system with independence $TTI = (S, s^I, L, Tran, I, \delta)$ is a pair $\Pi = (\pi, \tau) \in (\mathrm{Comp}((S, s^I, L, Tran, I)) \times (\mathbb{N} \cup \{\infty\}))$ with $\tau \geq \delta(\pi) = sup\{\delta(t) \mid t \in \pi\}$. Define $\mathrm{dom}(\Pi) = \mathrm{dom}(\pi)$ and $\mathrm{cod}(\Pi) = \mathrm{cod}(\pi)$. We denote the set of all (finite) timed computations of TTI as

$\mathrm{TComp}(TTI)$ ($\mathrm{TComp}^0(TTI)$), and write $\Pi \simeq_\tau \Pi'$ iff $\pi \simeq \pi'$ and $\tau = \tau'$. It is easy to see that $\simeq_\tau$ is an equivalence relation; the $\simeq_\tau$-equivalence class of a timed computation Π is denoted as $[\Pi]_\tau$. Let $\mathrm{TComp}_{\simeq_\tau}(TTI)$ ($\mathrm{TComp}^0_{\simeq_\tau}(TTI)$) be the sets of $\simeq_\tau$-equivalence classes of all (finite) timed computations of TTI.

For timed transition systems with independence $TTI = (S, s^I, L, Tran, I, \delta)$ and $TTI' = (S', s'^I, L', Tran', I', \delta')$, a *morphism* $h : TTI \to TTI'$ is a pair of mappings $h = (\sigma : S \to S', \lambda : L \to^* L')$[3] such that:

1. $\sigma(s^I) = s'^I$,
2. $(s, a, s') \in Tran \;\Rightarrow\; (\sigma(s), \alpha(a), \sigma(s') \in Tran'$ if $a \in \mathrm{dom}\,\lambda$, and $\sigma(s) = \sigma(s')$, otherwise,
3. $(s, a, s')I(\bar{s}, \bar{a}, \bar{s}')$ and $a, \bar{a} \in \mathrm{dom}\,\lambda \;\Rightarrow\; (\sigma(s), \alpha(a), \sigma(s')I'(\sigma(\bar{s}), \alpha(\bar{a}), \sigma(\bar{s}')$,
4. $\delta'((\sigma(s), \alpha(a), \sigma(s'))) \leq \delta((s, a, s'))$.

Timed transition systems with independence and morphisms between them form a category **TTSI** with unit morphisms $\mathbf{1}_{TTI} = (\mathbf{1}_S, \mathbf{1}_L) : TTI \to TTI$ for any $TTI = (S, s^I, L, Tran, I, \delta)$, and with composition defined in a componentwise manner.

We next aim at unfolding of timed transition systems with independence. To that end, we first define a subclass of timed transition systems with independence that serves as a target of unfolding. After that, we construct an unfolding mapping and show that together with the inclusion functor the unfolding functor defines a coreflection.

Definition 2. *A* timed occurrence transition system with independence $ToTI = (S, s_0, L, Tran, I, \delta)$ *is an acyclic timed transition system with independence such that* $(s'', a, u) \neq (s', b, u) \in Tran \Rightarrow \exists s \in S$ *s.t.* $\mathrm{Diam}_{a,b}(s, s', s'', u)$*, for all* $(s'', a, u), (s', b, u) \in Tran$.

Let **ToTSI** be the full subcategory of the category **TTSI**.

Define an unfolding mapping *ttsi.totsi* : **TTSI** $\to$ **ToTSI** as follows. For a timed transition system with independence $TTI = (S, s^I, L, Tran, I, \delta)$, specify *ttsi.totsi*$(TTI)$ as $(S_{\simeq_\tau}, [(s^I, 0)]_\tau, L, Tran_{\simeq_\tau}, I_{\simeq_\tau}, \delta_{\simeq_\tau})$, where

- $S_{\simeq_\tau} = \{[\Pi = (\pi, \delta(\pi))]_\tau \in \mathrm{TComp}^0_{\simeq_\tau}(TTI)\}$,
- $([\Pi = (\pi, \delta(\pi))]_\tau, a, [\Pi' = (\pi', \delta(\pi'))]_\tau) \in Tran_{\simeq_\tau} \iff \exists t_{\pi,\pi'} = (s, a, s') \in Tran \;.\; \Pi' \simeq_\tau (\pi t_{\pi,\pi'}, \max\{\delta(\pi), \delta(\pi')\})$,
- $([\Pi]_\tau, a, [\Pi']_\tau) I_{\simeq_\tau} ([\bar{\Pi}]_\tau, b, [\bar{\Pi}']_\tau) \iff t_{\pi,\pi'} I t_{\bar{\pi},\bar{\pi}'}$,
- $\delta_{\simeq_\tau}([\Pi]_\tau, a, [\Pi']_\tau) = \delta(t_{\pi,\pi'})$.

Lemma 1. *Given a timed transition system with independence* TTI*,* *ttsi.totsi*(TTI) *is a timed occurrence transition system with independence.*

[3] A partial mapping from a set A into a set B is denoted as $f : A \to^* B$. Let $\mathrm{dom}\, f = \{a \in A \mid f(a) \text{ is defined}\}$. For a subset $A' \subseteq A$, define $fA' = \{f(a') \mid a' \in A' \cap \mathrm{dom}\, f\}$.

In order to demonstrate that the mapping *ttsi.totsi* is adjoint to the inclusion functor **ToTSI** $\hookrightarrow$ **TTSI**, we define a mapping and prove that it is the unit of this adjunction. For a transition system with independence TTI, let $\varepsilon_{TTI} = (\sigma_\varepsilon, 1_L) : ttsi.totsi(TTI) \to TTI$, where $\sigma_\varepsilon([\Pi]_\tau) = \mathrm{cod}(\Pi)$ for all $[\Pi]_\tau \in S_{\simeq_\tau}$. It is easy to see that ε_{TTI} is a morphism of **TTSI**.

Lemma 2 (ε_{TTI} is couniversal). *For any object TTI of* **TTSI**, *any object $ToTI$ of* **ToTSI** *and any morphism $h : ToTI \to TTI$ of* **TTSI**, *there exists a unique morphism $h' : ToTI \to ttsi.totsi(TTI)$ of* **ToTSI** *such that $h = \varepsilon_{TTI} \circ h'$.*

The next theorem presents a categorical characterization of the unfolding.

Theorem 1 ($\hookrightarrow \dashv$ *ttsi.totsi*). *The unfolding mapping ttsi.totsi extends to a functor from* **TTSI** $\to$ **ToTSI** *which is right adjoint to the functor* $\hookrightarrow$: **ToTSI** $\to$ **TTSI**. *Moreover, this adjunction is a coreflection.*

3 Timed Event Structures

In this section we relate timed occurrence transition systems with independence and timed event structures, establishing the close relationships between categories of the models.

We start with the definition of an untimed variant of event structures. An *event structure* is a triple $\mathcal{E} = (E, \leq, \#)$, where E is a countable set of *events*; $\leq \subseteq E \times E$ is a partial order (*the causality relation*) such that $\downarrow e = \{e' \in E \mid e' \leq e\}$ is a finite set for each $e \in E$, $\# \subseteq E \times E$ is the symmetric irreflexive *conflict relation* such that $e \# e' \leq e'' \Rightarrow e \# e''$. A set of events $C \subseteq E$ is said to be a *configuration* of an event structure $\mathcal{E}$ if $\forall e \in C \centerdot \downarrow e \subseteq C$, and $\forall e, e' \in C \centerdot \neg(e \# e')$. We say that events $e, e' \in E$ are *concurrent* and write $e \smile e'$ if $\neg(e \leq e' \vee e' \leq e' \vee e \# e')$. Introduce the concept of a *reflexive conflict* as follows: $e \Cup e' \iff e \# e' \vee e = e'$.

We now recall the definition of timed event structures from [9]. Similarly to the model of timed transition systems with independence, there is a global non-negative integer-valued clock. Each event in the structure is associated with a time delay with respect to the initial time moment; i.e., if an event e is associated with a time delay t, then e may not occur earlier than all the predecessors of the event occur and the clock shows time t. In this case, the event itself occurs instantaneously.

Definition 3. *A* timed event structure *is a tuple $\mathcal{TE} = (E, \leq, \#, \Delta)$, where $(E, \leq, \#)$ is an event structure and $\Delta : E \to \mathbb{N}$ is the* delay function *such that $e' \leq e \Rightarrow \Delta(e') \leq \Delta(e)$.*

A *timed configuration* of $\mathcal{TE}$ is a pair (C, τ), where C is a configuration of $(E, \leq, \#)$ and $\tau \in \mathbb{N} \cup \{\infty\}$ such that $\tau \geq \Delta(C) = sup\{\Delta(e) \mid e \in C\}$. The set of all (finite) timed configurations of a timed event structure $\mathcal{TE}$ is denoted as $\mathrm{TConf}(\mathcal{TE})$ ($\mathrm{TConf}^0(\mathcal{TE})$). We define a transition relation $\longrightarrow$ on the set

$\mathrm{TConf}(\mathcal{TE})$ as follows: $(C,t) \longrightarrow (C',t')$ if $C \subseteq C'$ and $t \leq t'$. Clearly, the relation $\longrightarrow$ specifies a partial order on the set $\mathrm{TConf}(\mathcal{TE})$.

Let $\mathcal{TE} = (E, \leq, \#, \Delta)$ and $\mathcal{TE}' = (E', \leq', \#', \Delta')$ be timed event structures. A partial mapping $\theta : E \to^* E'$ is a *morphism* if $\downarrow\theta(e) \subseteq \theta \downarrow e$; $\theta(e) \Subset \theta(e') \Rightarrow e \Subset e'$, for all $e, e' \in \mathrm{dom}\,\theta$; $\Delta'(\theta(e)) \leq \Delta(e)$, for all $e \in \mathrm{dom}\,\theta$. Timed event structures with their morphisms define a category **TES** with unit morphisms $\mathbf{1}_{TS} = \mathbf{1}_E : TS \to TS$ for all $TS = (E, \leq, \#, \Delta)$ and the composition being a usual composition of partial functions.

We now establish the relationships between the categories of timed event structures and timed occurrence transition systems with independence. For this purpose, we first define a mapping *tpes.totsi* : **TPES** $\to$ **ToTSI** extending the mapping *pes.otsi* from [3] to the timed case. For a timed event structure $\mathcal{TE} = (E, \leq, \#, \Delta)$, let *tpes.totsi*$(\mathcal{TE})$ be $(S_{\mathcal{TE}}, s^I_{\mathcal{TE}}, L_{\mathcal{TE}}, Tran_{\mathcal{TE}}, I_{\mathcal{TE}}, \delta_{\mathcal{TE}})$, where

- $S_{\mathcal{TE}} = \big\{(C, \Delta(C)) \in \mathrm{TConf}^0(\mathcal{TE})\big\}$;
- $s^I_{\mathcal{TE}} = (\varnothing, 0)$;
- $L_{\mathcal{TE}} = E$;
- $\big((C, \Delta(C)), e, (C', \Delta(C'))\big) \in Tran_{\mathcal{TE}} \iff C' \setminus C = \{e\}$;
- $\big((C, \Delta(C)), e, (C', \Delta(C'))\big) I_{\mathcal{TE}} \big((\bar{C}, \Delta(\bar{C})), \bar{e}, (\bar{C}', \Delta(\bar{C}'))\big) \iff e \smile \bar{e}$;
- $\delta_{\mathcal{TE}}((C, \Delta(C)), e, (C', \Delta(C'))) = \Delta(e)$.

It is easy to see that the above definition is correct, i.e. *tpes.totsi* maps timed event structures to timed occurrence transition systems with independence.

Next, we construct a mapping *totsi.tpes* : **ToTSI** $\to$ **TPES**. For a timed occurrence transition system with independence $ToTI = (S, s^I, L, Tran, I, \delta)$, let *totsi.tpes*$(ToTI)$ be $(Tran_\sim, \leq, \#, \Delta)$, where

- $Tran_\sim = \{[t] \mid t \in Tran\}$,
- $[t] < [t'] \iff$
  $\forall (\pi \bar{t}', \tau) \in \mathrm{TComp}^0(ToTI) \,.\, \bar{t}' \sim t' \Rightarrow (\exists \bar{t} \in \pi \,.\, \bar{t} \sim t)$; $\leq = < \cup =$,
- $[t] \# [t'] \iff$
  $\forall (\pi, \tau) \in \mathrm{TComp}^0(ToTI), \forall \bar{t} \in [t], \forall \bar{t}' \in [t'] \,.\, \bar{t} \in \pi \Rightarrow \bar{t}' \notin \pi$,
- $\Delta([t]) = \max\{\delta(t') \mid [t'] \leq [t]\}$.

On morphisms $h = (\sigma, \lambda) : ToTI \to ToTI'$ in **ToTSI**, the mapping *totsi.tpes* acts as follows: *totsi.tpes*$(h) = \theta$, where $\theta([(s, a, s')]) = [(\sigma(s), \lambda(a), \sigma(s')]$, if $a \in \mathrm{dom}\,\lambda$, and $\theta([(s, a, s')])$ is undefined, otherwise.

Proposition 1. *totsi.tpes* : **ToTSI** $\to$ **TPES** *is a functor.*

Finally, we define the unit of the adjunction. For a timed event structure $\mathcal{TE}$, let $\eta_{\mathcal{TE}} : E_{\mathcal{TE}} \to E_{totsi.tpes \circ tpes.totsi(\mathcal{TE})}$ be a mapping such that $\eta_{\mathcal{TE}}(e) = [(C, \Delta(C)), e, (C \cup \{e\}, \Delta(C \cup \{e\}))]$. It is straightforward to show that $\eta_{\mathcal{TE}}$ is an isomorphism in **TPES**. In order to demonstrate the existence of the adjunction, we need to check that $\eta_{\mathcal{TE}}$ is indeed a unit, i.e. it is universal.

Lemma 3 ($\eta_{\mathcal{TE}}$ is universal).
For any object $\mathcal{TE}$ of **TPES**, *any object $ToTI$ of* **ToTSI**, *and any morphism $\theta : \mathcal{TE} \to totsi.tpes(ToTI)$ in* **TPES**, *there exists a unique morphism $h : tpes.totsi(\mathcal{TE}) \to ToTI$ in* **ToTSI** *such that $\theta = totsi.tpes(h) \circ \eta_{\mathcal{TE}}$.*

The next theorem establishes the existence of a coreflection between the categories of timed event structures and timed occurrence transition systems with independence.

Theorem 2 (*tpes.totsi* $\dashv$ *totsi.tpes*)**.** *The map tpes.totsi can be extended to a functor tpes.totsi* : **TPES** $\to$ **ToTSI**, *which is left adjoint to the functor totsi.tpes. Moreover, this adjunction is a coreflection.*

4 Marked Scott Domains

In this section, we extend the established chain of coreflections to marked Scott domains. To that end, we first recall related notions and notations.

Let $(D, \sqsubseteq)$ be a partial order, $d \in D$ and $X \subseteq D$. Then,

- $\uparrow d = \{d' \in D \mid d \sqsubseteq d'\}$ is an *upper cone of element* d, $\downarrow d = \{d' \in D \mid d' \sqsubseteq d\}$ is a *lower cone of element* d,
- X is downward (upward) closed if $\downarrow d \subseteq X$ ($\uparrow d \subseteq X$) for every $d \in X$,
- X is a *compatible set* (denoted as $X\uparrow$), if the following assertion is true: $\exists d \in D \forall x \in X \centerdot x \sqsubseteq d$, i.e., X has an upper bound. If $X = \{x, y\}$, we write $x \uparrow y$ instead of $\{x, y\}\uparrow$. The least upper bound of the set X is denoted as $\bigsqcup X$ (if it exists), and the greatest lower bound is denoted as $\bigsqcap X$ (if it exists). The least upper bound of two elements x and y is denoted as $x \sqcup y$, and the greatest lower bound, as $x \sqcap y$.
- X is a *finitely compatible set* if any finite subset of it $X' \subseteq X$ is compatible.
- X is a *(upper) directed set* if any finite subset of it $X' \subseteq X$ has an upper bound belonging to the set X (thus, X is a finitely compatible and nonempty set).
- $(D, \sqsubseteq)$ is a *directed-complete partial order (dcpo for short)* if every directed subset $X \subseteq D$ has $\bigsqcup X$.
- d is a *finite (compact) element* of a dcpo $(D, \sqsubseteq)$ if, for any directed subset $X \subseteq D$, the following assertion is true: $d \sqsubseteq \bigsqcup X \Rightarrow \exists x \in X \centerdot d \sqsubseteq x$. The set of finite elements is denoted as $C(D)$.
- A dcpo $(D, \sqsubseteq)$ is said to be *algebraic* if, for any $d \in D$, $d = \bigsqcup\{e \sqsubseteq d \mid e \in C(D)\}$. It is said to be ω-algebraic if $C(D)$ is countable.
- $(D, \sqsubseteq)$ is a *consistently complete partial order* (ccpo) if any finitely compatible subset $X \subseteq D$ has $\bigsqcup X$. Clearly, a ccpo has the least element $\bot = \bigsqcup \emptyset$, and is also a dcpo.
- An ω-algebraic ccpo is called a *Scott domain*. A Scott domain $(D, \sqsubseteq)$ is said to be *finitary* if $\downarrow d$ is finite for every $d \in C(D)$.

Describe some properties of Scott domains. An element p of a Scott domain $(D, \sqsubseteq)$ is said to be *prime* if, for any compatible subset $X \subseteq D \centerdot p \sqsubseteq \bigsqcup X \Rightarrow \exists x \in X \centerdot p \sqsubseteq x$. The set of the prime elements is denoted as $P(D)$. A Scott domain $(D, \sqsubseteq)$ is called *prime algebraic* if, for any $d \in D$, $d = \bigsqcup\{p \sqsubseteq d \mid p \in P(D)\}$ and *coherent* if all subsets $X \subseteq D$ satisfying the condition $\forall d', d'' \in X \centerdot d' \uparrow d''$ have $\bigsqcup X$.

Let $(D, \sqsubseteq)$ be a Scott domain and $\prec = \sqsubset \setminus \sqsubset^2$ be a *covering relation*. For elements $d, d' \in D$ such that $d \prec d'$, the pair $[d, d']$ is called a *prime interval*. The set of all prime intervals is denoted as $I(D)$. We write $[c, c'] \leq [d, d']$ if and only if $c = c' \sqcap d \vee d' = c' \sqcup d$. The relation $\sim$ is defined to be a transitive symmetric closure of the relation $\leq$. Note that $\sim$-equivalent prime intervals model one and the same action. Let $[d, d']_\sim$ denote the $\sim$-equivalence class of the prime interval $[d, d']$.

Now we are ready to present the definition of marked Scott domains. Informally, a marked Scott domain is meant to be a prime algebraic, finitary, and coherent Scott domain with the prime intervals modeling two – instantaneous and delayed – types of system actions. The former actions do not require time and are marked by zero, and the latter take one unit of time and are marked by one. It is natural to require that the $\sim$-equivalent prime intervals corresponding to one and the same system action are marked identically.

Definition 4. *A* marked domain *is a triple $(D, \sqsubseteq, m)$, where $(D, \sqsubseteq)$ is a prime algebraic, finitary, and coherent Scott domain and $m : I(D) \longrightarrow \{0, 1\}$ is a marking such that $[c, c'] \sim [d, d'] \Rightarrow m([c, c']) = m([d, d'])$.*

Introduce auxiliary notions and notations. For $d, d' \in D$ and $i \in \{0, 1\}$, we write $d \prec^i d'$, if $d \prec d' \wedge m([d, d']) = i$, and $d \preccurlyeq^i d'$, if $d \prec^i d' \vee d = d'$; $\sqsubseteq^i = (\prec^i)^*$; $\downarrow^i d = \{d' \mid d' \sqsubseteq^i d\}$, and $\uparrow^i d = \{d' \mid d \sqsubseteq^i d'\}$; $P^i(D) = \{p \in P(D) \mid \exists d \in D \,.\, m([d, p]) = i\}$. For a finite element $d \in D$ and a covering chain σ having the form $\bot = d_0 \prec^{k_1} d_1 \cdots d_{n-1} \prec^{k_n} d_n = d$ (the chain is finite as $(D, \sqsubseteq)$ is finitary), define the *norm* of d along σ by $\|d\|_\sigma = \sum_{i=1}^{n} k_i$. Since $(D, \sqsubseteq)$ is a prime algebraic Scott domain and m respects $\sim$, the value of $\|d\|_\sigma$ does not depend on σ. Therefore, we shall use $\|d\|$ to denote the norm of a finite element d. For a non-finite element $d \in D$, its norm is defined as follows: $\|d\| = sup\{\|d'\| \mid d' \in \downarrow d \cap C(D)\}$. A marked domain $(D, \sqsubseteq, m)$ is said to be *linear* if for any $d \in D$ such that $\|d\| < \infty$, $(\uparrow^1 d, \sqsubseteq^1) \cong (\mathbb{N}, \leq)$; *regular* if for any $d, d' \in D$, $d \uparrow d' \Rightarrow \forall d_1 \in \uparrow^1 d,\ \forall d'_1 \in \uparrow^1 d' \,.\, (d_1 \uparrow d'_1)$.

It is not difficult to see that linear regular marked domains, together with the additive stable mappings [10] preserving $\preccurlyeq^0$ and $\prec^1$, form the category **MDom**.

As shown in [9], marked Scott domains are related with timed event structures via a pair of functors *tpes.mdom* : **TPES** $\rightarrow$ **MDom** and *mdom.tpes* : **MDom** $\rightarrow$ **TPES** defined as follows[4].

For a timed event structure $\mathcal{TE} = (E, \leq, \#, \Delta)$, let *tpes.mdom*$(\mathcal{TE})$ be $(\mathrm{TConf}(\mathcal{TE}), \longrightarrow, m_{\mathcal{TE}})$, where

$$m(\big[(C, \tau), (C', \tau')\big]) = \begin{cases} 0, \text{ if } \ C' \setminus C = \{e\} \wedge \tau' = \tau, \\ 1, \text{ if } \ C' = C \wedge \tau' = \tau + 1. \end{cases}$$

For a marked Scott domain $MD = (D, \sqsubseteq, m) \in$ **MDom**, define *mdom.tpes*(MD) to be $(E, \leq, \#, \Delta)$, where $E = P^0(D)$, $p \leq p' \iff p \sqsubseteq p$, $p \# p' \iff p \not\uparrow p'$, and $\Delta(p) = \|p\|$.

[4] We do not specify how *tpes.mdom* and *mdom.tpes* act on morphisms since it is not essential to this paper.

Theorem 3. *[9]. The functors tpes.mdom and mdom.tpes constitute an equivalence between the categories* **TPES** *and* **MDom**.

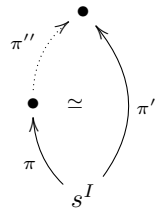
Theorems 1, 2 and 3 yield the following corollary.

Theorem 4. *The functor* $\hookrightarrow \circ tpes.totsi \circ mdom.tpes : \mathbf{MDom} \to \mathbf{TTSI}$ *is left adjoint to the functor* $tpes.mdom \circ totsi.tpes \circ ttsi.totsi : \mathbf{TTSI} \to \mathbf{MDom}$. *Moreover, this adjunction is a coreflection.*

5 Direct Characterization

In this section, we establish some relationships between timed transition systems with independence and marked Scott domains in a direct way.

We start with introducing auxiliary notations. For a transition system with independence $TI = (S, s^I, L, Tran, I)$ and computations $\pi, \pi' \in \mathrm{Comp}^0(TI)$, we write $\pi \trianglelefteq \pi'$ iff there exists a path π'' such that $\pi\pi'' \simeq \pi'$:



For possibly infinite computations $\pi, \pi' \in \mathrm{Comp}(TI)$, let $\pi \trianglelefteq \pi'$ iff for every finite prefix $\bar{\pi}$ of π there exists a finite prefix $\bar{\pi}'$ of π' such that $\bar{\pi} \trianglelefteq \bar{\pi}$. It is straightforward to check that $\trianglelefteq$ is a partial order on $\mathrm{Comp}(TI)$. Specify a partial order on timed computations as follows: $\Pi = (\pi, \tau) \trianglelefteq_\tau \Pi' = (\pi', \tau')$ iff $\pi \trianglelefteq \pi' \wedge \tau \leq \tau'$. Define a partial order $\sqsubseteq$ on the $\simeq_\tau$-equivalence classes of timed computations as follows: $[\Pi]_\tau \sqsubseteq [\Pi']_\tau$ iff $\Pi \trianglelefteq_\tau \Pi'$.

Lemma 4. $(\mathrm{TComp}_{\simeq_\tau}(TTI), \sqsubseteq)$ *is a finitary* ω*-algebraic dcpo. Moreover,*

$$C((\mathrm{TComp}_{\simeq_\tau}(TTI), \sqsubseteq)) = \mathrm{TComp}^0_{\simeq_\tau}(TTI).$$

In order to directly relate timed transition systems with independence and marked Scott domains, we construct a mapping $ttsi.mdom' : \mathbf{TTSI} \to \mathbf{MDom}$. Before doing so, consider a prime interval $\big[[\Pi = (\pi, \tau)]_\tau, [\Pi' = (\pi', \tau')]_\tau\big]$ in $(\mathrm{TComp}_{\simeq_\tau}(TTI), \sqsubseteq)$. It is not difficult to check that either $\pi' \simeq \pi \wedge \tau' = \tau + 1$ or $\pi' \simeq \pi t \wedge \tau' = \tau$ for some transition t. Define a map $m_{TTI} : I((\mathrm{TComp}_{\simeq_\tau}(TTI), \sqsubseteq)) \to \{0, 1\}$ as follows:

$$m_{TTI}(\big[[\Pi]_\tau, [\Pi']_\tau\big]) = \begin{cases} 0, \text{ if } \tau = \tau', \\ 1, \text{ otherwise.} \end{cases}$$

Let $ttsi.mdom'(TTI) = (\mathrm{TComp}_{\simeq_\tau}(TTI), \sqsubseteq, m_{TTI})$, for any timed transition system with independence TTI.

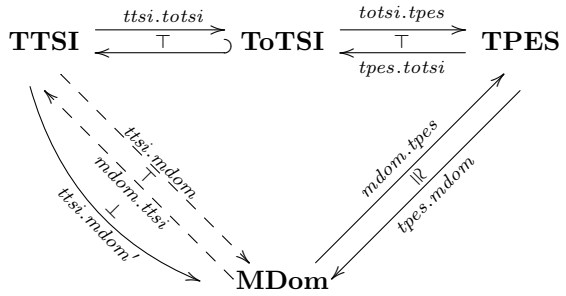
Proposition 2. *ttsi.mdom′ can be extended to a functor ttsi.mdom′* : **TTSI** → **MDom** *isomorphic to ttsi.mdom = tpes.mdom ∘ ottsi.tpes ∘ ttsi.ottsi.*

At last, we are ready to state the fact which is the last main result of this paper and that provides a direct characterisation.

Theorem 5. *ttsi.mdom′ is right adjoint to mdom.ttsi = tpes.mdom∘ottsi.tpes∘ ttsi.ottsi. Moreover, this adjunction is a coreflection.*

6 Conclusion

We have defined and studied a timed extension of a well-known "true concurrent" model of transition systems with independence and have shown that there exists a chain of coreflections between a category of the model and a category of marked Scott domains as well as a direct translation. The diagram below summarises the established relationships:



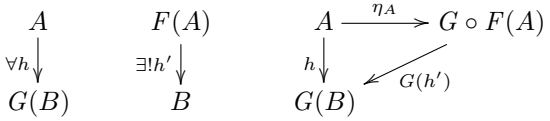
References

1. Bednarczyk, M.: Categories of asynchronous systems. PhD thesis, University of Sussex, UK (1987)
2. Shields, M.: Concurrent Machines. The Computer Journal **28**(5) (1985) 449–465
3. Sassone, V., Nielsen, M., Winskel, G.: Models for concurrency: towards a classification. Theoretical Computer Science **170**(1-2) (1996) 297–348
4. McLane, S.: Categories for the working mathematician. Graduate Texts in Mathematics. Springer, Berlin (1971)
5. Hildebrandt, T., Sassone, V.: Comparing Transition Systems with Independence and Asynchronous Transition Systems. International Conference on Concurrency Theory (1996) 84–97
6. Hildebrandt, T., Sassone, V.: Transition Systems with Independence and Multi-Arcs. BRICS Report Series RS-97-10, BRICS, Department of Computer Science, University of Aarhus, April (1997)
7. Alur, R., Dill, D.: A theory of timed automat. Theoretical computer science **126**(2) (1994) 183–235
8. Henzinger, T., Manna, Z., Pnueli, A.: Timed transition systems. In: Real-Time: Theory in Practice, Springer (1992) 226–251

9. Virbitskaite, I.B., Dubtsov, R.S.: Semantic domains of timed event structures. Programming and Computer Software **34**(3) (2008) 125–137
10. Winskel, G.: Event structures. Lecture Notes in Computer Science **255** (1987) 325–392

Appendix A: Elements of Category Theory

Here we briefly recall notions from category theory [4] important to this paper. Let $G : \mathbf{B} \to \mathbf{A}$ be a functor between categories $\mathbf{A}$ and $\mathbf{B}$, and let, for each object A of $\mathbf{A}$, there exist an object $F(A)$ of $\mathbf{B}$ and a morphism $\eta_A : A \to G \circ F(A)$ in $\mathbf{A}$ that is universal in the following sense: for any morphism $h : A \to G(B)$ in $\mathbf{A}$, where B is an object of $\mathbf{B}$, there exists a unique morphism $h' : F(A) \to B$ in $\mathbf{B}$ such that $G(h') \circ \eta_A = h$; i.e., the following diagram commutes.



In this case, we say that there exists an adjunction from $\mathbf{A}$ to $\mathbf{B}$, and the family of morphisms $\{\eta_A \mid A \in \mathbf{A}\}$ is said to be a unit of this adjunction. Then, F can be extended to a functor by assuming that, for any morphism $h : A \to A'$ in $\mathbf{A}$, $F(h) : F(A) \to F(A')$ is a unique morphism in $\mathbf{B}$ such that $G \circ F(h) \circ \eta_A = \eta_{A'} \circ h$. In this case, F is said to be left adjoint to G (denoted as $F \vdash G$), and G right adjoint to F (denoted as $G \dashv F$). In addition, if η_A is an isomorphism for each A, then the adjunction is called a coreflection. Categories $\mathbf{A}$ and $\mathbf{B}$ are equivalent if F is adjoint to G and both the unit and counit of the adjunction are isomorphisms.

A Multi-agent Approach to Unstructured Data Analysis Based on Domain-specific Onthology *

Natalia O. Garanina, Elena A. Sidorova, and Evgeny V. Bodin

A.P. Ershov Institute of Informatics Systems,
Lavrent'ev av., 6, Novosibirsk 630090, Russia,
{garanina,lena,bodin}@iis.nsk.su

Abstract. The paper presents a multi-agent algorithm implementing semantic analysis of unstructured data based on ontology. In this multi-agent model agents of two kinds interact: the instance agents correspond to meaningful units of the information being retrieved, and the rule agents implement rules of a given ontology. An original solution for the termination detection of this multi-agent algorithm is suggested.

1 Introduction

At present most organizations deal with large quantity of documents, licenses, manuals, emails, business letters, financial and technical reports etc. It is already impossible to process all these documents by hand, without automatic assistance. Ontological knowledge bases are a good solution for storing information from these documents, and automatical completing the ontology is necessary.

The essence of an ontological approach to information retrieval is to use knowledge represented by an ontology for extraction of data interpreted as the ontology instances. For example, semantic-oriented approach to text analysis without complete linguistic analysis can be used for ontological data generation. The standard productional approach to semantic-oriented analysis is to sequentially apply given rules of instance retrieval to data. This process takes a long time and causes such specific problems as information duplication, variability of results, etc. Using the multi-agent approach allows to create good alternatives to the data analysis systems with the sequential architecture. The main feature of the approach is that the system being developed is considered to be a set of autonomous entities (the agents) where the agents have the abilities to interact with the environment and with other agents. By means of this interaction the system works. The traditional benefits of the multi-agent approach is that the operations of the system are parallelized, due to independent agents and their ability to interact with each other, so that some system tasks are solved locally and therefore the result is obtained significantly faster. Besides, our multi-agent approach handles some of the above difficulties of productional approaches.

* The research has been supported by Siberian Branch of Russian Academy of Science (Integration Grant n.15/10 "Mathematical and Methodological Aspects of Intellectual Information Systems").

The proposed approach is in the framework of modern investigations of automatic processing and analyzing huge amount of unstructured data. Multi-agent approach for information retrieval from heterogeneous data source for completing ontology is widespread, in particular, it is used for natural language processing [1, 2, 11, 6] and web processing [3–5]. Agents in these works have different behavior. Usually in web processing, agents are high-level agents that manage rather data flows, using standard algorithm for knowledge retrieval, than data itself. In natural language processing, agents are either associated with conventional linguistic levels (morphological, syntactic, semantic) or targeted to recognize specific linguistic phenomena such as ellipsis, anaphora, parataxis, homonymy. These agents do not use ontological knowledge substantially. Thus they are computing processes which may speed up information retrieval due to their parallel work but they do not affect the retrieval qualitatively.

Unlike all the above works, in our approach we use two kinds of agents, collectively possessing complete information about both the data being investigated and the domain-specific ontology. Agents of one kind can analyze ontological (and linguistic) features. They do not use data directly, but they process information provided by requesting agents of the other kind. The latter agents are the most close to the ones from [10]. In cited paper every agent representing some word from a text has to determine correspondence between its word and an element of a given ontology. The authors do not use special agents for ontological (and linguistic) properties. Instead, they exploit statistical methods of text clustering.

Our variant of ontology-based approach for processing unstructured data contains the following stages. First, a proper domain ontology has to be selected. We suppose that rules for completing the ontology are defined formally. Then initial ontology instances (their classes and some attributes) have to be identified by some preliminary algorithm. Then other instances' attributes are evaluated by the ontology rules using our algorithm.

The idea of multi-agent aspect of the our approach is that a set of different data items is aggregated into an agent considered as an ontology instance. This process is assisted by special support agents corresponding the ontology. First, objects significant for the ontology are recognized preliminary in given data. We call these objects *instance agents*. Belonging to an ontology, the instance agents have attributes. The values of some of these attributes are evaluated as the result of the preliminary analysis. Non-evaluated attributes can be specified as a result of communication of instance agents and the support *rule agents*. In the process of interaction, the agents establish a correspondence between the ontology concepts and the instance units, and thus complete the ontology with specific instances of concepts and relationships.

This paper presents a multi-agent algorithm for arbitrary unstructured data processing. This algorithm improves and generalizes the algorithm for information retrieval from natural language text suggested in [7]. We estimate the time upper bound of the algorithm and prove the properties of termination and correctness of the termination controller agent.

The rest of the paper is organized as follows. The next section 2 describes the main agents in our systems and gives a simple example. The following section 3 presents protocols for the instance, rule and controller agents and sketches some properties of the systems. Finally, we conclude in the last section 4 with a discussion of further research topics.

2 Agent Model

Outline of the approach and multi-agent system follows. There is an ontology, a set of rules for completing it, and a finite set of data to extract the information for the ontology. The preliminary phase partially assigns values to *instance* agents attributes. The *rule* agents implementing the ontology rules (the rules depend on the ontology only, but not on the data), according to data received from various instance agents, generate new attribute values of the instances, send the obtained result to all agents interested in it, or generate new instance agents. Eventually, the instance agents assign values to all their attributes that can be evaluated with the information from the data, and the system stops. A special *controller* agent keeps track of system stopping.

Let the result of data pre-processing be the set IA of instance agents, where each $I \in IA$ is a tuple $I = (id; Cl; RO_0; a_1(RI_1; RO_1), ..., a_k(RI_k; RO_k))$, where

- id is a unique agent identifier;
- Cl is an ontological class of the agent;
- RO_0 is set of rule agents that use this instance agent as an argument;
- for each $i \in [1..k]$, a_i is the attribute of the agent, which value is determined by some rule agent from RI_i, every rule in set of rule agents RO_i requires the value of attribute a_i to get the result; let us denote the set of rule agents for incoming values as $RI = \cup_{i=1..k} RI_i$, the set of rule agents for outcoming values as $RO = \cup_{i=1..k} RO_i$.

The values of attributes of an instance agent are usually only partially determined before the algorithm starts. When the algorithm terminates, the initially unvalued attributes should be provided values with help of rule agents.

Let us define the set of rule agents RA, where each $R \in RA$ is a tuple $R = (id; arg_1(Cl_1), \ldots, arg_s(Cl_s); make\_res(args), a_{res})$, where

- id is a unique agent identifier;
- for each $i \in [1..s]$: arg_i is a set of argument values determined by the corresponding instance agent from ontological class Cl_i; let us denote the set of vectors of argument values as $args$, where each value is provided with the identifier of the defining instance agent, the set of these agents is $args.Ag$; let us consider the argument values vector non-empty, if all its values are non-empty;
- $make\_res(args)$ is a function for computing the result from argument vector $args$;
- a_{res} is the result of function $make\_res(args)$ which can be

- null, if the argument vector is inconsistent;
- a new value of some attribute[1] for instance agents;
- a new instance agent (there should not be an agent with the same attribute values in the system).

As a simple example let us consider the following multi-agent system for natural language text processing. Let the given ontology includes classes $Person$, $Organization$, and relation $Employee$. The corresponding instance agents have the following form:

- $A_1 = (id; Person; \{CWork, CPersonDeg, \ldots\};$
  $surname(\{CPerson, CPersonIni\}; \emptyset), first\_name(\{CPerson\}; \emptyset),$
  $degree(\{CPersonDeg\}; \emptyset), ...; Employee(\{CWork, CWorkPos\}).$
  $Person$ has the following attributes: $surname$, $first\ name$, (academic) $degree$ which can be used and evaluated by the corresponding rule agents $CPerson$, $CPersonIni$, $CPersonDeg$.
- $A_2 = (id; Organization; \{CWork, CPersonDeg, \ldots\};$
  $name(COrg; \emptyset), type(COrg, COrgType; CWork), \ldots;$
  $Employee(CWork, CWorkPos);$
  $Organization$ has attributes $name$ and $type$, and the corresponding input-output rule agents $COrg$ and $COrgType$.
- $A_3 = (id, Employee; \{CWork, CWorkPos\};$
  $arg_1(CWork; \emptyset), arg_2(CWork; \emptyset), pos(CWorkPos; \emptyset)).$
  Relation $Employee$ can be evaluated by rule agent $CWork$ directly connecting $Person$ and $Organization$, or by rule agent $CWorkPos$ which connects them using a position.

As an example of an ontology rule agent let us consider agent $CWork$:

$CWork = (id, arg_1(A_1), arg_2(A_2);$
$\{Sentence(\{arg_1, arg_2\}), BracketSegment(\{arg_2\}),$
$Preposition(arg_1, arg_2), Contact(arg_1, arg_2)\};$
$Employee.arg_1 = CWork.arg_1, Employee.arg_2 = CWork.arg_2;$
$A3 = newEmployee()).$

This agent recognizes sentences where an organization is enclosed in brackets after a person. For example:

> Academician Genrikh Aleksandrovich Tolstikov (Novosibirsk Institute of Organic Chemistry) is a prominent chemist, recognized authority in synthetic organic chemistry.

The following evaluation of attributes of the above agents is the result of analysis of the given text fragment:

$A_1 = Person(id = 1, surname = Tolstikov, first\_name = Genrikh,$
$degree = Academician);$
$A_2 = Organization(id = 2, name = Institute\ of\ Organic\ Chemistry,$

[1] This attribute is defined a priory.

$type = Institute$);
$A_3 = Employee(id = 3, arg_1 = A_1, arg_2 = A_2, pos = \emptyset)$.

In the next section, the algorithms of the instance, rule and controller agents are described in pseudocode.

3 Multi-agent Algorithm for Data Analysis

Let $IA = \{I_1, \ldots, I_n, \ldots\}$ be an instance agents set, and $RA = \{R_1, \ldots, R_m\}$, be a rule agents set. The result of executing of the following algorithm is data analysis, when the instance agents determine the possible values of their attributes. Let `Ii` be a protocol of actions of instance agent I_i, and `Rj`, be the protocol of actions of rule agent R_j, `C` be the protocol of actions of an agent-controller C. Then the multi-agent data analysis algorithm MDA can be presented in pseudocode as follows:

```
MDA::
  begin
   parallel {I1} ...{In} ...{R1} ...{Rm} {C}
  end.
```

Here we assume that the `parallel` operator means that all execution flows (threads) in the set of braces are working in parallel. That is, all agents act in parallel until either all attributes of the instance agents are evaluated or it happens that none of the rule agent can proceed. These events are determined by the controller agent. The system is dynamic because rule agents can create new instance agents. Let N be the maximal number of instance agents that can be obtained from a given data.

The agents are connected by duplex channels. The controller agent is connected with all agents, and every instance agent is connected with several rule agents (and vise versa). Messages are transmitted asynchronously and stored in FIFO channels until being read. The messages are transmitted in a fast and reliable medium.

We consider an agent *active* iff it does not complete its work (is not at the label "`end`" of the algorithms below) and either it processes some message or its queue of input messages is not empty. Otherwise, the agent is *passive*. We say that a multi-agent system *terminates* iff every system agent (possibly) except the agent-controller is passive.

In the agent protocols below the function `get_head(queue)` removes the first element from the queue and returns that element.

3.1 Protocol of Instance Agents

Let us comment a notation of the instance agent protocol. A message for an instance agent has two fields: `name` of sender (`name` $\in [1..m] \cup C$)[2] and a_i with value of attribute i. The pseudocode of the protocol follows.

[2] The agent receives messages only from the controller agent and from the rule agents.

Protocol of instance agents.

```
I::
  C: Controller Agent;
  R, R_ij: Rule Agent;
  RI, RO: set of Rule Agents;
  data_wait: set of Rule Agents = ∅;
  a_i: Attribute;
  mess: message;
  In: queue of incoming messages;
begin
1.  send |RI ∪ RO| + 1 to C;
2.  forall R ∈ RI ∪ RO send evaluated_data to R;
3.  forall a_i ∈ Atr forall R_ij ∈ RI_i {
4.     send request(a_i) to R_ij; add (ij) to data_wait;}
5.  send −1 to C;
6.  while (true){
7.   if In ≠ ∅ then {
8.      mess = get_head(In);
9.      if mess.name = C then break;
10.     if mess.name ∈ RI_i then {
11.       if a_i = ∅ then {
12.        upd(a_i);
13.        forall R_ij ∈ RI_i {
14.         send cancel(a_i) to R_ij; remove (ij) from data_wait;}
15.        forall R_ij ∈ RO_i
16.         send data(a_i) to R_ij;
17.        send |RO_i| − 1 to C;}
18.        if a_i ≠ ∅ then send −1 to C;} }
19.     if data_wait = ∅ then break;}
end.
```

Let us informally describe the protocol of an instance agent. First, the agent (1) notifies the controller agent that it started working and the number of rule agents that will process its data (line 1), (2) sends the available data (line 2), (3) sends the requests for evaluating and adds the corresponding rule agents to its waiting list (lines 3-4), and then (4) tells the controller agents that it is now passive. From the beginning of the work of the agent, its channel is open for incoming messages. As soon as a message arrives, it begins processing it (line 8). If it is from the controller agent, the agent terminates (line 9). If it is from a rule agent (line 10) then if the corresponding attribute is empty (line 11) the agent evaluates it with the obtained data (line 12) and notifies other rule agents related to this attribute that a value of this attribute is no more required from them, and then the instance agent deletes these agents from its waiting list (lines 13-14). Then the obtained attribute value is sent to those agents that require it (lines 15-16), and the controller agents is notified about the agent finishing its work and about the number of rule agents that will process sent attribute

value (line 17). If the message contains the value of an attribute that is already evaluated, the agent does not handle it and notifies the controller agent about it (line 18). If it turns out that all attributes are evaluated, the agent finishes its work (line 19).

Let us estimate the time upper bound of the instance agent protocol. In the first phase of its activities (line 2) the instance agent sends the evaluated data to all rules agents interested in this data. The complexity of this phase $C_1^{IA} = O(|RI| + |RO|) = O(m)$. Sending the activation messages to rule agent from RI (lines 3-4) is estimated as $C_2^{IA} = O(k \times m)$ and the size of the queue of incoming messages In is the same. The agent processes the received data (lines 8-19). It takes time $C_4^{IA} = O(|In| \times (|RI| + |RO|)) = O(k \times m^2)$. Thus, the upper bound of the protocol actions of each instance agent is $C^{IA} = O(k \times m^2)$.

3.2 Protocol of Rule Agents

In the algorithm of the rule agent's actions protocol, the following functions and notation are used. The rule agents receive messages only from instance agents and from the controller agent. The messages have (1) the `name` of the sender, (2) the `type` $\in$ {`data, request, cancel`} that means that it has received an attribute, a request for a result, or a cancelation request, respectively, and (3) the *value* of the attribute. The function `make_arg`(a, I) creates vectors of arguments with received values of attributes at the positions corresponding to ontology classes. The function `make_res`$(args)$ creates the output result: (1) the values of attributes that have sent a request to the rule agent, (2) a new instance agent which starts working immediately, or (3) the null result in a case of inconsistency of the argument vector. The pseudocode of the protocol follows.

Protocol of rule agents.

```
R::
  C: Controller Agent;
  I: Instance Agent;
  a: Attribute;
  args: vector of Argument = ∅;
  Arg: queue of vector of Argument = ∅; // set of tuples
  res_send: set of Instance Agents = ∅;
  In: queue of incoming messages;
begin
1.  parallel
2.   { while (true) {
3.      if In ≠ ∅ then {
4.        mess = get_head(In);
5.        if mess.name=C then goto end;
6.        if mess.name=I then {
7.            if mess.type=request then add I to res_send;
8.            if mess.type=cancel then remove I from res_send;
9.            if mess.type=data then {
10.               a = mess.val; Arg = Arg ∪ make_arg(a, I);
```

```
11.              send −1 to C; }}}}}
12.  { while (true) {
13.      if Arg ≠ ∅ then {
14.         send 1 to C;
15.         args = get_head(Arg);
16.         a_res = make_res(args);
17.         if a_res.type = attr then {
18.            forall I ∈ args.Ag ∩ res_send{
19.               send a_res to I; remove I from res_send;}
20.            send |args.Ag ∩ res_send| − 1 to C;}
21.         if a_res.type = new_agent then send a_res.val to a_res.name;
22.         if a_res.type = null then send −1 to C;} } }
end.
```

Let us informally describe the protocol of a rule agent. The agent can perform in parallel both processing of incoming messages(lines 2-10) and the generating of the outcome (lines 12-21). If it has received a message from the controller agent, it finishes the work (line 5). If the agent receives a request from the agent I for a result (line 7), it adds I to the recipients list; and it removes I from this list if it was the cancelation request (line 8). If it receives a value of the attribute a from the agent I, then using the procedure *make_arg* it tries to create a vector of arguments (set of vectors) (line 10). In such vectors the received value of the attribute is one of the elements and other elements are values of attributes received earlier. Then the agent tells the controller agent about becoming passive (line 11). If the vector (or the set of vectors) is formed, the agent immediately begins to process it/them (line 13). The result of processing is obtained using the function *make_res*. It may be (1) an attribute which is later sent to those agents that have requested it (lines 16-18), then the controller agent is informed about the number of the agents to process the data and about this agent has completed processing the vector of arguments, (2) a new instance agent, that starts working immediately as soon as it gets the attribute values from the rule agent, (3) no result, due to the vector of arguments is not consistent, and the controller agent is notified that the argument vector processing is finished.

Let us estimate the time upper bound of the rule agent's protocol. The time complexity depends on the time bounds of the parallel actions of the rule agents. Let Ag be the set of agents have sent attributes and Arg be the set of arguments of the rule agent. The complexity of requests and cancels is $C_2^{RA} = O(|Ag|) = O(N)$ (lines 7-8). Retrieving and storing data from the instance agents (lines 9-11) is a very time-consuming process with the estimate $C_1^{RA} = O(|Ag|^{|Arg|}) = O(N^s)$, since the obtained data generate a set of vectors of the argument values. The complexity of parallel data processing (lines 13-22) is $C_3^{RA} = O(C_1^{RA} \times (||make\_res|| + |Ag|) = O(N^s \times (s+N))$, where $||make\_res||$ is the time complexity of the function, which is linear with respect to the size of the argument. Thus, the overall time upper bound of the actions of each rule agent is $C^{RA} = O(C_3^{RA}) = O(N^s \times (s+N))$.

3.3 Protocol of the Controller Agent

A special agent-controller handles the Distributed Termination Detection problem [8]. We suggest an algorithm for the problem which fits to our multi-agent system more than known termination detection algorithms, the credit/recovery algorithms in particular [9, 12]. The main feature of this agent-controller is to sequentially calculate other agents' activities by using variable Act. Instance and rule agents send information about their activities to the agent-controller. After system termination the agent informs others about this fact.

Protocol of agent-controller C.

```
C ::
  Act, num: integer;
  messages: queue of integer;
begin
1.  Act = 0;
2.  while(true){
3.   if messages ≠ ∅ then { num =get_head(messages); Act=Act+num;}
4.   if messages = ∅ and Act = 0 then break; }
5.  send STOP to all;
end.
```

Let us estimate the time upper bound of the controller agent's protocol. The size of the queue of incoming messages for the controller agent C^{CA} is less then $N + \sum_{i\in[1..N]} |k_i| + \sum_{j\in[1..m]}(N + N_j^s)$, where k_i is the number of attributes of instance agent i and s_j is the number of attributes of instance agent j.

3.4 MDA Protocol Properties

The time complexity of the multi-agent analysis algorithm MDA follows from the above estimations: $C^{MDA} = O(max\{C_1^{IA}, ..., C_N^{IA}, C_1^{RA}, ..., C_m^{RA}, C^{CA}\})$, where C_i^{IA} and C_j^{RA} are the complexities of the protocols of the instance and rule agents, respectively, for all $i \in [1..N]$, $j \in [1..m]$.

Correctness (completeness and soundness)[3] of information retrieval algorithms is rather a notion of data analysis theory than the theory of multi-agent algorithms, thus it is out of the scope of this paper. But this multi-agent algorithm has some properties to be proved.

Proposition 1. *Multi-agent system* **MDA** *terminates and the agent-controller determines the termination moment correctly.*

Sketch of the proof. First, an analyzed data contains a finite number of information objects for a given ontology. Hence the number of corresponding agents and their attributes is finite. Hence (1) every instance agent determines values of all its attributes and goes to a passive state or (2) some attributes can not be evaluated because there is no appropriate information in the data and after

[3] Completeness means that all relevant information has been retrieved from data. Soundness mans that this information has been retrieved correctly.

determining evaluable attributes an instance agents goes to a passive state also. Every rule agent (1) gets enough information from instance agents to process received data and goes to a passive state after that or (2) goes to a passive state after receiving messages and never processes data. After processing data, the generation of new instance agents does not duplicate agents. Hence, there is no infinite loop because the number of information objects in the data is finite.

Second statement of the proposition follows from the fact that the value of variable *Act* becomes 0 no earlier than the termination moment. Let $active(t)$ be the number of active agents. For every time moment t the following holds: $\sum_{i \leq |mess(t)|} mess(i,t) + Act = active(t)$, because agents influence (1) increase of *Act* when after their local termination they send to the controller the number of meaningful messages sent to instance/rule agents (lines 1,17/20), and (2) decrease of *Act* when they informs about their passive state (lines 5,18/11,22).■

4 Conclusion

The proposed approach aims at taking advantage of the agent-based approach to knowledge representation and processing. Thus, using the agent-based technology allows to avoid unnecessary information retrieval, since at any given time, only information required for an agent is being searched for. Furthermore, due to the agents working in parallel, the speed of data processing significantly increases.

Note that this paper presents only a basic formal model of agents' interaction that implements a simplified model of data analysis, which does not yet take into account specific problems related to ambiguity of input data. For example, let us consider a case of data ambiguity when the different ontology instance agents correspond to the same data ("toast" as "fried bread" and as "a tribute or proposal of health"). In order to handle such ambiguities competitive instance agents acquire points which characterize their connections with other instance agents. These connections are defined by agents' attributes that could be the *linked* agents themselves or their attributes. The more links some agent has and the more points its linked agents have, it becomes more probable that this agent is the most accurate data based instance of a given ontology.

These problems can be solved by increasing the expressive power of the proposed agent-based models by giving the agent the ability to work cooperatively, to compete (as above), to keep the history of its creation and development, etc.

Acknowledgments. We would like to thank Dr. I.S. Anureev for discussions.

References

1. Aref, M.M. *A Multi-Agent System for Natural Language Understanding* International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2003, 36

2. Ariadne Maria B.R. Carvalho, Daniel S. de Paiva, Jaime S. Sichman, João Luís T. da Silva, Raul S. Wazlawick & Vera Lúcia S. de Lima *Multi-Agent Systems for Natural Language Processing* In Francisco J. Garijo & Cristian Lemaitre (eds.), Multi Agent Systems Models Architecture and Appications, Proceedings of the II Iberoamerican Workshop on D.A.I. and M.A.S(Toledo, Spain, October 1-2 1998), pp. 61-69.
3. Banares-Alcantara R., Jimenez R., Aldea L. *Multi-agent systems for ontology-based information retrieval* // European Symposium on Computer-Aided Chemical Engineering-15 (ESCAPE-15),2005, Barcelona, Espaa
4. Cheng X., Xie Y., Yang T. *Study of Multi-Agent Information Retrieval Model in Semantic Web* // In Proc. of the 2008 International Workshop on Education Technology and Training and 2008 International Workshop on Geoscience and Remote Sensing (ETTANDGRS'08), 2008, Vol. 02, P. 636-639.
5. Clark K.L., Lazarou V.S. *A Multi-Agent System for Distributed Information Retrieval on the World Wide Web* // In Proc. of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises (WET-ICE '97), 1997, P. 87-93.
6. Danilo Fum, Giovanni Guida, Carlo Tasso *A Distributed Multi-Agent Architecture for Natural Language Processing* // In Proc. of the 12th conference on Computational linguistics (COLING '88), 1988, Vol. 2, P. 812-814.
7. Garanina N., Sidorova E., Zagorulko Yu. *Multi-agent algorithm of text analysis based on domain-specific onthology.* // Proc. of The 13th Russian Conference on Artificial Intelligence (CAI-2012), October 16-20, 2012, Belgorod, Vol.1, P. 219-226. (In Russian)
8. Matocha J., Camp T. *A taxonomy of distributed termination detection algorithms* // The Journal of Systems and Software, 1998, Vol. 43, P. 207-221
9. Mattern, F. *Global quiescence detection based on credit distribution and recovery* // Inform. Process. Lett. 30 (4), 1989, P. 195-200.
10. Minakov I., Rzevski G., Skobelev P., Volman S. *Creating Contract Templates for Car Insurance Using Multi-agent Based Text Understanding and Clustering*// In Proc. Holonic and Multi-Agent Systems for Manufacturing, Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2007, Regensburg, Germany, September 3-5, 2007. Springer, Lecture Notes in Computer Science, 2007, Vol. 4659, P. 361-370.
11. Cássia Trojhan dos Santos, Paulo Quaresma, Irene Rodrigues, Renata Vieira *A Multi-Agent Approach to Question Answering* // In Renata Vieira, Paulo Quaresma, Maria da Graça Volpes Nunes, Nuno J. Mamede, Cláudia Oliveira & Maria Carmelita Dias (eds.), Computational Processing of the Portuguese Language: 7th International Workshop, PROPOR 2006. Itatiaia, Brazil, May 2006 (PROPOR'2006) LNAI 3960, 13-17 de Maio de 2006, Berlin/Heidelberg: Springer Verlag, pp. 131-139.
12. Rokusawa, K., Iciyoshi, N., Chikayama, T., Nakashima, H. *An ecient termination detection and abortion algorithm for distributed processing systems* // In: Proceedings of the International Conference on Parallel Processing, pp. 18-22.
13. Zagorulko Yu.A., Sidorova E.A. *Document analysis technology in information systems for supporting research and commercial activities* // Optoelectronics, Instrumentation and Data Processing, 2009. Volume 45, Number 6. -pp. 520-525.

An Explicit Formula for Sorting and its Application to Sorting in Lattices

Jens Gerlach

Fraunhofer FOKUS
jens.gerlach@fokus.fraunhofer.de

Abstract In a totally ordered set the notion of sorting a finite sequence is defined through the existence of a suitable permutation of the sequence's indices. A drawback of this definition is that it only implicitly expresses how the elements of a sequence are related to those of its sorted counterpart. To alleviate this situation we prove a simple formula that explicitly describes how the kth element of a sorted sequence can be computed from the elements of the original sequence. As this formula relies only on the minimum and maximum operations we use it to define the notion of sorting for lattices. A major difference of sorting in lattices is that it does *not* guarantee that sequence elements are only rearranged. To the contrary, sorting in general lattices may introduce new values into a sequence or completely remove values from it. We can show, however, that other fundamental properties that are associated with sorting are preserved. Furthermore, we address the problem that the direct application of our explicit formula for sorting leads to an algorithm with exponential complexity. We present therefore for distributive lattices a recursive formulation to compute the sort of a sequence. This alternative formulation, which is inspired by the identity $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ that underlies Pascal's triangle, allows for sorting in lattices with quadratic complexity and is in fact a generalization of *insertion sort* for lattices.

1 Introduction

In this paper we present the results of two preprints [1,2] where we outline basic principles of a theory of sorting in lattices.

Sorting a sequence in a total order $(X, \leq)$ is typically defined through the existence of a suitable permutation (cf. [3, p. 4]). There exists for each sequence x of length n in a totally ordered set a permutation φ of $[1, n] = \{1, \ldots, n\}$ such that $x \circ \varphi$ is a increasing sequence. If x is injective, then φ is uniquely determined, and vice versa. However, regardless whether there is exactly one permutation, the rearrangement $x^{\uparrow} = x \circ \varphi$ is uniquely determined and we thus refer to it as the *increasing sort of* x.

Sorting defines a map $x \mapsto x^{\uparrow}$ from X^n to the subset of increasing sequences. This map has several interesting properties. First of all, it is idempotent

$$\left(x^{\uparrow}\right)^{\uparrow} = x^{\uparrow} \tag{1}$$

and thus a projection. Secondly, for each permutation ψ of $[1, n]$ we have

$$(x \circ \psi)^{\uparrow} = x^{\uparrow} \tag{2}$$

The definition of sorting through the existence of a suitable permutation only provides an implicit relationship between the elements of x and $x^{\uparrow}$. However, sometimes we prefer explicit relationships.

If, for example, someone asked whether there is for the numbers a and b and the exponent n a general relationship between the value $(a + b)^n$ and the powers a^n and b^n, then the (obvious) answer is that this relationship is captured by the Binomial Theorem

$$(a + b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{3}$$

which also shows that other powers of a and b are involved.

When looking for an explicit relationship between the elements of x and the elements of its increasingly sorted counterpart $x^{\uparrow} = \left(x_1^{\uparrow}, \ldots, x_n^{\uparrow}\right)$, one can provide an easy answer for the first and last elements of $x^{\uparrow}$. In fact, we know that $x_1^{\uparrow}$ is the least element of $\{x_1, \ldots, x_n\}$

$$x_1^{\uparrow} = x_1 \wedge \ldots \wedge x_n = \bigwedge_{k=1}^{n} x_k, \tag{4}$$

whereas $x_n^{\uparrow}$ is the greatest element of x

$$x_n^{\uparrow} = x_1 \vee \ldots \vee x_n = \bigvee_{k=1}^{n} x_k. \tag{5}$$

In Section 2 we prove Identity (7) that explicitly states how the elements $x_1^{\uparrow}, \ldots, x_n^{\uparrow}$ are related to $x_1, \ldots, x_n$. This formula only uses the minimum and maximum operations on finite sets. Based on this observation, we define in Section 3 the notion of *sorting of sequences in a lattice* through simply replacing the minimum/maximum operations by the infimum/supremum operations, respectively. We also show that sorting in lattices in general not just reorders the elements of a sequence but really changes them. However, we are able to prove that our definition satisfies various properties that are associated with sorting.

The direct application of Identity (7) leads to an algorithm with exponential complexity (cf. Section 4). In order to address this problem, we prove the *recursive* Identity (19) for the case of *bounded distributive lattices*. This identity is closely related to the well-known fact that the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

can be efficiently computed through the recursion

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

which underlies Pascal's triangle.

Furthermore, we prove that a lattice, in which the recursive Identity (19) holds, is necessarily distributive. The main advantage of our recursive identity is that it allows for an algorithm for sorting in lattices with quadratic complexity. In fact, this algorithm is a generalization of *insertion sort* for lattices (cf. Section 5).

2 A formula for sorting

Let $(X, \leq)$ be a totally ordered set, then each nonempty finite subset A of X contains a *least* and a *greatest* element [4, R. 6.5]. We also speak of the minimum and maximum of A and refer to these special elements as $\bigwedge A$ and $\bigvee A$, respectively. The following inequalities hold for all $a \in A$

$$\bigwedge A \leq a \leq \bigvee A \tag{6}$$

For $A = \{x, y\}$ we use the notation $x \wedge y$ and $x \vee y$ to denote the minimum and maximum of x and y, respectively.
The main results of this paper depend on a particular family of finite sets.

Definition 1. *For $k \in [1, n]$ we denote with $\mathbb{N}\binom{n}{k} := \left\{A \subset [1, n] \;\middle|\; |A| = k\right\}$ the set of subsets of $[1, n]$ that contain exactly k elements. The set $\mathbb{N}\binom{n}{k}$ consists of $\binom{n}{k}$ elements.*

Proposition 1. *Let $(x_1, \ldots, x_n)$ be a sequence in a totally ordered set, then the following identity holds for the elements of the sequence $\left(x_1^{\uparrow}, \ldots, x_n^{\uparrow}\right)$*

$$x_k^{\uparrow} = \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i. \tag{7}$$

Before we prove Proposition 1 we introduce an abbreviation for the right hand side of Identity (7). For a sequence x of length n we define for $1 \leq k \leq n$

$$x_k^{\Delta} := \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i. \tag{8}$$

With this notation Proposition 1 reads

$$x^{\uparrow} = x^{\Delta}. \tag{9}$$

We remark that because $(X, \leq)$ is a total order, we know that each element of x^{Δ} is also an element of x. When applying Identity (8) it is sometimes convenient to use a slightly more explicit way to write the elements of x^{Δ}.

$$x_k^{\Delta} = \bigwedge_{1 \leq i_1 < \ldots < i_k \leq n} x_{i_1} \vee \ldots \vee x_{i_k} \tag{10}$$

We see then that x_1^{Δ} is the least element of x and thus equals $x_1^{\uparrow}$ (cf. Identity (4)), whereas x_n^{Δ} is the greatest element of x and thus equals $x_n^{\uparrow}$ (cf. Identity (5)). This means that Identity (7) is satisfied for $k = 1$ and $k = n$.

Lemma 1. *If x is a sequence of length n in a totally ordered set $(X, \leq)$, then x^{Δ} is a increasing sequence.*

Proof. Let $1 \leq k < n$ and I be an arbitrary subset of $[1, n]$ with $k+1$ elements. If J is a subset of I with k elements, then we have by Inequality (6) and $J \subset I$

$$x_k^{\vartriangle} = \bigwedge_{L \in \mathbb{N}\binom{n}{k}} \bigvee_{l \in L} x_l \leq \bigvee_{j \in J} x_j \leq \bigvee_{i \in I} x_i.$$

Since I is an arbitrary set of $k+1$ elements we obtain from here

$$x_k^{\vartriangle} \leq \bigwedge_{I \in \mathbb{N}\binom{n}{k+1}} \bigvee_{i \in I} x_i = x_{k+1}^{\vartriangle},$$

which shows that $x^{\vartriangle}$ is increasing. □

Note that in the proof of Lemma 1 we have only used the fact that the minimum of a set is a *lower bound* for all elements of that set (cf. Inequality (6)).

Proof (Proposition 1). We will show that for each k with $1 \leq k \leq n$ both $x_k^{\vartriangle} \leq x_k^{\uparrow}$ and $x_k^{\uparrow} \leq x_k^{\vartriangle}$ hold. Let φ be a permutation of $[1, n]$ with

$$x^{\uparrow} = x \circ \varphi \tag{11}$$

and let $J \subset [1, n]$ be the subset for which

$$J = \varphi([1, k]) \tag{12}$$

holds. From the fact that J contains exactly k elements we conclude

$$\begin{aligned}
x_k^{\vartriangle} = \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i &\leq \bigvee_{j \in J} x_j && \text{by Inequality (6)} \\
&= \bigvee_{j \in J} x^{\uparrow}\left(\varphi^{-1}(j)\right) && \text{by Identity (11)} \\
&= \bigvee_{i \in [1,k]} x_i^{\uparrow} && \text{by Identity (12)} \\
&= x_k^{\uparrow} && \text{by monotonicity of } x^{\uparrow}.
\end{aligned}$$

This finishes the first part of the proof.
Conversely, we conclude from the fact that $(X, \leq)$ is a total order and Identity (11) that there exists a subset B of $[1, n]$ with exactly k elements such that

$$x_k^{\vartriangle} = \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i = \bigvee_{i \in B} x_i = \bigvee_{i \in B} x^{\uparrow}\left(\varphi^{-1}(i)\right) = \bigvee_{j \in \varphi^{-1}(B)} x_j^{\uparrow}$$

holds. Since $x^{\uparrow}$ is increasing we have $\bigvee_{j \in \varphi^{-1}(B)} x_j^{\uparrow} = x_m^{\uparrow}$ where $m = \bigvee(\varphi^{-1}(B))$ is the greatest element of $\varphi^{-1}(B)$. We have, thus,

$$x_k^{\vartriangle} = x_m^{\uparrow}. \tag{13}$$

However, since $\bigvee(\varphi^{-1}(B))$ is a subset of $[1, n]$ that contains exactly k elements we obtain $k \leq m$. Since $x^{\uparrow}$ is increasing we conclude $x_k^{\uparrow} \leq x_m^{\uparrow}$. This inequality and Identity (13) imply $x_k^{\uparrow} \leq x_k^{\vartriangle}$, which completes the proof. □

3 Sorting in lattices

Let $(X, \leq)$ be a partially ordered set that is also a *lattice* $(X, \wedge, \vee)$, then for each $x, y \in X$ there exists the infimum $x \wedge y$ and the supremum $x \vee y$ (cf. [5, Chapter 3]). These operations are commutative and associative and they satisfy for all $x, y \in X$ the so-called absorption properties $x \vee (x \wedge y) = x$ and $x \wedge (x \vee y) = x$. If $(X, \leq)$ is a *total order*, then $\wedge$ and $\vee$ are the minimum and maximum operations of Section 2.

In a lattice, the infimum and supremum exist for every finite subset A and are denoted by $\bigwedge A$ and $\bigvee A$, respectively (cf. [5, p. 49]). We therefore know that for a sequence x of length n the value

$$x_k^{\Delta} = \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i$$

from Identity (8) is well-defined in a lattice. This motivates the following definition.

Definition 2. *If x is a sequence of length n in a lattice $(X, \wedge, \vee)$, then we refer to x^{Δ} as defined by Identity* (8) *as the* increasing sort of *x with respect to the lattice $(X, \wedge, \vee)$.*

Before we start to investigate which properties that are traditionally associated with sorting are maintained by our definition we want to point out a major difference: In a lattice the value x_k^{Δ} might be different from the original values $x_1, \ldots, x_n$. The reason for this is the following: While in a lattice the inequalities $x \wedge y \leq x, y \leq x \vee y$ generally hold, there might be also the case that the set $\{x \wedge y, x \vee y\}$ is different from the set $\{x, y\}$. In a total order these two sets are always equal.

Examples of sorting in lattices As a first example we consider the finite set $X = \{x, y, z\}$. Figure 1 shows the lattice of all subsets of X. Let x be the sequence $a = (\{x\}, \{y\}, \{z\})$, then $a^{\Delta} = (\emptyset, \emptyset, X)$. Thus, a^{Δ} is a increasing sequence that consists of elements that are completely different from those of a.

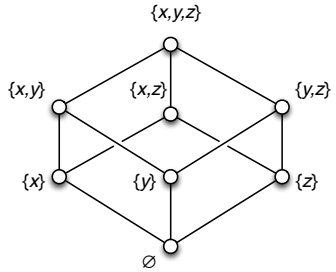


Figure 1. The lattice of $\{x, y, z\}$

x	x^{Δ}
(1)	(1)
(1, 2)	(1, 2)
(1, 2, 3)	(1, 1, 6)
(1, 2, 3, 4)	(1, 1, 2, 12)
(1, 2, 3, 4, 5)	(1, 1, 1, 2, 60)
(1, 2, 3, 4, 5, 6)	(1, 1, 1, 2, 6, 60)
(1, 2, 3, 4, 5, 6, 7)	(1, 1, 1, 1, 2, 6, 420)
(1, 2, 3, 4, 5, 6, 7, 8)	(1, 1, 1, 1, 2, 2, 12, 840)

Table 1. Sorting in the lattice $(\mathbb{N}, \gcd, \mathrm{lcm})$

As a second example we consider the lattice $(\mathbb{N}, \gcd, \mathrm{lcm})$ where $\gcd(x, y)$ and $\mathrm{lcm}(x, y)$ denote the *greatest common divisor* and *least common multiple* of x and y,

respectively. The associated partial order of this lattices is defined by divisibility of natural numbers. Table 1 shows some examples of our definition of sorting for different sequences in $(\mathbb{N}, \gcd, \operatorname{lcm})$. Again we see that sorting in a lattice may change the elements in a sequence.

Elementary properties of sorting in lattices The following lemma states that x^Δ is indeed a increasing sequence with respect to the partial order $(X, \leq)$ of the lattice $(X, \wedge, \vee)$.

Lemma 2. *If x is a finite sequence in a lattice $(X, \wedge, \vee)$ with associated partial order $(X, \leq)$, then Identity* (8) *defines a increasing sequence x^Δ.*

Proof. In order to prove this lemma we can proceed exactly as in the proof of Lemma 1 where $(X, \leq)$ is a total order. As remarked on Page 2, we have used only the fact that $\bigwedge A$ is a lower bound of A which by definition also holds for lattices. □

A simple consequence of Lemma 2 is the following Lemma 3 which states that sorting in lattices respects lower and upper bounds of the original sequence.

Lemma 3. *Let x be a sequence of length n in a lattice $(X, \wedge, \vee)$ with associated partial order $(X, \leq)$. If for $1 \leq i \leq n$ holds $a \leq x_i \leq b$, then $a \leq x_i^\Delta \leq b$ holds as well.*

Proof. From Identity (10) follows that x_n^Δ is the supremum of the elements $x_1, \ldots, x_n$. Thus, we have $x_n^\Delta \leq b$. Lemma 2 ensures that x_n^Δ is the largest element of x^Δ. Thus we have $x_i^\Delta \leq b$ for $1 \leq i \leq n$. The case for the lower bound a is treated analogously. □

The following lemma restates the idempotence of sorting for the case of lattices (cf. Identity (1)).

Lemma 4. *If x is a finite sequence in a lattice $(X, \wedge, \vee)$, then $(x^\Delta)^\Delta = x^\Delta$.*

Proof. We know from Lemma 2 that x^Δ is a increasing sequence in the partial order $(X, \leq)$. Thus, the relation $\leq$ is a *total order* on the set $\left\{x_1^\Delta, \ldots, x_n^\Delta\right\} \subset X$. In other words we can sort x^Δ in the classical sense. From this follows by Identity (7)

$$x^\Delta = (x^\Delta)^\uparrow = (x^\Delta)^\Delta.$$

□

We can also show the invariance of sorting in lattices under permutations (cf. Identity (2)).

Lemma 5. *If x is a sequence of length n in a lattice and ψ a permutation of $[1, n]$, then $(x \circ \psi)^\Delta = x^\Delta$ holds.*

Proof. We have for $1 \leq k \leq n$

$$(x \circ \psi)_k^\Delta = \bigwedge_{A \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in A} (x \circ \psi)_i = \bigwedge_{A \in \mathbb{N}\binom{n}{k}} \bigvee_{j \in \psi(A)} x_j = \bigwedge_{B \in \psi\left(\mathbb{N}\binom{n}{k}\right)} \bigvee_{j \in B} x_j$$

Because ψ is a permutation of $[1, n]$ we find that $\psi\left(\mathbb{N}\binom{n}{k}\right) = \mathbb{N}\binom{n}{k}$ and conclude

$$(x \circ \psi)_k^\Delta = \bigwedge_{B \in \mathbb{N}\binom{n}{k}} \bigvee_{j \in B} x(j) = x_k^\Delta.$$

□

4 Recursive sorting in lattices

The definition of x^Δ through Identity (8) is nice and succinct, but it is also quite impractical to use in computations. Table 2 shows simple performance measurements (conducted on a notebook computer) for computing $(1,\dots,n)^\Delta$ in $(\mathbb{N}, \gcd, \operatorname{lcm})$. The reason for this dramatic slowdown is of course the exponential complexity inherent in Identity (8): In order to compute x^Δ from x it is necessary to consider all $2^n - 1$ nonempty subsets of $[1, n]$.

sequence length	20	21	22	23	24	25	26
time in s	0.6	1.3	2.7	5.8	11.8	25.5	51.6

Table 2. Wall-clock time for computing $(1,\dots,n)^\Delta$ according to Identity (8)

For the remainder of this paper we assume that $(X, \wedge, \vee, \bot, \top)$ is a *bounded* lattice. Here $\bot$ is the least element of X and the neutral element of join, that is,

$$x = \bot \vee x = x \vee \bot \qquad \forall x \in X \tag{14}$$

whereas $\top$ is the greatest element of X and the neutral element of meet, that is,

$$x = \top \wedge x = x \wedge \top \qquad \forall x \in X. \tag{15}$$

We now introduce a notation that allows us to concisely refer to individual elements of both $(x_1,\dots,x_n)^\Delta$ and $(x_1,\dots,x_{n-1})^\Delta$. Here again, it is convenient to employ the notation for the binomial coefficient $\binom{n}{k}$ in the context of sorting in lattices. For a sequence x of length n we define for $0 \le m \le n$

$$x^\Delta \binom{m}{k} \coloneqq \begin{cases} \bot & k = 0 \\ (x_1,\dots,x_m)^\Delta(k) & k \in [1,m] \\ \top & k = m+1 \end{cases} \tag{16}$$

We know from Identity (8) that $(x_1,\dots,x_m)^\Delta(k) = \bigwedge_{I \in \mathbb{N}\binom{m}{k}} \bigvee_{i \in I} x_i$ holds for $1 \le k \le m$.

We therefore have

$$x^\Delta \binom{m}{k} = \bigwedge_{I \in \mathbb{N}\binom{m}{k}} \bigvee_{i \in I} x_i. \tag{17}$$

In particular, the following identity holds for $1 \le k \le n$

$$x^\Delta \binom{n}{k} = x^\Delta_k. \tag{18}$$

The main result of this section is Proposition 2, which states in Identity (19), how the kth element of $(x_1,\dots,x_n)^\Delta$ can be computed from $(x_1,\dots,x_{n-1})^\Delta$ and x_n by simply applying one *join* and one *meet*. The proof of Proposition 2 relies on the fact that the lattice under consideration is both *bounded* and *distributive*.

Proposition 2. *If $(X, \wedge, \vee, \bot, \top)$ is a bounded distributive lattice and if x is a sequence of length n, then for $1 \leq k \leq n$ holds*

$$x^{\vartriangle}\binom{n}{k} = x^{\vartriangle}\binom{n-1}{k} \wedge \left(x^{\vartriangle}\binom{n-1}{k-1} \vee x_n\right) \tag{19}$$

Proof. For $k = 1$, we have

$$\begin{aligned}
x^{\vartriangle}\binom{n}{1} &= \bigwedge_{i=1}^{n} x_i && \text{by Identity (17)} \\
&= \left(\bigwedge_{i=1}^{n-1} x_i\right) \wedge x_n && \text{by associativity} \\
&= x^{\vartriangle}\binom{n-1}{1} \wedge x_n && \text{by Identity (17)} \\
&= x^{\vartriangle}\binom{n-1}{1} \wedge \left(\bot \vee x_n\right) && \text{by Identity (14)} \\
&= x^{\vartriangle}\binom{n-1}{1} \wedge \left(x^{\vartriangle}\binom{n-1}{0} \vee x_n\right) && \text{by Identity (16).}
\end{aligned}$$

We deal similarly with the case $k = n$ (cf. [2, p. 5]). In the general case of $1 < k < n$, we first remark that if A is a subset of $[1, n]$, which consists of k elements, then there are two cases possible:

1. If n does not belong to A, then A is a subset of $\mathbb{N}\binom{n-1}{k}$.
2. If n is an element of A, then the set $B \coloneqq A \setminus \{n\}$ belongs to $\mathbb{N}\binom{n-1}{k-1}$.

In other words, $\mathbb{N}\binom{n}{k}$ can be represented as the following (disjoint) union

$$\mathbb{N}\binom{n}{k} = \mathbb{N}\binom{n-1}{k} \cup \left\{B \cup \{n\} \;\middle|\; B \in \mathbb{N}\binom{n-1}{k-1}\right\}. \tag{20}$$

We obtain therefore

$$\begin{aligned}
x^{\vartriangle}\binom{n}{k} &= \bigwedge_{I \in \mathbb{N}\binom{n}{k}} \bigvee_{i \in I} x_i && && && \text{by Identity (17)} \\
&= \bigwedge_{I \in \mathbb{N}\binom{n-1}{k}} \bigvee_{i \in I} x_i && \wedge && \bigwedge_{I \in \mathbb{N}\binom{n-1}{k-1}} \bigvee_{i \in I \cup \{n\}} x_i && \text{by Identity (20)} \\
&= x^{\vartriangle}\binom{n-1}{k} && \wedge && \bigwedge_{I \in \mathbb{N}\binom{n-1}{k-1}} \bigvee_{i \in I \cup \{n\}} x_i && \text{by Identity (17)} \\
&= x^{\vartriangle}\binom{n-1}{k} && \wedge && \bigwedge_{I \in \mathbb{N}\binom{n-1}{k-1}} \left(\bigvee_{i \in I} x_i \vee x_n\right) && \text{by associativity} \\
&= x^{\vartriangle}\binom{n-1}{k} && \wedge && \left(\left(\bigwedge_{I \in \mathbb{N}\binom{n-1}{k-1}} \bigvee_{i \in I} x_i\right) \vee x_n\right) && \text{by distributivity} \\
&= x^{\vartriangle}\binom{n-1}{k} && \wedge && \left(x^{\vartriangle}\binom{n-1}{k-1} \vee x_n\right) && \text{by Identity (17)}
\end{aligned}$$

which completes the proof. □

The following Proposition 3 states that the converse of Proposition 2 also holds.

Proposition 3. *Let* $(X, \wedge, \vee, \bot, \top)$ *be a bounded lattice which is* not *distributive. Then there exists a sequence* $x = (x_1, x_2, x_3)$ *in* X *such that Identity* (19) *is not satisfied.*

Proof. According to a standard result on distributive lattices [5, Theorem 4.7], a lattice is *not* distributive, if and only if it contains a sublattice which is isomorphic to either N_5 or M_3 (cf. Figure 2).

Figure 2. The non-distributive lattices N_5 and M_3

From Identity (10) follows for the elements of $x^\Delta = \left(x_1^\Delta, x_2^\Delta, x_3^\Delta\right)$

$$x_1^\Delta = x_1 \wedge x_2 \wedge x_3 \tag{21a}$$
$$x_2^\Delta = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \tag{21b}$$
$$x_3^\Delta = x_1 \vee x_2 \vee x_3. \tag{21c}$$

If X contains the sublattice N_5, then we consider the sequence $x = (c, d, b)$ and its subsequence (c, d). From Identity (21) then follows

$$(c, d, b)^\Delta = (a, d, e) \qquad \text{and} \qquad (c, d)^\Delta = (a, e).$$

Thus, we have

$$x^\Delta \binom{3}{2} = d \qquad x^\Delta \binom{2}{2} = e \qquad x^\Delta \binom{2}{1} = a.$$

However, applying Identity (19) we obtain

$$\begin{aligned} x^\Delta \binom{3}{2} &= x^\Delta \binom{2}{2} \wedge \left(x^\Delta \binom{2}{1} \vee x_3\right) \\ &= e \wedge (a \vee b) = e \wedge b = b \end{aligned}$$

instead of d.

If X contains the sublattice M_3, then we consider the sequence $x = (b, c, d)$ and its subsequence (b, c). From Identity (21) then follows

$$(b, c, d)^\Delta = (a, e, e) \qquad \text{and} \qquad (b, c)^\Delta = (a, e).$$

We therefore have

$$x^{\Delta}\binom{3}{2} = e \qquad x^{\Delta}\binom{2}{2} = e \qquad x^{\Delta}\binom{2}{1} = a.$$

Again, applying Identity (19) we obtain

$$\begin{aligned} x^{\Delta}\binom{3}{2} &= x^{\Delta}\binom{2}{2} \wedge \left(x^{\Delta}\binom{2}{1} \vee x_3\right) \\ &= e \wedge (a \vee d) = e \wedge d = d \end{aligned}$$

instead of e. □

Using Identity (19), we can prove the following Lemma 6, which generalizes a known fact known from sorting in a total order: If one knows that x_n is greater or equal that the preceding elements $x_1, \ldots, x_{n-1}$ then sorting the sequence $(x_1, \ldots, x_n)$ can be accomplished by sorting $(x_1, \ldots, x_{n-1})$ and simply appending x_n.

Lemma 6. *Let $(X, \wedge, \vee, \bot, \top)$ be a bounded distributive lattice and x be a sequence of length n. If the condition $x_i \leq x_n$ holds for $1 \leq i \leq n-1$, then the identities*

$$\begin{aligned} x^{\Delta}\binom{n}{i} &= x^{\Delta}\binom{n-1}{i} \\ x^{\Delta}\binom{n}{n} &= x_n \end{aligned}$$

hold.

Proof. The first equation follows directly from the fact that x_n^{Δ} is the supremum of the values $x_1, \ldots, x_n$. Regarding the second equation, we know from Lemma 2 that if for $1 \leq i \leq n-1$ the inequality $x_i \leq x_n$ holds, then

$$x^{\Delta}\binom{n-1}{i} \leq x_n.$$

This inequality is also valid for $i = 0$ because $x^{\Delta}\binom{n-1}{0} = \bot$ holds by Identity (16). From general properties of meet and join then follows that

$$\begin{aligned} x^{\Delta}\binom{n-1}{i} \vee x_n &= x_n \\ x^{\Delta}\binom{n-1}{i} \wedge x_n &= x^{\Delta}\binom{n-1}{i} \end{aligned}$$

holds for $0 \leq i \leq n-1$. We can therefore simplify Identity (19) as follows

$$\begin{aligned} x^{\Delta}\binom{n}{i} &= x^{\Delta}\binom{n-1}{i} \wedge \left(x^{\Delta}\binom{n-1}{i-1} \vee x_n\right) \\ &= x^{\Delta}\binom{n-1}{i} \wedge x_n \\ &= x^{\Delta}\binom{n-1}{i}. \end{aligned}$$

□

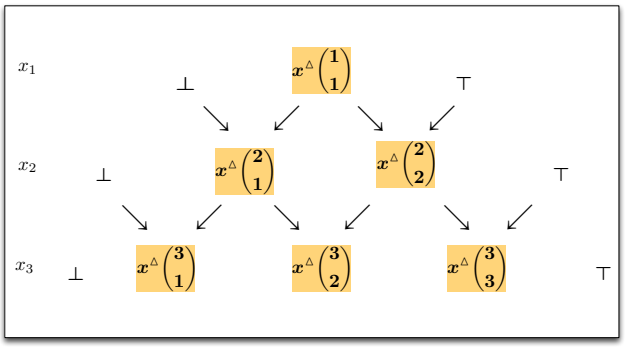


Figure 3. Graphical representation of Identity (19)

5 Insertion sort in lattices

Figure 3 graphically represents Identity (19) in a form that emphasizes its close relationship to Pascal's triangle. Whenever an arrow $\searrow$ and and arrow $\swarrow$ meet, the values are combined by a *meet*. In the case of an arrow $\searrow$, however, first the value at the origin of the arrow is combined with the sequence value x_n through a *join*.

Formula (22) outlines an algorithm that is based on Identity (19). The algorithm starts from $x_1 = (x_1)^\Delta$ and successively computes

$$(x_1, \ldots, x_{i-1})^\Delta, x_i \quad \mapsto \quad (x_1, \ldots, x_{i-1}, x_i)^\Delta. \tag{22}$$

From Identity (19) follows that in step i exactly i joins and i meets must be performed. Thus, altogether there are

$$\sum_{i=2}^{n} 2 * i = n(n+1) - 2$$

applications of join and meet. In other words, such an implementation has quadratic complexity. This algorithm can be considered as *insertion sort* [3, § 5.2.1] for lattices because one element at a time is added to an already "sorted" sequence. Table 3 shows some performance measurements for this algorithm in the bounded and distributive lattice $(\mathbb{N}, \gcd, \mathrm{lcm}, 1, 0)$.

sequence length	100	1000	10000	100000
time in s	0	0	3.4	420

Table 3. Wall-clock time for computing $(1, \ldots, n)^\Delta$ according to Identity (19)

These results show that sorting in lattices can now be applied to much larger sequences than those shown in Table 2 before the limitations of an algorithm with quadratic complexity become noticeable.

6 Conclusions

Proposition 1 states through Identity (7) a simple explicit relationship between the elements of a finite sequence in a totally ordered sets to its sorted counterpart.

A sorting algorithm that directly uses Identity (7) would have exponential complexity. Thus, Identity (7) appears not relevant for implementing computationally efficient algorithms. The reader should bear in mind, however, that this is also true for the Binomial Theorem. In fact, directly computing $(x + y)^n$ is normally more efficient than computing the expansion

$$x^n + nx^{n-1}y + \frac{n(n-1)}{2}x^{n-2}y^2 + \ldots + y^n$$

A more interesting aspect of Identity (7) is therefore that it allows to *generalize the notion of sorting finite sequences to lattices*. Compared to sorting in a totally ordered set, sorting in lattices is a more *invasive* procedure because it may change sequence elements. While this may be considered as a major drawback one should bear in mind that generalizations often lead to surprising properties. The real criterion for accepting a generalization is whether it provides new insights or has useful applications. With respect to sorting in lattices, the latter question has not been addressed in this paper and remains a topic of future research.

We are able to show that our definition of sorting in lattices maintains many properties that are associated with sorting. Another important results of this paper are Proposition 2, which proves Identity (19) for bounded distributive lattices, and Proposition 3, which shows that the distributivity is necessary for Identity (19) to hold. The remarkable points of Identity (19) are that it

- exhibits a strong analogy between sorting and Pascal's triangle,
- allows to sort in lattices with quadratic complexity, and that it
- is in fact a generalization of *insertion sort* for lattices.

I would like to thank the reviewers for their comments. I am also very grateful for the many corrections and valuable suggestions of my colleagues Jochen Burghardt and Hans Werner Pohl: Jochen Burghardt's suggestion to investigate whether the distributivity in Proposition 2 is really necessary led to Proposition 3. Hans Werner Pohl pointed out the analogy of the algorithm in Equation 22 to insertion sort.

References

1. J. Gerlach. Sorting in Lattices. *ArXiv e-prints*, March 2013. http://arxiv.org/abs/1303.5560.
2. J. Gerlach. Recursive Sorting in Lattices. *ArXiv e-prints*, May 2013 http://arxiv.org/abs/1306.0019.
3. Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
4. Bourbaki, N., *Elements of Mathematics, Theory of Sets*, Addison-Wesley, Reading, MA, 1968.
5. S. Roman. *Lattices and Ordered Sets*. Springer-Verlag New York, 2008.

Rough Inclusion Functions and Similarity Indices*

Anna Gomolińska[1] and Marcin Wolski[2]

[1] Białystok University, Computer Science Institute,
Sosnowa 64, 15-887 Białystok, Poland,
anna.gom@math.uwb.edu.pl
[2] Maria Curie-Skłodowska University, Dept. of Logic and Philosophy of Science,
Pl. Marii Curie-Skłodowskiej 4, 20-031 Lublin, Poland,
marcin.wolski@umcs.lublin.pl

Abstract. Rough inclusion functions are mappings considered in the rough set theory with which one can measure the degree of inclusion of a set in a set (and in particular, the degree of inclusion of an information granule in an information granule) in line with rough mereology. On the other hand, similarity indices are mappings in cluster analysis with which one can compare clusterings, and clustering methods with respect to similarity. In this article we investigate the relationships between rough inclusion functions and similarity indices.

Keywords: rough inclusion function, rough mereology, similarity index, cluster analysis, granular computing.

1 Introduction

In 1994, L. Polkowski and A. Skowron introduced the formal notion of a *rough inclusion*, making it a fundamental concept of *rough mereology* (see, e.g. [1–4]).[3] Rough inclusion may be interpreted as a ternary relation with which one can express the fact that a set of objects is to some degree included in the same or another set of objects. Rough mereology is a theory extending the Leśniewski mereology [6, 7] from a theory of being-a-part to a theory of being-a-part-to-degree. *Rough inclusion functions* (RIFs) are mappings with which one can measure the degree of inclusion of sets in sets and which comply with the axioms of rough inclusion. Since according to L. A. Zadeh's definition [8], an *information granule* is a clump of objects drawn together on the basis of indistinguishability, similarity or functionality, RIFs can be used in particular to measure the degree of inclusion of information granules in information granules. Hence, the concept of a RIF is fundamental not only for the rough set theory [5, 9] but also for the foundations and the development of granular computing [10, 11].

* Many thanks to the anonymous referees for interesting comments on the paper. All errors left are our sole responsibility.

[3] It is worthy to note that some ideas on rough inclusion were presented by Z. Pawlak in [5].

RIFs can be useful in the rough set theory and, more generally, in granular computing in many ways. First, they can be applied to compare sets (and information granules) with respect to inclusion. Secondly, they can be used to define rough membership functions [12] and various approximation operators as those in the Skowron – Stepaniuk approach (see, e.g. [13, 14] and other papers by the same authors), in the Ziarko variable-precision rough set model (see, e.g. [15, 16] and more recent papers), or in the decision-theoretic rough set model [17, 18]. RIFs can also be used to estimate the confidence (known as accuracy as well) and the coverage of decision rules and association rules (see, e.g. [19]). Another application of RIFs is graded semantics of formulas (see, e.g. [20]). An important application of RIFs is obviously their usage to compute the degree of similarity (nearness, closeness) between sets of objects and, in particular, between information granules. Some steps into this direction have already been made (see, e.g. [21, 4, 14]).

The *similarity indices* we are going to speak about are used in *cluster analysis* [22–24] to compare clusterings, and clustering methods with respect to how they are similar to (or dissimilar from) one another. Many of these similarity indices were originally designed to compare species with respect to their mutual similarity, given information about presence and/or absence of some features. A. N. Albatineh, M. Niewiadomska-Bugaj, and D. Michalko thoroughly examined 28 similarity indices known from the literature on classification and cluster analysis, from which 22 turned out to be different.[4] The results of their research on *correction for chance agreement* for similarity indices can be found, e.g. in [25]. In the present article we continue our earlier works [26, 27], where among other things, three similarity indices out of those 22 were derived from RIFs. Our actual goal is to show that all 22 similarity indices investigated in [25] can be obtained starting with the RIFs $\kappa^{\pounds}$, κ_1, and κ_2 only. This reveals one more connection between the rough set theory and cluster analysis.

The rest of the paper is organized as follows. In Sect. 2 we recall the notion of a rough inclusion function and the three particular RIFs mentioned above. In Sect. 3 we present the 22 similarity indices known from the literature and discussed in [25], and we characterize them one by one by means of the standard RIF $\kappa^{\pounds}$ or two other RIFs, viz. κ_1 and κ_2. The last section contains final remarks.

2 Rough Inclusion Functions

Rough inclusion functions (RIFs for short) are supposed to be mappings to measure the degree of inclusion of sets in sets and to comply with the axioms of rough inclusion. In detail, a rough inclusion function upon a non-empty set of objects U (in short, a RIF upon U or simply, a RIF) is a mapping $\kappa : \wp U \times \wp U \mapsto [0, 1]$, assigning to any pair of sets (X, Y) of elements of U, a number $\kappa(X, Y)$ from the unit interval $[0, 1]$ interpreted as the degree to which X is included in

[4] Some similarity indices were introduced more than once, under different names.

Y, and such that the conditions $\mathrm{rif}_1(\kappa)$ and $\mathrm{rif}_2^*(\kappa)$ are satisfied, where

$$\mathrm{rif}_1(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall X, Y \subseteq U.(\kappa(X,Y) = 1 \;\Leftrightarrow\; X \subseteq Y),$$
$$\mathrm{rif}_2^*(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall X, Y, Z \subseteq U.(\kappa(Y,Z) = 1 \;\Rightarrow\; \kappa(X,Y) \leq \kappa(X,Z)).$$

Condition $\mathrm{rif}_1(\kappa)$ expresses the fact that the set-theoretical inclusion of sets is the most perfect case of rough inclusion. When $\mathrm{rif}_1(\kappa)$ holds, condition $\mathrm{rif}_2^*(\kappa)$ will be equivalent with condition $\mathrm{rif}_2(\kappa)$ below:

$$\mathrm{rif}_2(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall X, Y, Z \subseteq U.(Y \subseteq Z \;\Rightarrow\; \kappa(X,Y) \leq \kappa(X,Z))$$

expressing monotonicity of κ in the second variable. In the literature, weaker versions of RIFs are considered as well, where $\mathrm{rif}_1(\kappa)$ is replaced by "a half of it". Then, $\mathrm{rif}_2^*(\kappa)$ and $\mathrm{rif}_2(\kappa)$ will define different classes of inclusion mappings (see, e.g. [28]).

In summary, any RIF κ upon U should satisfy $\mathrm{rif}_1(\kappa)$ and $\mathrm{rif}_2^*(\kappa)$ or, equivalently, $\mathrm{rif}_1(\kappa)$ and $\mathrm{rif}_2(\kappa)$. Among RIFs, various subclasses of mappings can be distinguished by adding new postulates to be satisfied. These can be, for instance,

$$\mathrm{rif}_3(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall \emptyset \neq X \subseteq U.\kappa(X,\emptyset) = 0,$$
$$\mathrm{rif}_4(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall X, Y \subseteq U.(\kappa(X,Y) = 0 \;\Rightarrow\; X \cap Y = \emptyset),$$
$$\mathrm{rif}_4^{-1}(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall \emptyset \neq X \subseteq U. \forall Y \subseteq U.(X \cap Y = \emptyset \;\Rightarrow\; \kappa(X,Y) = 0),$$
$$\mathrm{rif}_5(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall \emptyset \neq X \subseteq U. \forall Y \subseteq U.(\kappa(X,Y) = 0 \;\Leftrightarrow\; X \cap Y = \emptyset),$$
$$\mathrm{rif}_6(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall \emptyset \neq X \subseteq U. \forall Y \subseteq U. \kappa(X,Y) + \kappa(X,Y^c) = 1,$$
$$\mathrm{rif}_7(\kappa) \overset{\mathrm{def}}{\Leftrightarrow} \forall X, Y, Z \subseteq U.(Z \subseteq Y \subseteq X \;\Rightarrow\; \kappa(X,Z) \leq \kappa(Y,Z)),$$

where Y^c denotes the set-theoretical complement of Y.[5] Obviously, $\mathrm{rif}_5(\kappa)$ if and only if $\mathrm{rif}_4(\kappa)$ and $\mathrm{rif}_4^{-1}(\kappa)$. Apart from that

$$\mathrm{rif}_4^{-1}(\kappa) \;\Rightarrow\; \mathrm{rif}_3(\kappa),$$
$$\mathrm{rif}_1(\kappa) \;\&\; \mathrm{rif}_6(\kappa) \;\Rightarrow\; \mathrm{rif}_5(\kappa). \tag{1}$$

The *standard* RIF, denoted by $\kappa^{\pounds}$ here, is the most famous and frequently used by the rough set community. The idea underlying this notion is closely related to the conditional probability. In logic, J. Łukasiewicz was the first who employed this idea when calculating the probability of truth associated with implicative formulas [31, 32]. Let us recall that $\kappa^{\pounds}$ is only defined for a finite U by putting

$$\kappa^{\pounds}(X,Y) \overset{\mathrm{def}}{=} \begin{cases} \frac{\#(X \cap Y)}{\#X} & \text{if } X \neq \emptyset, \\ 1 & \text{otherwise,} \end{cases} \tag{2}$$

[5] The last condition was mentioned in [29, 30]. There, rough inclusion is understood in a different way than in our paper.

where X, Y are any subsets of U and $\#X$ denotes the number of elements of X. In words, the standard RIF measures the fraction of the elements having the property described by the second argument (Y) among the elements with the property described by the first argument (X). Apart from being a true RIF, $\kappa^{\pounds}$ has a number of interesting properties recalled, e.g. in [27]. For instance, it satisfies $\mathrm{rif}_i(\kappa)$ $(i = 3, \ldots, 7)$ and $\mathrm{rif}_4^{-1}(\kappa)$.

Examples of other RIFs are mappings κ_1 and κ_2 such that for any $X, Y \subseteq U$,

$$\kappa_1(X,Y) \stackrel{\mathrm{def}}{=} \begin{cases} \frac{\#Y}{\#(X \cup Y)} & \text{if } X \cup Y \neq \emptyset, \\ 1 & \text{otherwise,} \end{cases}$$

$$\kappa_2(X,Y) \stackrel{\mathrm{def}}{=} \frac{\#(X^c \cup Y)}{\#U}. \tag{3}$$

Also in this case, U has to be finite. While κ_1 was introduced in [26], κ_2 had already been mentioned in [33]. The both RIFs were investigated in detail in [27]. The RIFs $\kappa^{\pounds}$, κ_1, and κ_2 are different from one another. Below we recall a few other properties of these mappings.

Proposition 1. *For any $X, Y \subseteq U$, we have:*

(i) $X \neq \emptyset \Rightarrow (\kappa_1(X,Y) = 0 \Leftrightarrow Y = \emptyset)$,
(ii) $\kappa_2(X,Y) = 0 \Leftrightarrow X = U \,\&\, Y = \emptyset$,
(iii) $\mathrm{rif}_4(\kappa_1)$ & $\mathrm{rif}_4(\kappa_2)$,
(iv) $\kappa^{\pounds}(X,Y) \leq \kappa_1(X,Y) \leq \kappa_2(X,Y)$,
(v) $\kappa_1(X,Y) = \kappa^{\pounds}(X \cup Y, Y)$ & $\kappa^{\pounds}(X,Y) = \kappa_1(X, X \cap Y)$,
(vi) $\kappa_2(X,Y) = \kappa^{\pounds}(U, X^c \cup Y)$.

Let us also note that due to (i), $\mathrm{rif}_3(\kappa_1)$ holds. The same cannot be however said about κ_2 (compare (ii)).

3 Similarity Indices in Terms of RIFs

In this section we reformulate the similarity indices studied in [25] in terms of the RIFs $\kappa^{\pounds}$, κ_1, or κ_2. The proofs that the indices can really be expressed in this way will be given in the full version of this paper.

Consider a set U_0 of $m > 0$ data points to be grouped by some clustering methods A_1 and A_2. Let U (our universe) be the set of all unordered pairs of data points $\{x, y\} \subseteq U_0$ to be compared in order to obtain clusterings, i.e. partitions of U_0 generated by A_1 and by A_2, and denoted by C_1 and C_2 here. Thus, $\#U = M = \binom{m}{2} = m(m-1)/2$. The similarity between the clusterings C_1 and C_2 (and the clustering methods A_1 and A_2) is usually assessed on the basis of the number of pairs of data points that are put into the same cluster or are put into different clusters by each of the grouping methods considered. For $i = 1, 2$, let us define

$$X_i = \{\{x, y\} \in U \mid x, y \text{ are clustered by } A_i\}. \tag{4}$$

Additionally, let

$$\begin{aligned} a &= \#(X_1 \cap X_2), \\ b &= \#(X_1 \cap X_2^c), \\ c &= \#(X_1^c \cap X_2), \\ d &= \#(X_1^c \cap X_2^c). \end{aligned} \tag{5}$$

In words, a is the number of pairs of data points $\{x, y\}$ such that x and y are placed in the same cluster according to both A_1 and A_2; b (respectively, c) is the number of pairs of data points $\{x, y\}$ such that x and y are placed in the same cluster by A_1 (resp., A_2), but they are placed in different clusters by A_2 (resp., A_1); finally, d is the number of pairs of data points $\{x, y\}$ such that x and y are placed in different clusters according to both A_1 and A_2. We also have $\#X_1 = a+b$, $\#X_2 = a+c$, $\#X_1^c = c+d$, $\#X_2^c = b+d$, and $\#U = a+b+c+d = M$. For simplicity assume that $a, b, c, d > 0$. Then, we will have that

$$\begin{aligned} \kappa^{\pounds}(X_1, X_2) &= \frac{a}{a+b}, \\ \kappa_1(X_1, X_2) &= \frac{a+c}{a+b+c}, \\ \kappa_2(X_1, X_2) &= \frac{a+c+d}{M}. \end{aligned} \tag{6}$$

In what follows we will present similarity indices one by one and their new formulation in terms of $\kappa^{\pounds}$, κ_1, or κ_2.

Wallace (1983). The similarity indices W_1, W_2 with range $[0, 1]$ were introduced by D. L. Wallace:

$$\begin{aligned} W_1(C_1, C_2) &\stackrel{\text{def}}{=} \frac{a}{a+b}, \\ W_2(C_1, C_2) &\stackrel{\text{def}}{=} \frac{a}{a+c}. \end{aligned} \tag{7}$$

It is easy to see that

$$\begin{aligned} W_1(C_1, C_2) &= \kappa^{\pounds}(X_1, X_2), \\ W_2(C_1, C_2) &= \kappa^{\pounds}(X_2, X_1). \end{aligned} \tag{8}$$

Kulczyński (1927). The similarity index K with range $[0, 1]$ was proposed by S. Kulczyński in 1927:

$$K(C_1, C_2) \stackrel{\text{def}}{=} \frac{1}{2}\left(\frac{a}{a+b} + \frac{a}{a+c}\right) \tag{9}$$

K can be rewritten to the following form:

$$K(C_1, C_2) = \frac{1}{2}\left(\kappa^{\pounds}(X_1, X_2) + \kappa^{\pounds}(X_2, X_1)\right) \tag{10}$$

In words, $K(C_1, C_2)$ is the arithmetical mean of $\kappa^{\pounds}(X_1, X_2)$ and $\kappa^{\pounds}(X_2, X_1)$.

McConnaughey (1964). The similarity index MC with range $[-1, 1]$ goes back to B. H. McConnaughey:

$$MC(C_1, C_2) \stackrel{\text{def}}{=} \frac{a^2 - bc}{(a+b)(a+c)} \tag{11}$$

This index can be expressed by the following equation:

$$MC(C_1, C_2) = \kappa^{\mathcal{L}}(X_1, X_2) + \kappa^{\mathcal{L}}(X_2, X_1) - 1 \tag{12}$$

Peirce (1884). The similarity index PE with range $[-1, 1]$ is attributed to C. S. Peirce:

$$PE(C_1, C_2) \stackrel{\text{def}}{=} \frac{ad - bc}{(a+c)(b+d)} \tag{13}$$

The index PE can be characterized as follows:

$$PE(C_1, C_2) = \frac{1}{2}\left(\kappa^{\mathcal{L}}(X_2, X_1) + \kappa^{\mathcal{L}}(X_2^c, X_1^c) - \kappa^{\mathcal{L}}(X_2, X_1^c) - \kappa^{\mathcal{L}}(X_2^c, X_1)\right) \tag{14}$$

The Gamma index. The similarity index Γ with range $[-1, 1]$ is given by

$$\Gamma(C_1, C_2) \stackrel{\text{def}}{=} \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}. \tag{15}$$

In this case, the following characterization can be obtained:

$$\begin{aligned}\Gamma(C_1, C_2) = &\sqrt{\frac{1}{2}\left(\kappa^{\mathcal{L}}(X_2, X_1) + \kappa^{\mathcal{L}}(X_2^c, X_1^c) - \kappa^{\mathcal{L}}(X_2, X_1^c) - \kappa^{\mathcal{L}}(X_2^c, X_1)\right)} \\ &\cdot \sqrt{\kappa^{\mathcal{L}}(X_1, X_2) - \kappa^{\mathcal{L}}(X_1^c, X_2)}\end{aligned} \tag{16}$$

Ochiai (1957), Fowlkes and Mallows (1983). The similarity index OFM ranges over $[0, 1]$. It was introduced by A. Ochiai in 1957 and again by E. B. Fowlkes and C. L. Mallows in 1983:

$$OFM(C_1, C_2) \stackrel{\text{def}}{=} \frac{a}{\sqrt{(a+b)(a+c)}} \tag{17}$$

After rewriting we get

$$OFM(C_1, C_2) = \sqrt{\kappa^{\mathcal{L}}(X_1, X_2)\kappa^{\mathcal{L}}(X_2, X_1)}. \tag{18}$$

That is, $OFM(C_1, C_2)$ is the geometrical mean of $\kappa^{\mathcal{L}}(X_1, X_2)$ and $\kappa^{\mathcal{L}}(X_2, X_1)$.

The Pearson index. The similarity index P named after C. Pearson ranges over $[-1, 1]$. It is given by

$$P(C_1, C_2) \stackrel{\text{def}}{=} \frac{ad - bc}{(a+b)(a+c)(b+d)(c+d)}. \tag{19}$$

The index P can be expressed in the following ways:

$$\begin{aligned} P(C_1, C_2) &= \begin{vmatrix} a & b \\ c & d \end{vmatrix}^{-1} \cdot \Gamma^2(C_1, C_2) \\ &= (\kappa^{\pounds}(X_1, X_2) - \kappa^{\pounds}(X_1^c, X_2))\kappa^{\pounds}(X_2, \{u\})\kappa^{\pounds}(X_2^c, \{u'\}) \end{aligned} \tag{20}$$

for arbitrary $u \in X_2$ and $u' \notin X_2$.

Sokal and Sneath (1963). The similarity indices SS_1, SS_2, SS_3 with range $[0, 1]$ were introduced by R. R. Sokal and P. H. Sneath in 1963. The third index is also attributed to A. Ochiai (1957):

$$\begin{aligned} SS_1(C_1, C_2) &\stackrel{\text{def}}{=} \frac{1}{4}\left(\frac{a}{a+b} + \frac{a}{a+c} + \frac{d}{b+d} + \frac{d}{c+d}\right), \\ SS_2(C_1, C_2) &\stackrel{\text{def}}{=} \frac{a}{a + 2(b+c)}, \\ SS_3(C_1, C_2) &\stackrel{\text{def}}{=} \frac{ad}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}. \end{aligned} \tag{21}$$

One can prove the following:

$$\begin{aligned} SS_1(C_1, C_2) &= \frac{1}{4}\left(\kappa^{\pounds}(X_1, X_2) + \kappa^{\pounds}(X_2, X_1) + \kappa^{\pounds}(X_1^c, X_2^c) + \kappa^{\pounds}(X_2^c, X_1^c)\right), \\ SS_2(C_1, C_2) &= \frac{\kappa_1(X_1, X_2) + \kappa_1(X_2, X_1) - 1}{3 - (\kappa_1(X_1, X_2) + \kappa_1(X_2, X_1))}, \\ SS_3(C_1, C_2) &= \sqrt{\kappa^{\pounds}(X_1, X_2)\kappa^{\pounds}(X_2, X_1)\kappa^{\pounds}(X_1^c, X_2^c)\kappa^{\pounds}(X_2^c, X_1^c)}. \end{aligned} \tag{22}$$

Thus, $SS_1(C_1, C_2)$ (resp., $SS_3(C_1, C_2)$) is the arithmetical (geometrical) mean of $\kappa^{\pounds}(X_1, X_2)$, $\kappa^{\pounds}(X_2, X_1)$, $\kappa^{\pounds}(X_1^c, X_2^c)$, and $\kappa^{\pounds}(X_2^c, X_1^c)$.

Jaccard (1908). The similarity index J with range $[0, 1]$ goes back to P. Jaccard:

$$J(C_1, C_2) \stackrel{\text{def}}{=} \frac{a}{a+b+c} \tag{23}$$

It can be shown that

$$J(C_1, C_2) = \kappa_1(X_1, X_2) + \kappa_1(X_2, X_1) - 1. \tag{24}$$

Sokal and Michener (1958), Rand (1971). The similarity index R with range $[0,1]$ was introduced by R. R. Sokal and C. D. Michener, and later independently by W. Rand:

$$R(C_1, C_2) \stackrel{\text{def}}{=} \frac{a+d}{M} \tag{25}$$

The index R can be rewritten to

$$R(C_1, C_2) = \kappa_2(X_1, X_2) + \kappa_2(X_2, X_1) - 1. \tag{26}$$

Hamann (1961), Hubert (1977). The similarity index H, ranging over $[-1,1]$, was proposed by U. Hamann and independently by L. J. Hubert:

$$H(C_1, C_2) \stackrel{\text{def}}{=} \frac{(a+d)-(b+c)}{M} \tag{27}$$

By certain transformations we obtain

$$H(C_1, C_2) = 2(\kappa_2(X_1, X_2) + \kappa_2(X_2, X_1)) - 3. \tag{28}$$

Czekanowski (1932), Dice (1945), Gower and Legendre (1986). The similarity index CZ ranges over $[0,1]$. It was proposed by J. Czekanowski in 1932, L. R. Dice in 1945, and by J. C. Gower and P. Legendre in 1986:

$$CZ(C_1, C_2) \stackrel{\text{def}}{=} \frac{2a}{2a+b+c} \tag{29}$$

On can prove the following:

$$CZ(C_1, C_2) = \frac{2(\kappa_1(X_1, X_2) + \kappa_1(X_2, X_1) - 1)}{\kappa_1(X_1, X_2) + \kappa_1(X_2, X_1)} \tag{30}$$

Russel and Rao (1940). The similarity index RR ranges over $[0,1]$ and is attributed to P. F. Russel and T. R. Rao:

$$RR(C_1, C_2) \stackrel{\text{def}}{=} \frac{a}{M} \tag{31}$$

In this case we obtain that

$$RR(C_1, C_2) = \kappa^{\mathcal{L}}(U, X_1 \cap X_2) = \kappa_2(U, X_1 \cap X_2). \tag{32}$$

Fager and McGowan (1963). The similarity index FMG with range $[-1/2, 1)$ goes back to E. W. Fager and J. A. McGowan :

$$FMG(C_1, C_2) \stackrel{\text{def}}{=} \frac{a}{\sqrt{(a+b)(a+c)}} - \frac{1}{2\sqrt{a+b}} \tag{33}$$

The above formula can be expressed in the following way:

$$FMG(C_1, C_2) = \sqrt{\kappa^{\mathcal{L}}(X_1, X_2)\kappa^{\mathcal{L}}(X_2, X_1)} - \frac{1}{2}\sqrt{\kappa^{\mathcal{L}}(X_1, \{u\})} \tag{34}$$

for an arbitrary $u \in X_1$.

Sokal and Sneath (1963), Gower and Legendre (1986). The similarity index GL with range $[0,1]$ was introduced by R. R. Sokal and P. H. Sneath in 1963, and again by J. C. Gower and P. Legendre in 1986:

$$GL(C_1,C_2) \stackrel{\text{def}}{=} \frac{a+d}{a+\frac{1}{2}(b+c)+d} \tag{35}$$

A characterization of GL in terms of κ_2 is the following:

$$GL(C_1,C_2) = \frac{2(\kappa_2(X_1,X_2)+\kappa_2(X_2,X_1)-1)}{\kappa_2(X_1,X_2)+\kappa_2(X_2,X_1)} \tag{36}$$

Rogers and Tanimoto (1960). The similarity index RT with range $[0,1]$ is attributed to D. J. Rogers and T. T. Tanimoto:

$$RT(C_1,C_2) \stackrel{\text{def}}{=} \frac{a+d}{a+2(b+c)+d} \tag{37}$$

This index can be rewritten to the following form:

$$RT(C_1,C_2) = \frac{\kappa_2(X_1,X_2)+\kappa_2(X_2,X_1)-1}{3-(\kappa_2(X_1,X_2)+\kappa_2(X_2,X_1))} \tag{38}$$

Yule (1927), Goodman and Kruskal (1954). The similarity index GK ranges over $[-1,1]$. It was proposed by G. U. Yule in 1927, and again by L. A. Goodman and W. H. Kruskal in 1954:

$$GK(C_1,C_2) \stackrel{\text{def}}{=} \frac{ad-bc}{ad+bc} \tag{39}$$

This index can be expressed in terms of the standard RIF as follows:

$$GK(C_1,C_2) = \frac{\kappa^{\mathcal{L}}(X_2,X_1)\kappa^{\mathcal{L}}(X_2^c,X_1^c)-\kappa^{\mathcal{L}}(X_2,X_1^c)\kappa^{\mathcal{L}}(X_2^c,X_1)}{\kappa^{\mathcal{L}}(X_2,X_1)\kappa^{\mathcal{L}}(X_2^c,X_1^c)+\kappa^{\mathcal{L}}(X_2,X_1^c)\kappa^{\mathcal{L}}(X_2^c,X_1)} \tag{40}$$

Baulieu (1989). The similarity indices B_1 and B_2 range over $[0,1]$ and $[-1/4,1/4]$, respectively. They were introduced by F. B. Baulieu in 1989:

$$\begin{aligned} B_1(C_1,C_2) &\stackrel{\text{def}}{=} \frac{M^2-M(b+c)+(b-c)^2}{M^2}, \\ B_2(C_1,C_2) &\stackrel{\text{def}}{=} \frac{ad-bc}{M^2}. \end{aligned} \tag{41}$$

As in all previous cases, a RIF (precisely, κ_2 here) underlies the definitions of these similarity indices, viz.,

$$\begin{aligned} B_1(C_1,C_2) &= \kappa_2(X_1,X_2)+\kappa_2(X_2,X_1)-1+(\kappa_2(U,X_1)-\kappa_2(U,X_2))^2, \\ B_2(C_1,C_2) &= (1-\kappa_2(X_1,X_2^c))\kappa_2(U,X_1^c)-(1-\kappa_2(X_1^c,X_2^c))\kappa_2(U,X_1). \end{aligned} \tag{42}$$

4 Final Remarks

The main goal realized in this paper was to show that a pretty vast number of various similarity indices known from the literature can be formulated in terms of some rough inclusion functions. Rough inclusion functions (RIFs) are mappings, inspired by the notion of a rough inclusion introduced by L. Polkowski and A. Skowron as a basic concept of rough mereology, by means of which one can measure the degree of inclusion of a set of objects in a set of objects. Since information granules can be viewed as particular sets of objects, RIFs are important not only for the rough set theory but also for granular computing.

Starting with the standard RIF $\kappa^{\pounds}$ and two other RIFs of a similar origin, denoted by κ_1 and κ_2, we have obtained all 22 similarity indices discussed in [25]. In the paper just mentioned it is proved that the indices K and MC are equivalent after some correction known as the correction for agreement due to chance, and the same holds for R, H, and CZ. We have not referred to this question because we are interested in other aspects concerning similarity indices. For example, we think about a usage of similarity indices in granular computing to calculate the degree of similarity between compound information granules such as indistinguishability relations and tolerance relations on a set of elementary objects considered. Let us note that similarity indices can also be used in granular computing in a more general setting, viz. to compute the degree of similarity between arbitrary sets of objects.

In the full version of this article we will give an illustrating example and proofs of the formulas characterizing the similarity indices considered. In the future research we will generalize our results, viz. we will propose general schemata for generation of similarity indices from an arbitrary RIF. Another question, also suggested by the referee, is the discovery of relationships among RIFs and quality measures for clusters.

References

1. Polkowski, L.: Reasoning by Parts: An Outline of Rough Mereology, Warszawa (2011)
2. Polkowski, L., Skowron, A.: Rough mereology. Lecture Notes in Artificial Intelligence **869** (1994) 85–94
3. Polkowski, L., Skowron, A.: Rough mereology: A new paradigm for approximate reasoning. Int. J. Approximated Reasoning **15**(4) (1996) 333–365
4. Polkowski, L., Skowron, A.: Rough mereological calculi of granules: A rough set approach to computation. Computational Intelligence **17**(3) (2001) 472–492
5. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning About Data. Kluwer, Dordrecht (1991)
6. Leśniewski, S.: Foundations of the General Set Theory 1 (in Polish). Volume 2 of Works of the Polish Scientific Circle., Moscow (1916) Also in [7], pages 128–173.
7. Surma, S.J., Srzednicki, J.T., Barnett, J.D., eds.: Stanisław Leśniewski Collected Works. Kluwer/Polish Scientific Publ., Dordrecht/Warsaw (1992)
8. Zadeh, L.A.: Outline of a new approach to the analysis of complex system and decision processes. IEEE Trans. on Systems, Man, and Cybernetics **3** (1973) 28–44

9. Pawlak, Z., Skowron, A.: Rudiments of rough sets. Information Sciences **177**(1) (2007) 3–27
10. Pedrycz, W., Skowron, A., Kreinovich, V., eds.: Handbook of Granular Computing. John Wiley & Sons, Chichester (2008)
11. Stepaniuk, J.: Rough-Granular Computing in Knowledge Discovery and Data Mining. Springer-V., Berlin Heidelberg (2008)
12. Pawlak, Z., Skowron, A.: Rough membership functions. In Fedrizzi, M., Kacprzyk, J., Yager, R.R., eds.: Fuzzy Logic for the Management of Uncertainty. John Wiley & Sons, New York (1994) 251–271
13. Skowron, A., Stepaniuk, J.: Tolerance approximation spaces. Fundamenta Informaticae **27**(2–3) (1996) 245–253
14. Stepaniuk, J.: Knowledge discovery by application of rough set models. In: [34]. (2001) 137–233
15. Ziarko, W.: Variable precision rough set model. J. Computer and System Sciences **46**(1) (1993) 39–59
16. Ziarko, W.: Probabilistic decision tables in the variable precision rough set model. Computational Intelligence **17**(3) (2001) 593–603
17. Yao, Y.Y.: Decision-theoretic rough set models. Lecture Notes in Artificial Intelligence **4481** (2007) 1–12
18. Yao, Y.Y., Wong, S.K.M.: A decision theoretic framework for approximating concepts. Int. J. of Man–Machine Studies **37**(6) (1992) 793–809
19. Tsumoto, S.: Modelling medical diagnostic rules based on rough sets. Lecture Notes in Artificial Intelligence **1424** (1998) 475–482
20. Gomolińska, A.: Satisfiability and meaning of formulas and sets of formulas in approximation spaces. Fundamenta Informaticae **67**(1–3) (2005) 77–92
21. Nguyen, H.S., Skowron, A., Stepaniuk, J.: Granular computing: A rough set approach. Computational Intelligence **17**(3) (2001) 514–544
22. Cios, K.J., Pedrycz, W., Swiniarski, R.W., Kurgan, L.A.: Data Mining: A Knowledge Discovery Approach. Springer Science + Business Media, LLC, New York (2007)
23. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Computing Surveys **31**(3) (1999) 264–323
24. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, Chichester (1990)
25. Albatineh, A.N., Niewiadomska-Bugaj, M., Mihalko, D.: On similarity indices and correction for chance agreement. J. of Classification **23** (2006) 301–313
26. Gomolińska, A.: On three closely related rough inclusion functions. Lecture Notes in Artificial Intelligence **4585** (2007) 142–151
27. Gomolińska, A.: On certain rough inclusion functions. Transactions on Rough Sets IX: journal subline of LNCS **5390** (2008) 35–55
28. Gomolińska, A.: Rough approximation based on weak q-RIFs. Transactions on Rough Sets X: journal subline of LNCS **5656** (2009) 117–135
29. Xu, Z.B., Liang, J.Y., Dang, C.Y., Chin, K.S.: Inclusion degree: A perspective on measures for rough set data analysis. Information Sciences **141** (2002) 227–236
30. Zhang, W.X., Leung, Y.: Theory of including degrees and its applications to uncertainty inference. In: Proc. of 1996 Asian Fuzzy System Symposium. (1996) 496–501
31. Borkowski, L., ed.: Jan Łukasiewicz – Selected Works. North Holland/Polish Scientific Publ., Amsterdam/Warsaw (1970)
32. Łukasiewicz, J.: Die logischen Grundlagen der Wahrscheinlichkeitsrechnung, Kraków (1913) English translation in [31], pages 16-63.

33. Drwal, G., Mrózek, A.: System RClass – software implementation of a rough classifier. In Kłopotek, M.A., Michalewicz, M., Raś, Z.W., eds.: Proc. 7th Int. Symp. Intelligent Information Systems (IIS'1998), Malbork, Poland, June 1998. (1998) 392–395
34. Polkowski, L., Tsumoto, S., Lin, T.Y., eds.: Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Physica V., Heidelberg New York (2001)

Efficient Rough Set Theory Merging

Adam Grabowski

Institute of Informatics
University of Białystok
ul. Akademicka 2
15-267 Białystok, Poland
adam@math.uwb.edu.pl

Abstract. Theory exploration is a term describing the development of a formal (i.e. with the help of an automated proof-assistant) approach to selected topic, usually within mathematics or computer science. This activity however usually doesn't reflect the view of science considered as a whole, not as separated islands of knowledge. Merging theories essentially has its primary aim of bridging these gaps between specific disciplines.
As we provided formal apparatus for basic notions within rough set theory (as e.g. approximation operators and membership functions), we try to reuse the knowledge which is already contained in available repositories of computer-checked mathematical knowledge, or which can be obtained in a relatively easy way. We can point out at least three topics here: topological aspects of rough sets – as approximation operators have properties of the topological interior and closure; lattice-theoretic approach giving the algebraic viewpoint (e.g. Stone algebras); possible connections with formal concept analysis.
In such a way we can give the formal characterization of rough sets in terms of topologies or orders. Although fully formal, still the approach can be revised to keep the uniformity all the time.

Keywords: rough sets, knowledge management, formal mathematics.

1 Introduction

The era of extensive use of computers brought also an evolution of the mathematicians' work. Among new possibilities offered by computers we can point out the better transfer of knowledge between researchers via repositories of knowledge. Such computer algebra tools as Mathematica or MathCAD are very popular nowadays; researchers can also develop their own specialized software for computing relatively easier than before. The possibility of enhancing human work using automated proof assistants should be also underlined. We try to disscuss some issues concerned with the latter activity, concentrating on formalizing not only selected fields; but viewing specific disciplines from a wider perspective.

As we provided formal apparatus for basic notions within rough set theory (as e.g. approximation operators and membership functions), we try to reuse

the knowledge which is already contained in available repositories of computer-checked mathematical knowledge, or which can be obtained in a relatively easy way. We can point out at least three topics here: topological aspects of rough sets – as approximation operators have properties of the topological interior and closure; possible connections with formal concept analysis; lattice-theoretic approach giving the algebraic viewpoint (e.g. Stone algebras).

Our main aim is to develop (i.e. to describe in the formal computer language to be used within the repository of the existing mathematical knowledge) concrete examples of such formal knowledge reuse on the area of rough set theory. We also discuss some issues concerned with our implementation, but as we offer more than purely theoretical considerations (actual implementation is given), hence the word 'efficient' in the paper's title.

The structure of the paper is as follows: in the next section we present the overall methodological background for our work while in the third we focus on the activity of putting formal things together, called *merging theories*. Then we describe briefly the formal approach to rough sets we developed and some examples of successful, although not yet fully reused, bridging between various fields of formal mathematics.

2 Mathematical Knowledge Management

"Computer certification" is a relatively new term describing the process of the formalization via rewriting the text in a specific manner, usually in a rigorous language. Now this idea, although rather old (taking Peano, Whitehead and Russell as protagonists), gradually obtains a new life. As the tools evolved, the new paradigm was established: computers can potentially serve as a kind of oracle to check if the text is really correct. And then, the formalization is not *l'art pour l'art,* but it extends perspectives of knowledge reusing. The problem with computer-driven formalization is that it draws the attention of researchers somewhere at the intersection of mathematics and computer science, and if the complexity of the tools will be too high, only software engineers will be attracted and all the usefulness for an ordinary mathematician will be lost. But here, at this border, where there are the origins of MKM – Mathematical Knowledge Management, the place of fuzzy sets can be also. To give more or less formal definition, according to Wiedijk [26], *the formalization* can be seen presently as "the translation into a formal (i.e. rigorous) language so computers check this for correctness."

In this era of digital information anyone is free to choose his own way; to quote Vladimir Voevodsky, Fields Medal winner's words: "Eventually I became convinced that the most interesting and important directions in current mathematics are the ones related to the transition into a new era which will be characterized by the widespread use of automated tools for proof construction and verification". However he is focused as of now on the constructive Martin-Löf type theory many ordinary mathematicians aren't really familiar with. On the other hand, if we take into account famous Four Colour Theorem, automated

tools can really enable making some significant part of proofs, so hard to discuss with this opinion.

Among many available systems which serve as a proof-assistant we have chosen Mizar. The Mizar system [11] consists of three parts – the formal language, the software, and the database. The latter, called Mizar Mathematical Library (MML for short) established in 1989 is considered one of the largest repositories of computer checked mathematical knowledge. The basic item in the MML is called a Mizar article. It reflects roughly a structure of an ordinary paper, being considered at two main layers – the declarative one, where definitions and theorems are stated and the other one – proofs. Naturally, although the latter is the larger, the earlier needs some additional care.

As lattice theory (steered by Trybulec, Białystok, Poland) and functional analysis (led by Shidama, Nagano, Japan) are the most developed disciplines within the MML, further codification of rough sets, especially including their lattice-theoretic flavour, looks very promising. As a by-product, apart of readability of the Mizar language, we obtain also the presentation of the source accessible to ordinary mathematicians: pure HTML form with clickable links to corresponding notions and theorems.

3 Merging Theories

Theory exploration is a term describing the development of a formal (i.e. with the help of an automated proof-assistant) approach to selected topic, usually within mathematics or computer science. This activity however usually doesn't reflect the view of science considered as a whole, not as separated islands of knowledge. Merging theories essentially has its primary aim of bridging these gaps between specific disciplines. Of course, even digging deep in the area of selected discipline, eventually one have to use the apparatus from another field (usually category theory sheds some light), but this touches the informal layer, where interpretations can be somehat flexible.

In our CS&P 2012 paper [9] we have shown our translation of Zhu's paper about connections between ordinary properties of binary relations and underlying properties of rough approximation operators which proves some usefulness of proof assistants within a single area of research (essentially just the field of binary relations), but it is known that e.g., category theory gives nice interpretations for various questions; the same goes for modal logics. From another viewpoint, lattices can deliver similarly useful interpretations. Many fields can be reused depending on the author's preferred selected approach. Even in rough approach one can prefer either P-sets or I-sets (equivalence classes or just pairs) reflecting in the chosen language – either of ordinary sets (partitions) or subsets of Cartesian product.

We can consider merging on two levels:

Structures – when we inherit the overall signature of the object (as we can tell that groups are predecessors of rings or fields);

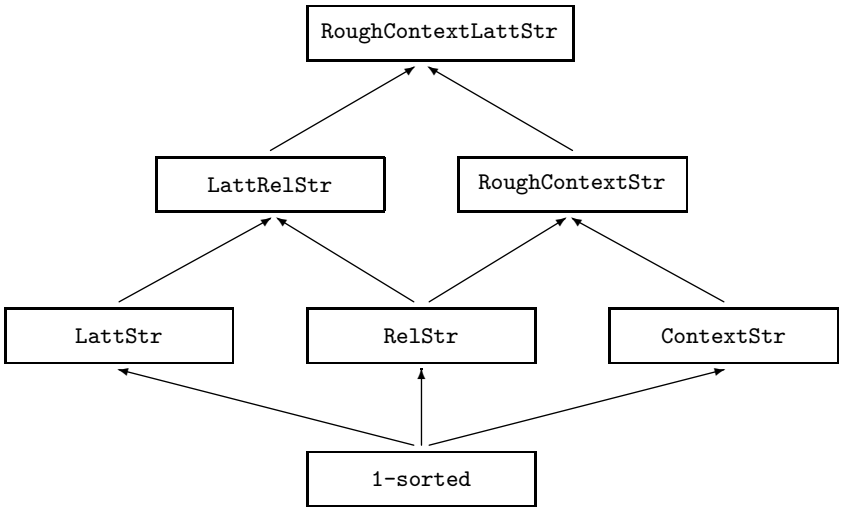


Fig. 1. The net of structures for chosen theories

Adjectives – when the hierarchy of axioms is described; here the example is that all Boolean algebras are Stone algebras.

Although from the informal point of view both given examples seem to be just the correspondence between axiom sets, formally this issue should be considered more deeply.

First of all, there are automatic theorem provers operating on the form of an equational characterization (collection of identities) of the theory. Hence the formula binding distinct items from a given signature gives more possibilities than the axiom postulating the existence of an object (even if we don't take into account Birkhoff variety theorem; equationally definable classes of mathematical structures are hereditary, admit homomorphic images and admit products – they form a variety). Good illustrative example here is the treatment of Boolean rings and Boolean algebras; we can see them as subvarieties of each other but formallywe should cope somehow with different signatures both are defined on. The same problem apears in the case of lattices viewed on the one hand as structures with join and meet operations or posets, otherwise. One can freely define lattices as posets with the existence of binary joins and meets; hence we obtain the algebraic interpretation of a lattice used, e.g. in universal algebra. Obviously both definitions are equivalent, buth they are definitely not the same as the order-theoretic one uses the signature

$$\langle L, \leq \rangle$$

while the algebraic one takes

$$\langle L, \sqcup, \sqcap \rangle$$

with binary operations: join $\sqcup$ and meet $\sqcap$.

Taking into account the aforementioned two stages of merging – on the level of structures both have really little in common as only the carrier L can be identical (we can call it a kind of syntactical point of view). But the latter viewpoint (of universal algebra) can give a path to semilattices $\langle L, \sqcup \rangle$ and $\langle L, \sqcap \rangle$ and here the second level of merging (semantical) really makes sense. Namely, on the signature of lower semilattice we can give an axiom of $\sqcap$-commutativity or $\sqcap$-associativity which can be then used on all its descendants. Both identities can be expressed as adjectives binded with appropriate structures.

The extensive use of identities in the form of attributes is really close to standard manipulation of axioms, so the example of the connection between Boolean and Stone lattices is really illlustrative here: as we work on the common signature $\langle L, \sqcup, \sqcap, ', 0, 1 \rangle$, there is no need to extend the corresponding structures and the work really depends on the deductive power of proof assistant (and computers do some computations which is quite natural).

Of course, the term 'formal' or 'formally' is used in this paper in two threads: on the one hand, 'formal' means the strict description of the rules governing the theory – in common use, it is 'rigorous' method. But hence all mathematics should be called formal in this sense, and this adjective should not then be used at all. There is also another interpretation of this attribute, which stems from Hilbert's formalism. In the latter view, computer assistance is the recent emerging trend which can be really controversial from the pen-and-paper mathematician viewpoint as the mathematics developed without machines for ages. Many computer scientists and mathematical intuitionists really advocate this approach, as Voevodsky who was quoted before.[1]

4 Rough Sets

Originally, we dealt with the more often used and methodologically simpler approach, i.e. equivalence relations-based rough sets. One of the key issues was also the possibility of further reusing, but soon this was automatically generalized. The concept of an information system can be also formalized as the descendant of the approximation space in a natural way. At the first sight, the underlying Mizar structure is `RelStr`, which has two fields: the `carrier` and the `InternalRel`, that is a binary relation of the `carrier`. The theory of relational structures has been developed and improved mainly during formalization of the *Compendium of Continuous Lattices* (which is described in [1] in detail). While in this context `RelStr` was used with attributes `reflexive transitive` and `antisymmetric` to establish posets, we decided to reuse it in our own way. First, we defined two new attributes: `with_equivalence` and `with_tolerance` which state that the `InternalRel` of the underlying `RelStr` is an equivalence resp. a tolerance relation (where a tolerance relation is a total reflexive symmetric relation, see [20]). With such defined notions, the basic definitions are as follows:

[1] Thanks go the anonymous referee for pointing out this inconsequence.

```
definition
  mode Approximation_Space is with_equivalence non empty RelStr;
  mode Tolerance_Space is with_tolerance non empty RelStr;
end;
```

Formalized theories can be treated as objects (axioms, definitions, theorems) clustered by certain relations based on information flow. The more atomic the notions are, the more is their usefulness. Driven by this idea we tried to drop selected properties of the equivalence relations. Our first choice was transitivity – therefore the use of tolerance spaces – as it seemed to be less substantial than the other two. The generalization work went rather smoothly. As we discovered soon, similar investigations, but without any machine-motivations, were done by Järvinen [14].

5 Formal Concept Analysis

Formal context analysis (FCA for short) has been introduced by Wille [27] as a formal tool for the representation and analysis of data. The main idea is to consider not only data objects, but to take into account properties (attributes) of the objects also. This leads to the notion of a *concept* which is a pair of a set of objects and a set of properties. In a concept all objects possess all the properties of the concept and vice versa. Thus the building blocks in FCA are given by both objects and their properties following the idea that we distinguish sets of objects by a common set of properties.

In the framework of FCA the set of all concepts (for given sets of objects and properties) constitutes a complete lattice. Thus based on the lattice structure the given data – that is its concepts and concept hierarchies – can be computed, visualized, and analyzed. In the area of software engineering FCA has been successfully used to build intelligent search tools as well as to analyze and reorganize the structure of software modules and software libraries. In the literature a number of extensions of the original approach can be found. So, for example, multi-valued concept analysis where the value of features is not restricted to two values (true and false). Also more involved models have been proposed taking into account additional aspects of knowledge representation such as different sources of data or the inclusion of rule-based knowledge in the form of ontologies.

Being basically an application of lattice theory FCA is a well-suited topic for machine-oriented formalization. On the one hand it allows to investigate the possibilities of reusing an already formalized lattice theory. On the other hand it can be the starting point for the formalization of the extensions mentioned above. In the following we briefly present the Mizar formalization of the basic FCA notions. The starting point is a formal context giving the objects and attributes of concern. Formally such a context consists of two sets of objects O and attributes A, respectively. Objects and attributes are connected by an incidence relation $I \subseteq O \times A$. The intension is that object $o \in O$ has property $a \in A$ if and only if $(o, a) \in I$. In Mizar [23] this has been modelled by the following structure definitions.

```
definition
  struct 2-sorted (# Objects, Attributes -> set #);
end;

definition
  struct (2-sorted) ContextStr
    (# Objects, Attributes -> set,
       Information -> Relation of the Objects,the Attributes #);
end;
```

Now a formal context is a non-empty `ContextStr`. To define formal concepts in a given formal context C two derivation operators `ObjectDerivation(C)` and `AttributeDerivation(C)` are used. For a set O of objects (A of attributes) the derived set consists of all attributes a (objects o) such that $(o, a) \in I$ for all $o \in O$ (for all $a \in A$). The Mizar definition of these operators is straightforward and omitted here.

A formal concept FC is a pair (O, A) where O and A respect the derivation operators: the derivation of O contains exactly the attributes of A, and vice versa. O is called the extent of FC, A the intent of FC. In Mizar this gives rise to a structure introducing the `extent` and the `intent` and an attribute `concept-like`.

```
definition let C be 2-sorted;
  struct ConceptStr over C
    (# Extent -> Subset of the Objects of C,
       Intent -> Subset of the Attributes of C #);
end;

definition let C be FormalContext;
           let CP be ConceptStr over C;
  attr CP is concept-like means  :: CONLAT_1:def 13
    (ObjectDerivation(C)).(the Extent of CP) = the Intent of CP &
    (AttributeDerivation(C)).(the Intent of CP) = the Extent of CP;
end;

definition let C be FormalContext;
  mode FormalConcept of C is concept-like non empty ConceptStr over C;
end;
```

Formal concepts over a given formal context can be easily ordered: a formal concept FC_1 is more specialized (and less general) than a formal concept FC_2 iff the extent of FC_1 is included in the extent of FC_2 (or equivalently iff the intent of FC_2 is included in the intent of FC_1). With respect to this order the set of all concepts over a given formal context C forms a complete lattice, the concept lattice of C.

```
theorem
  for C being FormalContext holds ConceptLattice(C) is complete Lattice;
```

This theorem, among others, has been proven in [23]. The formalization of FCA in Mizar went rather smoothly, the main reason being that lattice theory has already been well developed. Given objects, attributes and an incidence relation between them, this data can now be analyzed by inspecting the structure of the (concept) lattice; see [27, 7] for more details and techniques of formal concept analysis.

6 Rough Concept Analysis

In this section we present issues concerning the merging of concrete theories in the Mizar system. We will illustrate them by living examples from Rough Concept Analysis done in Mizar and skipping most technical details (this part is an extension of [12]). For details of used type system, see [11, 2]. We like to mention that in the course of FCA formalization the formal apparatus yet existing in the Mizar Mathematical Library also had to be improved and cleaned up.

A basic structure for the merged theory should inherit fields from its ancestors, which would be hard to implement if structures were implemented as ordered tuples (multiple copies of the same selector, inadequate ordering of fields in the result). The more feasible realization is by partial functions rather, and that is the way Mizar structures work.

```
definition
  struct (ContextStr, RelStr) RoughContextStr
    (# carrier, carrier2 -> set,
       Information -> Relation of the carrier, the carrier2,
       InternalRel -> Relation of the carrier #);
end;
```

As it often happens, an extension of the theory to another need not be unique. There are at least three different methods of adding roughness to formal concepts [15, 22]. The question which approach to choose depends on the author. The notion of a free structure in a class of descendant type conservative with respect to the original object is very useful.

```
definition let C be ContextStr;
  mode RoughExtension of C -> RoughContextStr means
    the ContextStr of it = the ContextStr of C;
end;
```

Now, if C is a given context, we can introduce roughness in many different ways by adjectives.

Up to now, we described only mechanisms of independent inheritance of notions. Within the merged theory it is necessary to define connections between its source ingredients. Here the attributes describing mutual interferences between selectors from originally disjoint theories proved their enormous value. They may determine the set of properties of a free extension.

```
definition let C be RoughFormalContext;
  attr C is naturally_ordered means
    for x, y being Element of C holds
    [x,y] in the InternalRel of C iff
      (ObjectDerivation C).{x} = (ObjectDerivation C).{y};
end;
```

Since the relation from the definiens above is an equivalence relation on the objects of C and hence determines a partition of the set of objects of C into the so-called *elementary sets*, it is a constructor of an approximation space induced by given formal context.

Theory merging makes no sense, if proving the same theorem would be necessary within both source and target theory. Since a new Mizar type called `RoughFormalContext` is defined analogously to the notion of `FormalContext`, as `non quasi-empty RoughContextStr`, the following Fundamental Theorem of RCA is justified only by the Fundamental Theorem of FCA. Even more, clusters providing automatic acceptance of the original theorems do it analogously within target theory. That is also a workplace for clusters `rough` and `exact` from the core rough set theory.

```
for C being RoughFormalContext holds
  ConceptLattice(C) is complete Lattice by CONLAT_1:48;
```

7 Topological Spaces and Partitions

Of course, there are cases we shouldn't even change the language when switching between various fields of mathematics. An illustrative example here is again the notion of rough sets in its primal setting. When we see at the approximation space given by an equivalence relation, it is quite natural to consider just classes of abstractions forgetting about original relation. Hence, the lattice of such objects can be defined:

```
definition
  let X be set;
  func EqRelLatt X -> strict Lattice means
:: MSUALG_5:def 2
  the carrier of it = { x where x is Relation of X,X :
    x is Equivalence_Relation of X } &
  for x,y being Equivalence_Relation of X holds
    (the L_meet of it).(x,y) = x /\ y &
    (the L_join of it).(x,y) = x "\/" y;
end;
```

Among many interesting properties which were proven about this structure we can quote its completeness, for example:

```
registration
  let A be set;
  cluster EqRelLATT A -> complete;
end;
```

The natural definition of the topological space is that we have a family of open sets called the topology, τ. Then a topological space can be considered as a pair consisting of the universe X and the topology τ defined on the subsets of X if τ satisfies the axioms of topology. As they are widely known, informally, we quote below only a formal counterpart of it:

```
definition
  struct (1-sorted) TopStruct
    (# carrier -> set,
      topology -> Subset-Family of the carrier
     #);
end;
```

reflecting the bare $\langle X, \tau \rangle$ tuple and

```
definition
  let IT be TopStruct;
  attr IT is TopSpace-like means
:: PRE_TOPC:def 1
  the carrier of IT in the topology of IT &
  (for a being Subset-Family of IT st a c= the topology of IT holds
    union a in the topology of IT) &
   for a,b being Subset of IT st
     a in the topology of IT & b in the topology of IT holds
      a /\ b in the topology of IT;
end;
```

as axiomatic description of τ.

Then a topological space is just the structure `TopStruct` to which the adjective `TopSpace-like` can be added. As usual, with every such object we can associate the closure and the interior operators, with axioms in Kuratowski style and then the existing apparatus of topological spaces (`Cl` and `Int` for the closure and interior, respectively) can be reused.

8 Conclusions

Even if we are aware that this paper is really an emerging work and most technicalities were really skipped (but they can of course tracked in corresponding Mizar source files freely available from the project homepage), there are some its clear advantages – considering the repository of formalized mathematical knowledge as a whole extends our knowledge. Some of the ideas contained in this paper

are dated back to 2004 and our paper [12] presented at the International Conference in Mathematical Knowledge Management where some of the problems were only identified, but until now many new tools were developed and many interesting new topics were formalized.

Quoting Pawlak's own words about the role of computers (or *mathematical machines* as they were called):

> *"One can formulate a risky opinion that almost all contemporary mathematical theories in their current state cannot be automatically treated. Reformulating them is not an easy task. So, the question arises, to which extent the amount of work done can be justified by the importance of obtained results. (...) Automated discovery of new important results seems to us rather unlikely."*

Even if Pawlak's doubts about finding new theorems were clearly expressed, he was convinced that computers can help in a bit different way:

> *"(...) the view for theories which are already known, but from another viewpoint can shed some new light for the structure of mathematical theories and improve human creativity."*
> ([18], p. 142, translation ours).

We try to argue that the formalization (still having in mind the discussion on the (over)use of the word 'formal' from the end of the third section) of knowledge in the way accessible by computers is not the question of the sense; it is the question of time. Real efficiency of this activity will be shown by much more examples, much more work, and definitely by much more automation many proof assistants offer. We implemented in Mizar already three paths of rough set theory merging: with topology, formal concepts and lattices (including interval sets, which is formalized in [10]). Hence preliminary steps were already done and as this work makes no sense in the island of isolated knowledge, anyone is invited to contribute.

References

1. G. Bancerek, Development of the theory of continuous lattices in Mizar, in: M. Kerber and M. Kohlhase (eds.), *The Calculemus-2000 Symposium Proceedings,* pp. 65–80, 2001.
2. G. Bancerek, On the structure of Mizar types, in: H. Geuvers and F. Kamareddine (eds.), *Proc. of MLC 2003, ENTCS* 85(7), 2003.
3. B. Buchberger, Mathematical Knowledge Management in Theorema, in: B. Buchberger and O. Caprotti (eds.), *Proc. of MKM 2001,* Linz, Austria, 2001.
4. L. Cruz-Filipe, H. Geuvers, and F. Wiedijk, C-CoRN, the Constructive Coq Repository at Nijmegen, `http://www.cs.kun.nl/~freek/notes/`.
5. D. Dubois, H. Prade, Rough fuzzy sets and fuzzy rough sets, *International Journal of General Systems,* 17(2–3), 191–209, 1990.
6. W. Farmer, J. Guttman, and F. Thayer, Little theories, in: D. Kapur (ed.), *Automated Deduction – CADE-11, LNCS* 607, pp. 567–581, 1992.

7. B. Ganter and R. Wille, *Formal concept analysis – mathematical foundations,* Springer Verlag, 1998.
8. A. Grabowski, Basic properties of rough sets and rough membership function, *Formalized Mathematics,* 12(1), 21–28, 2004; can be tracked also under `http://mizar.org/version/current/html/roughs_1.html`.
9. A. Grabowski, Automated discovery of properties of rough sets, to appear in *Fundamenta Informaticae,* 2013.
10. A. Grabowski, M. Jastrzębska, On the lattice of intervals and rough sets, *Formalized Mathematics,* 17(4), 237–244, 2009.
11. A. Grabowski, A. Korniłowicz, A. Naumowicz, Mizar in a nutshell, *Journal of Formalized Reasoning,* 3(2), 153–245, 2010.
12. A. Grabowski, Ch. Schwarzweller, Rough Concept Analysis – theory development in the Mizar system, MKM 2004 Proceedings, LNCS, 3119, pp. 130–144, 2004.
13. A. Grabowski, Ch. Schwarzweller, Towards automatically categorizing mathematical knowledge, M. Ganzha, L. Maciaszek, and M. Paprzycki (Eds.), FedCSIS 2012 Proceedings, 63–68, 2012.
14. J. Järvinen, Approximations and rough sets based on tolerances, in: W. Ziarko and Y. Yao (eds.), *Proc. of RSCTC 2000, LNAI* 2005, pp. 182–189, 2001.
15. R.E. Kent, Rough Concept Analysis: a synthesis of rough sets and formal concept analysis, *Fundamenta Informaticae* 27(2–3), pp. 169–181, 1996.
16. T. Nipkow, L. Paulson, and M. Wenzel, Isabelle/HOL – a proof assistant for higher-order logic, *LNCS* 2283, 2002.
17. S. Owre and N. Shankar, Theory interpretations in PVS, Technical Report, NASA/CR-2001-211024, 2001.
18. Z. Pawlak: *Automatyczne dowodzenie twierdzeń,* Warsaw, PZWS, 1965 (Eng. *Automated theorem proving,* in Polish).
19. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data,* Kluwer, Dordrecht, 1991.
20. K. Raczkowski and P. Sadowski, Equivalence relations and classes of abstraction, *Formalized Mathematics,* 1(3), pp. 441–444, 1990.
21. P. Rudnicki and A. Trybulec, Mathematical Knowledge Management in Mizar, in: B. Buchberger and O. Caprotti (eds.), *Proc. of MKM 2001,* Linz, Austria, 2001.
22. J. Saquer and J.S. Deogun, Concept approximations based on rough sets and similarity measures, *International Journal on Applications of Mathematics in Computer Science,* 11(3), pp. 655–674, 2001.
23. C. Schwarzweller, Introduction to concept lattices, *Formalized Mathematics,* 7(2), pp. 233–242, 1998.
24. M. Strecker, Formal verification of a Java compiler in Isabelle, *Lecture Notes in Computer Science,* 2392, 63–77, 2002.
25. J. Urban, G. Sutcliffe, Automated reasoning and presentation support for formalizing mathematics in Mizar, *Lecture Notes in Computer Science,* 6167, 132–146, 2010.
26. F. Wiedijk, Formal proof – getting started, *Notices of the American Mathematical Society,* 55(11), 1408–1414, 2008.
27. R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, in: I. Rival (ed.), *Ordered Sets,* Reidel, Dordrecht-Boston, 1982.
28. Y.Y. Yao, A comparative study of fuzzy sets and rough sets, *Information Sciences,* 109(1-4), 227–242, 1998.

Opacity Testing *

Damas P. Gruska

Institute of Informatics, Comenius University,
Mlynska dolina, 842 48 Bratislava, Slovakia,
gruska@fmph.uniba.sk.

Abstract. Opacity testing is formalized and studied. We specify opacity testers as well as tested systems by (timed) process algebras. We model various testers according to how sophisticated observations of tested system they can make and which kind of conclusions they can obtain. We use this technique to define several realistic security properties. The properties are studied and compared with other security concepts.

Keywords: opacity, process algebras, information flow, security

1 Introduction

Several formulations of system security can be found in the literature. Many of them are based on non-interference (see [GM82]) which assumes an absence of any information flow between private and public systems activities. More precisely, systems are considered to be secure if from observations of their public activities no information about private activities can be deduced. This approach has found many reformulations for different formalisms, computational models and nature or "quality" of observations.

One of the most general notion is opacity (see [BKR04,BKMR06]) and many security properties can be viewed as its special cases (see, for example, [Gru07]). A predicate is opaque if for any trace of a system for which it holds there exists another trace for which it does not hold and both traces are indistinguishable for an observer. Opacity is widely studied also in process algebras framework. Here, as well as later in this paper, we mention those ones which are close to the the presented work. For example, in [Gru07,Gru12] opacity for very simple observations is studied for timed process algebra. In [Gru09] a quantification of opacity by means of the information theory is studied. In [Gru10,Gru12a] we defined security properties which could be described by specific relations on contexts. In general, opacity is an undecidable property even for very simple observation functions or predicates. On the other side, opacity is based on traces and hence inadequate for any finer "attacker" who is capable not only observe traces but also interact with systems.

The aim of this paper is twofold. On the one side, we weaken opacity by modeling both predicate and observations by processes (particularly, finite state

* Work supported by the grant VEGA 1/1333/12.

processes) and hence we obtain (polynomial time) decidable properties. On the other side, we strength opacity by defining simulation opacity which is not restricted to trace observations and which is stronger than opacity. While opacity of predicate is defined for a given process (and an observation function), simulation opacity requires (roygly speaking) that it is opaque also for every successor of the process. Moreover, our formalism of timed process algebra, allows us to express various types of timed attacks.

The paper is organized as follows. In Section 2 we describe the timed process algebra TPA which will be used as a basic formalism. In Section 3 we present opacity and in the next section simulation opacity is defined and studied.

2 Timed Process Algebra

In this section we define Timed Process Algebra, TPA for short. TPA is based on Milner's CCS but the special time action t which expresses elapsing of (discrete) time is added. The presented language is a slight simplification of Timed Security Process Algebra introduced in [FGM00]. We omit an explicit idling operator ι used in tSPA and instead of this we allow implicit idling of processes. Hence processes can perform either "enforced idling" by performing t actions which are explicitly expressed in their descriptions or "voluntary idling". But in the both cases internal communications have priority to action t in the case of the parallel operator. Moreover we do not divide actions into private and public ones as it is in tSPA. TPA differs also from the tCryptoSPA (see [GM04]). TPA does not use value passing and strictly preserves *time determinacy* in case of choice operator $+$ what is not the case of tCryptoSPA.

To define the language TPA, we first assume a set of atomic action symbols A not containing symbols τ and t, and such that for every $a \in A$ there exists $\overline{a} \in A$ and $\overline{\overline{a}} = a$. We define $Act = A \cup \{\tau\}, Actt = Act \cup \{t\}$. We assume that $a, b, \ldots$ range over A, $u, v, \ldots$ range over Act, and $x, y \ldots$ range over $Actt$. Assume the signature $\Sigma = \bigcup_{n\in\{0,1,2\}} \Sigma_n$, where

$$\begin{aligned}
\Sigma_0 &= \{Nil\} \\
\Sigma_1 &= \{x. \mid x \in A \cup \{t\}\} \cup \{[S] \mid S \text{ is a relabeling function}\} \\
&\quad \cup \{\backslash M \mid M \subseteq A\} \\
\Sigma_2 &= \{|, +\}
\end{aligned}$$

with the agreement to write unary action operators in prefix form, the unary operators $[S], \backslash M$ in postfix form, and the rest of operators in infix form. Relabeling functions, $S : Actt \to Actt$ are such that $\overline{S(a)} = S(\bar{a})$ for $a \in A, S(\tau) = \tau$ and $S(t) = t$.

The set of TPA terms over the signature Σ is defined by the following BNF notation:

$$P \ ::= \ X \ \mid \ op(P_1, P_2, \ldots P_n) \ \mid \ \mu X P$$

where $X \in Var$, Var is a set of process variables, $P, P_1, \ldots P_n$ are TPA terms, $\mu X-$ is the binding construct, $op \in \Sigma$.

The set of CCS terms consists of TPA terms without t action. We will use an usual definition of opened and closed terms where μX is the only binding operator. Closed terms which are t-guarded (each occurrence of X is within some subexpression $t.A$ i.e. between any two t actions only finitely many non timed actions can be performed) are called TPA processes. Note that Nil will be often omitted from processes descriptions and hence, for example, instead of $a.b.Nil$ we will write just $a.b$.

We give a structural operational semantics of terms by means of labeled transition systems. The set of terms represents a set of states, labels are actions from $Actt$. The transition relation $\rightarrow$ is a subset of TPA $\times$ $Actt$ $\times$ TPA. We write $P \xrightarrow{x} P'$ instead of $(P, x, P') \in \rightarrow$ and $P \not\xrightarrow{x}$ if there is no P' such that $P \xrightarrow{x} P'$. The meaning of the expression $P \xrightarrow{x} P'$ is that the term P can evolve to P' by performing action x, by $P \xrightarrow{x}$ we will denote that there exists a term P' such that $P \xrightarrow{x} P'$. We define the transition relation as the least relation satisfying the inference rules for CCS plus the following inference rules:

$$\frac{}{Nil \xrightarrow{t} Nil} \quad A1 \qquad \frac{}{u.P \xrightarrow{t} u.P} \quad A2$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \not\xrightarrow{\tau}}{P \mid Q \xrightarrow{t} P' \mid Q'} \; Pa \qquad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \; S$$

Here we mention the rules that are new with respect to CCS. Axioms $A1, A2$ allow arbitrary idling. Concurrent processes can idle only if there is no possibility of an internal communication (Pa). A run of time is deterministic (S). In the definition of the labeled transition system we have used negative premises (see Pa). In general this may lead to problems, for example with consistency of the defined system. We avoid these dangers by making derivations of τ independent of derivations of t. For an explanation and details see [Gro90]. Regarding behavioral relations we will work with the timed version of weak trace equivalence. Note that here we will use also a concept of observations which contain complete information which includes also τ actions and not just actions from A and t action as it is in [FGM00]. For $s = x_1.x_2.\ldots.x_n, x_i \in Actt$ we write $P \xrightarrow{s}$ instead of $P \xrightarrow{x_1}\xrightarrow{x_2} \ldots \xrightarrow{x_n}$ and we say that s is a trace of P. By ϵ we will denote the empty sequence of actions, by $Succ(P)$ we will denote the set of all successors of P. If the set $Succ(P)$ is finite we say that P is finite state.

Let $s \in Actt^\star$. By $|s|$ we will denote the length of s i.e. a number of action contained in s. By $s|_B$ we will denote the sequence obtained from s by removing all actions not belonging to B. For example, $|s|_{\{t\}}|$ denote a number of occurrences of t in s, i.e. time length of s.

To express what an observer can see from system behaviour we will define modified transitions $\stackrel{x}{\Rightarrow}_M$ which hide actions from M (as well as τ action). Formally, we will write $P \stackrel{x}{\Rightarrow}_M P'$ for $M \subseteq A$ iff $P \xrightarrow{s_1}\xrightarrow{x}\xrightarrow{s_2} P'$ for $s_1, s_2 \in (M \cup \{\tau\})^\star$

and $P \stackrel{s}{\Rightarrow}_M$ instead of $P \stackrel{x_1}{\Rightarrow}_M \stackrel{x_2}{\Rightarrow}_M \ldots \stackrel{x_n}{\Rightarrow}_M$. Instead of $\Rightarrow_\emptyset$ we will write $\Rightarrow$ and instead of $\Rightarrow_{\{h\}}$ we will write $\Rightarrow_h$. We will write $P \stackrel{x}{\Rightarrow}_M$ if there exists P' such that $P \stackrel{x}{\Rightarrow}_M P'$. We will write $P \stackrel{\widehat{x}}{\Rightarrow}_M P'$ instead of $P \stackrel{\epsilon}{\Rightarrow}_M P'$ if $x \in M$.

We conclude this section with definitions of variants of weak simulation and weak bisimulation.

Definition 1. Let (TOA, $Actt, \rightarrow$) be a labelled transition system (LTS). A relation $\Re \subseteq \text{CCS} \times \text{CCS}$ is called a *weak M-bisimulation* if it is symmetric and it satisfies the following condition: if $(P, Q) \in \Re$ and $P \stackrel{x}{\rightarrow} P', x \in Actt$ then there exists a process Q' such that $Q \stackrel{\widehat{x}}{\Rightarrow}_M Q'$ and $(P', Q') \in \Re$. Two processes P, Q are *M-bisimilar*, abbreviated $P \approx_M Q$, if there exists a strong bisimulation relating P and Q. If it is not required that relation $\Re$ is symmetric we call it M-simulation and we say that process P simulates process Q, abbreviated $P \prec_M Q$, if there exists a simulation relating P and Q.

We will write $\approx$ and $\prec$ instead of $\approx_M$ and $\prec_M$, respectively, if $M = \emptyset$.

3 Opacity

To formalize an information flow we do not divide actions into public and private ones at the system description level, as it is done for example in [GM04,BG04], but we use a more general concept of observation and opacity. This concept was exploited in [BKR04] and [BKMR06] in a framework of Petri Nets and transition systems, respectively.

First we define observation function on sequences from $Actt^\star$.

Definition 2 (Observation). *Let Θ be a set of elements called observables. Any function $\mathcal{O} : Actt^\star \rightarrow \Theta^\star$ is an observation function. It is called static /dynamic /orwellian / m-orwellian ($m \geq 1$) if the following conditions hold respectively (below we assume $w = x_1 \ldots x_n$):*

- *static if there is a mapping $\mathcal{O}' : Actt \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1) \ldots \mathcal{O}'(x_n)$,*
- *dynamic if there is a mapping $\mathcal{O}' : Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1).\mathcal{O}'(x_1.x_2) \ldots \mathcal{O}'(x_1 \ldots x_n)$,*
- *orwellian if there is a mapping $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1, w).\mathcal{O}'(x_2, w) \ldots \mathcal{O}'(x_n, w)$,*
- *m-orwellian if there is a mapping $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1, w_1).\mathcal{O}'(x_2, w_2) \ldots \mathcal{O}'(x_n, w_n)$ where $w_i = x_{max\{1,i-m+1\}}.x_{max\{1,i-m+1\}+1} \ldots x_{min\{n,i+m-1\}}$.*

In the case of the static observation function each action is observed independently from its context. In the case of the dynamic observation function an observation of an action depends on the previous ones, in the case of the orwellian and m-orwellian observation function an observation of an action depends on the all and on m previous actions in the sequence, respectively. The

static observation function is the special case of m-orwellian one for $m = 1$. Note that from the practical point of view the m-orwellian observation functions are the most interesting ones. An observation expresses what an observer - eavesdropper can see from a system behavior and we will alternatively use both the terms (observation - observer) with the same meaning.

Now suppose that we have some security property. This might be an execution of one or more classified actions, an execution of actions in a particular classified order which should be kept hidden, etc. Suppose that this property is expressed by predicate ϕ over process traces. We would like to know whether an observer can deduce the validity of the property ϕ just by observing sequences of actions from $Actt^\star$ performed by given process.

The observer cannot deduce the validity of ϕ if there are two traces $w, w' \in Actt^\star$ such that $\phi(w), \neg\phi(w')$ and the traces cannot be distinguished by the observer i.e. $\mathcal{O}(w) = \mathcal{O}(w')$. We formalize this concept by opacity.

Definition 3 (Opacity). *Given process P, a predicate ϕ over $Actt^\star$ is opaque w.r.t. the observation function $\mathcal{O}$ if for every sequence w, $w \in Tr(P)$ such that $\phi(w)$ holds and $\mathcal{O}(w) \neq \epsilon$, there exists a sequence $w', w' \in Tr(P)$ such that $\neg\phi(w')$ holds and $\mathcal{O}(w) = \mathcal{O}(w')$. The set of processes for which the predicate ϕ is opaque with respect to $\mathcal{O}$ will be denoted by $Op^\phi_\mathcal{O}$.*

The definition of opacity (see Definition 3) of predicate ϕ is asymmetric in the sense that if $\phi(w)$ does not hold than it is not required that there exists another trace for which it holds (in general $Op^\phi_\mathcal{O} \neq Op^{\neg\phi}_\mathcal{O}$). This means that opacity says something to an intruder which tries to detect only validity of ϕ (if it is opaque, than validity cannot be detected) but not its non-validity i.e. it says nothing about predicate $\neg\phi$. Hence we define strong variant of opacity.

Definition 4 (Strong Opacity). *Given process P, a predicate ϕ over $Actt^\star$ is strongly opaque w.r.t. the observation function $\mathcal{O}$ if for every sequence w, $w \in Tr(P)$ such that $\phi(w)$ holds and $\mathcal{O}(w) \neq \epsilon$, there exists a sequence $w', w' \in Tr(P)$ such that $\neg\phi(w')$ holds and $\mathcal{O}(w) = \mathcal{O}(w')$. Moreover, for for every sequence w, $w \in Tr(P)$ such that $\neg\phi(w)$ holds and $\mathcal{O}(w) \neq \epsilon$, there exists a sequence $w', w' \in Tr(P)$ such that $\phi(w')$ holds and $\mathcal{O}(w) = \mathcal{O}(w')$. The set of processes for which the predicate ϕ is opaque with respect to $\mathcal{O}$ will be denoted by $sOp^\phi_\mathcal{O}$.*

Lemma 1. *$sOp^\phi_\mathcal{O} \subseteq Op^\phi_\mathcal{O}$ for every ϕ and $\mathcal{O}$. Moreover, there exist ϕ and $\mathcal{O}$ such that $sOp^\phi_\mathcal{O} \subset Op^\phi_\mathcal{O}$.*

Proof. Main idea. Let $P \in sOp^\phi_\mathcal{O}$. Then for every trace of P for which ϕ holds there exists a trace indistinguishable by observation function $\mathcal{O}$ for which ϕ does not hold and hence $P \in Op^\phi_\mathcal{O}$. Let us consider process $P = h.l.Nil + l.Nil + l'.Nil$ and let ϕ holds for tracecs which contain action h and observation function $\mathcal{O}$ which hide h action. Then we have $P \in Op^\phi_\mathcal{O}$ but $P \notin sOp^\phi_\mathcal{O}$ and hence the inclusion is proper, i.e $sOp^\phi_\mathcal{O} \subset Op^\phi_\mathcal{O}$ for such ϕ and $\mathcal{O}$.

4 Simulation Opacity

We start with a motivation example. Let us consider process $P = l.h.l'.Nil + l.(h.l'.Nil + l'.Nil)$, an observation function which does not see action h and a predicate which holds for sequences containing h action. It is easy to check that this predicate is opaque in this setting. That means than an attacker which can observe traces of P cannot deduce whether action h has occurred or nor. On the other side for a "simulation attacker" i.e. the attacker which can not only observe traces but can interact with systems, the predicate is not "opaque" anymore. This is a natural consequence of simulation being more powerful then just a trace inclusion. Now we will extend the notion of opacity to reflect more powerful attackers than those ones which just observe traces or alternatively, predicate should be opaque not only for a given process P but also for every its successor.

Definition 5 (Simulation Opacity). *Given a set of processes $\Re$, predicate ϕ over $Actt^\star$ is simulation opaque for $\Re$ w.r.t. the observation function $\mathcal{O}$ if for every $P \in \Re$ if $P \xrightarrow{s} P'$ for such s that $\phi(s)$ holds and $o(s) \neq \epsilon$ then there exists s' such that $\neg\phi(s')$ holds, $\mathcal{O}(s) = \mathcal{O}(s')$ and $P \xrightarrow{s'}$ and moreover $P' \in \Re$. Predicate ϕ is simulation opaque for process P with respect to $\mathcal{O}$ (denoted $P \in SOp^{\phi}_{\mathcal{O}}$) if $P \in \Re$ for some simulation opaque $\Re$ with respect to ϕ and $\mathcal{O}$.*

Now let us return to process P and the predicate and the observation function from the beginning of this section. Now we can check that P is not simulation opaque in this setting. This is also the proof that an inclusion from the next proposition is proper.

Proposition 1. *$SOp^{\phi}_{\mathcal{O}} \subseteq Op^{\phi}_{\mathcal{O}}$ for every ϕ and $\mathcal{O}$. Moreover, there exist ϕ and $\mathcal{O}$ such that $SOp^{\phi}_{\mathcal{O}} \subset Op^{\phi}_{\mathcal{O}}$.*

Proof. The main idea. Let $P \in SOp^{\phi}_{\mathcal{O}}$. Then we have that for every trace s of P for which ϕ holds there exists another trace s' for which ϕ does not hold and both traces cannot be distinguished by $\mathcal{O}$. Hence $P \in Op^{\phi}_{\mathcal{O}}$. The example from of this section we see that the inclusion can be proper.

The simulation opacity is defined for arbitrary predicates and observation functions. Now we we will reformulate it for those ones which can be expressed by process algebras. Now we will model simulation opacity in a process algebra setting. Suppose that $Actt \cap \Theta = \{t\}$ and hence we extend the set of actions A by Θ. We combine a process which checks validity of ϕ with a process which computes observation function $\mathcal{O}$ into two process O_ϕ and $O_{\neg\phi}$. Now we define process O_ϕ.

Definition 6. *Process O_ϕ is called process definition of predicate ϕ and observation function $\mathcal{O}$ over sequences of actions if for every P it holds $(P|O_\phi) \setminus A \stackrel{o}{\Rightarrow} (P'|O_\phi) \setminus A$ iff $P \xrightarrow{s} P'$ such that $\phi(s)$ and $\mathcal{O}(s) = o$.*

Note that we expect that process O_ϕ makes some computation resulting on observable o and then it returns to the initial state (actually, to be more precise, we should write to the process bisimilar with it). Now we will define simulation opacity with respect to O_ϕ and $O_{\neg\phi}$ (see Fig. 1). Its definition is a reformulation of Definition 5 in process algebra setting.

Fig. 1. Testing scenario

Definition 7. *We say that process P is simulation opaque with respect to O_ϕ and $O_{\neg\phi}$ (denoted $P \in SO(O_\phi, O_{\neg\phi})$ iff $(P|O_\phi) \setminus A \prec (P|O_{\neg\phi}) \setminus A$.*

In fact, from the following proposition we see that both types of simulation opacity coincide for those predicated and observation functions which can be expressed by processes.

Proposition 2. *Let O_ϕ and $O_{\neg\phi}$ are process definitions of observation function $\mathcal{O}$ and predicates ϕ and $\neg\phi$, respectively. Then $SOp^\phi_\mathcal{O} = SO(O_\phi, O_{\neg\phi})$.*

Proof. The main idea. Process definition O_ϕ and $O_{\neg\phi}$ mimic both observations and predicates validity (see Definition 7). Moreover, the simulation $\prec$ reflects the fact that after each "step" the resulting process is again opaque and hence similation opaque.

Many trace based security properties can be viewed as special cases of opacity (see for example [Gru07]) but not those ones which are based on more powerful equivalences. Now we show how we can express by simulation opacity a stronger security property. We define an absence-of-information-flow property - Bisimulation Strong Nondeterministic Non-Interference (BSNNI, for short, see [FGM00]). Suppose that all actions are divided in two groups, namely public (low level) actions L and private. Process P has BSNNI property (we will write $P \in BSNNI$) if $P \setminus H$ behaves like P for which all high level actions are hidden for an observer. To express this hiding we introduce hiding operator $P/M, M \subseteq A$, for which it holds if $P \xrightarrow{a} P'$ then $P/M \xrightarrow{a} P'/M$ whenever $a \notin M \cup \bar{M}$ and $P/M \xrightarrow{\tau} P'/M$ whenever $a \in M \cup \bar{M}$. Formal definition of BSNNI follows.

Definition 8. *Let $P \in TPA$. We say that P has BSNNI property, and we write $P \in BSNNI$ iff $P \setminus H \approx P/H$.*

Example 1. Let $\phi(s)$ holds iff s contains actions from H and let $\theta = \{o_x | x \in L$, $\mathcal{O}(s) = o$ such that $o = o_{l_1} \dots o_{l_n}$ where $s|_L = l_1.l_2 \dots l_n$.

Then the following process

$$O_\phi = \mu X.(\sum_{x\in L} x.o_{\bar{x}}.X + \sum_{x\in H} x.\mu Y.(\sum_{x\in L} x.o_{\bar{x}}.Y + \sum_{x\in H} x.Y))$$

is the process definition of predicate ϕ and observation function $\mathcal{O}$.

Moreover process

$$O_{\neg\phi} = \mu X.(\sum_{x\in L} x.o_x.X)$$

is the process definition of predicate $\neg\phi$ and observation function $\mathcal{O}$.

Proposition 3. *$P \in BSNNI$ iff $P \in SO(O_\phi, O_{\neg\phi})$ for $O_\phi, O_{\neg\phi}$ defined in the previous example.*

Proof. Sketch. Process O_ϕ outputs o_x for every low level action which can be performed by P and switches to "accepting" state after the first high level action occurs. Similarly for $O_{\neg\phi}$. In definition of BSNNI the weak bisimulation is exploited but clearly, everything which can be performed by $P \setminus H$ can be performed by P/H and hence no more then simulation is needed.

Now we can return to the strong opacity. First we define its simulation version.

Definition 9 (Strong Simulation Opacity). *Given a set of processes $\Re$, predicate ϕ over $Actt^\star$ is strongly simulation opaque for $\Re$ w.r.t. the observation function $\mathcal{O}$ if for every $P \in \Re$ if $P \xrightarrow{s} P'$ for such s that $\phi(s)$ holds and $o(s) \neq \epsilon$ then there exists s' such that $\neg\phi(s')$ holds , $\mathcal{O}(s) = \mathcal{O}(s')$ and $P \xrightarrow{s'}$, and $P' \in \Re$ and, moreover, $P \xrightarrow{s} P''$ for such s that $\neg\phi(s)$ holds and $o(s) \neq \epsilon$ then there exists s' such that $\phi(s')$ holds , $\mathcal{O}(s) = \mathcal{O}(s')$ and $P \xrightarrow{s'}$, and $P'' \in \Re$. Predicate ϕ is strongly simulation opaque for process P with respect to $\mathcal{O}$ (denoted $P \in sSOp^\phi_\mathcal{O}$) if $P \in \Re$ for some strongly simulation opaque $\Re$ with respect to ϕ and $\mathcal{O}$.*

Similarly to the opacity, its stronger version is really different as it is stated by the following proposition.

Proposition 4. *$sSOp^\phi_\mathcal{O} \subseteq SOp^\phi_\mathcal{O}$ for every ϕ and $\mathcal{O}$. Moreover, there exist ϕ and $\mathcal{O}$ such that $sSOp^\phi_\mathcal{O} \subset SOp^\phi_\mathcal{O}$.*

Proof. The proof is just a variation of the proof of Proposition 1.

Definition 10. *We say that process P is strongly simulation opaque with respect to O_ϕ and $O_{\neg\phi}$ (denoted $P \in sSO(O_\phi, O_{\neg\phi})$ iff $(P|O_\phi) \setminus A \approx (P|O_{\neg\phi}) \setminus A$.*

Proposition 5. *Let O_ϕ and $O_{\neg\phi}$ are process definitions of observation function $\mathcal{O}$ and predicates ϕ and $\neg\phi$, respectively. Then $sSOp^\phi_\mathcal{O} = sSO(O_\phi, O_{\neg\phi})$.*

Proof. Again, the proof is similar as the proof of Proposition 2.

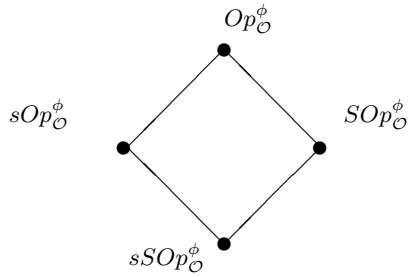


Fig. 2.

To complete a relationship between proposed opacity concepts we have the following proposition.

Proposition 6. *The relation between proposed opacities is depicted on Fig. 2.*

Proof. First we prove that $sSOp^{\phi}_{\mathcal{O}} \subset sOp^{\phi}_{\mathcal{O}}$. Let $P \in sSOp^{\phi}_{\mathcal{O}}$. Since for every sequence for which ϕ holds there exists observationally equal one, for which it does not hold and vice versa, we have $P \in sOp^{\phi}_{\mathcal{O}}$. Now let us consider process $P = l.l'.Nil + l.(h.l'.Nil + l'.Nil)$, an observation function which does not see action h and a predicate which holds for sequences containing h action. It easy to check that $P \in SOp^{\phi}_{\mathcal{O}}$ but $P \notin sOp^{\phi}_{\mathcal{O}}$. For process $P' = l.l'.Nil + l.h.l'.Nil$ we have $P' \notin SOp^{\phi}_{\mathcal{O}}$ but $P' \in sOp^{\phi}_{\mathcal{O}}$. The rest of the proof follows from Lemma 1, Propositions 1 and 4.

As it was mentioned, the opacity properties could be undecidable even for very simple observation functions or predicates (depending on their mutual combination). Here we can obtain its decidability by restrictions put on O_{ϕ} and $O_{\neg\phi}$, respectively. Note that existence of $O_{\neg\phi}$ for given O_{ϕ} is not guaranteed in general due to Turing power of TPA. As regards observation function, m-orwellian ones are the most interesting, since for their computations we do not need infinite memory and still the most of real attacks are based on them. As regards predicates, again those ones, which can be associated with finite automata are the most useful and frequent ones. If a combination of an observation function and predicates results in finite state process algebra the resulting properties are decidable. We elaborate this more precisely now.

We say that process E emulates an observational function $\mathcal{O}$ if it produces the corresponding output after receiving input traces. Formally, for every $w \in Actt^*$ it holds $\mathcal{O}(w) = o$ iff $(E|w.Nil) \setminus A \approx o$.

Lemma 2. *For every m-orwellian observation function there exists finite state process which emulates it.*

Proof. Sketch. Emulating process has to record the previous m inputs from emulated trace to produce an output. Emulation is straightforward. If $|A| = n$ then process which emulates given m-orwellian function has $O(m^n)$ states.

We call predicate ϕ finitely definable, if there exist finite state process T such that for every $w \in Actt^*$ $\phi(w)$ holds iff $(T|w.Nil) \setminus A \approx \surd.Nil$ where $\surd$ is a new symbol indicating the successful termination.

Proposition 7. *Let ϕ and $\neg\phi$ are finitely definable. Then opacity properties $SOp^{\phi}_{\mathcal{O}}$ and $sSOp^{\phi}_{\mathcal{O}}$ could be decided in time $O((n.m.k.|A|)^6)$ and $O((n.m.k.|A|)^3)$ for finite state processes and every m-orwellian observation function $\mathcal{O}$, where n, m, k are numbers of states of P, process emulating $\mathcal{O}$ and maximum of number of states of processes corresponding to $\phi, \neg\phi$, respectively.*

Proof. Sketch. We combine processes $\phi, \neg\phi$ and $\mathcal{O}$. First we need a special process which duplicates all action and one copy is send to process corresponding to the predicate and to proces for observation function. The size of this auxiliary process is $O(|A|)$. Hence the overall size of the process is $n.m.k.|A|$. The rest of the proof follows from complexity results for weak simulation and weak bisimulation (see [CPS90,KS83]).

If we have a process which does not belong to $SOp^{\phi}_{\mathcal{O}}$ for some ϕ and $\mathcal{O}$ then this means that the process could be jeopardize by an attacker which can react to process by means of $\mathcal{O}$ and is interested in validity of ϕ. But there are attacks which are not covered by our framework. For example, timing attacks, which have a particular position among attacks against systems security. They represent a powerful tool for "breaking" "unbreakable" systems, algorithms, protocols, etc. For example, by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems (see [Ko96]). This idea was developed in [DKL98] where a timing attack against smart card implementation of RSA was conducted.

We can extend our framework so that we can model also timing attacks and we can distinguish them from ordinary attacks. Here we formulate the property for simulation opacity but the same can be done also for strong simulation opacity.

Definition 11. *We say that process P is jeopardized by timing attack on validity of ϕ with a given observation function iff $(P|O_{\phi}) \setminus A \not\prec (P|O_{\neg\phi}) \setminus A$ and $(P|O_{\phi}) \setminus A \prec_{\{t\}} (P|O_{\neg\phi}) \setminus A$.*

Example 2. Let $\phi^{n,m}(s)$ for $1 < n < m$ holds iff $s = s_1.h.s_2.h'.s_3, h, h' \in H$ such that $n \leq |s_2|_{\{t\}}| \leq m$ and $s_1, s_2, s_3 \in (L \cup \{t\})^*$, i.e. $\phi^{n,m}(s)$ holds if s contains two private actions from H and time elapsing between their occurrences is between n and m time units and observation function see just low level actions and elapsing of time. Then the following process

$$O_{\phi}\mu X.(\sum_{x \in L} x.o_x.X + \sum_{x \in H} x.F')$$

where

$$F' = \mu X.(\sum_{x \in L} x.o_x.X + t.F_1),$$

$$F_i = \mu X.(\sum_{x \in L} x.o_x.X + t.F_{i+1})$$

for $i < n$ and

$$F_i = \mu X.(\sum_{x \in L} x.o_x.X + t.F'_{i+1})$$

for $i = n$,

$$F'_i = \mu X.(\sum_{x \in L} x.o_x.X + \sum_{x \in H} x.x^g.F'' + t.F'_{i+1})$$

for $i < m$ and

$$F'' = \mu X.(\sum_{x \in L} x.o_x.X + \sum_{x \in H} x.o.X + \sum_{x \in L} x.o_x.O_\phi + \sum_{x \in H} x.o.O_\phi)$$

is the process definition of predicate $\phi^{n,m}$. Similarly, for predicate $\neg\phi^{n,m}$ we can construct an appropriate finite state process. Clearly, timed proces are jeopardize by timing attacks on validity of $\phi^{n,m}$.

5 Conclusions

We have presented generalization of opacity called simulation opacity and we have elaborated it in timed process algebra setting. This concept offers not only an uniform framework for security theory but can be used to model more elaborated security properties than traditional ones and moreover, by careful choice of processes expressing predicated and observations we can obtain properties which can be effectively checked (note that in general, opacity is undecidable). By this concept we can also naturally model security with respect to limited time length of an attack, with a limited number of attempts to perform an attack and so on.

The presented approach allows us to use also other types of process algebras enriched by operators expressing also other properties (space, distribution, networking architecture, processor or power consumption and so on) and in this way also other types of attacks which exploit these information to detect information flow through various covert channels can be described.

Our approach limits us to predicates and observation functions (i.e. observers) which can be expressed by process algebra processes. In fact, this restriction does not represent any real limitation. Practically, all predicates and observation function of interest (used in known attacks) can be described by finite state processes and there is even no need to exploit full universal power of process algebras. In other words, it has no practical meaning to consider predicates and observation functions which cannot be effectively computed.

References

[BKR04] Bryans J., M. Koutny and P. Ryan: Modelling non-deducibility using Petri Nets. Proc. of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models, 2004.

[BKMR06] Bryans J., M. Koutny, L. Mazare and P. Ryan: Opacity Generalised to Transition Systems. In Proceedings of the Formal Aspects in Security and Trust, LNCS 3866, Springer, Berlin, 2006.

[BG04] Busi N. and R. Gorrieri: Positive Non-interference in Elementary and Trace Nets. Proc. of Application and Theory of Petri Nets 2004, LNCS 3099, Springer, Berlin, 2004.

[CPS90] Cleaveland R, J. Parrow and B. Steffen: A semantics-based verification tool for finite-state systems. Proc of Protocol specification, testong and verification, Elsevier Science Publishers, 1990.

[DKL98] Dhem J.-F., F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater and J.-L. Willems: A practical implementation of the timing attack. Proc. of the Third Working Conference on Smart Card Research and Advanced Applications (CARDIS 1998), LNCS 1820, Springer, Berlin, 1998.

[FGM00] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[GM04] Gorrieri R. and F. Martinelli: A simple framework for real-time cryptographic protocol analysis with compositional proof rules. Science of Computer Programing, Volume 50, Issues 13, 2004.

[GM82] Goguen J.A. and J. Meseguer: Security Policies and Security Models. Proc. of IEEE Symposium on Security and Privacy, 1982.

[Gro90] Groote, J. F.: "Transition Systems Specification with Negative Premises". Baeten, J.C.M. and Klop, J.W. (eds.), *CONCUR'90*, Springer Verlag, Berlin, LNCS 458, 1990.

[Gru12] Gruska D.P.: Informational analysis of security and integrity. Fundamenta Informaticae, vol. 120, Numbers 3-4, 2012.

[Gru12a] Test based security. Concurrency, Specification and Verification CS&P 2012, Vol. 1, Berlin, 2012.

[Gru11] Gruska D.P.: Gained and Excluded Private Actions by Process Observations. Fundamenta Informaticae, Vol. 109, Number 3, 2011.

[Gru10] Gruska D.P.: Process algebra contexts and security properties. Fundamenta Informaticae, vol. 102, Number 1, 2010.

[Gru09] Gruska D.P.: Quantifying Security for Timed Process Algebras. Fundamenta Informaticae, vol. 93, Numbers 1-3, 2009.

[Gru08] Gruska D.P.: Probabilistic Information Flow Security. Fundamenta Informaticae, vol. 85, Numbers 1-4, 2008.

[Gru07] Gruska D.P.: Observation Based System Security. Fundamenta Informaticae, vol. 79, Numbers 3-4, 2007.

[KS83] Kanellakis, P. C. and S.A. Smolka: CCS expressions, finite state processes, and three problems of equivalence. Proc. of the second annual ACM symposium on Principles of distributed computing, ACM, 1983.

[Ko96] Kocher P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. Proc. Advances in Cryptology - CRYPTO'96, LNCS 1109, Springer, Berlin, 1996.

[Mil89] Milner, R.: *Communication and concurrency.* Prentice-Hall International, New York,1989.

Structural and Dynamic Restrictions of Elementary Object Systems

Frank Heitmann and Michael Köhler-Bußmeier

University of Hamburg, Department for Informatics
Vogt-Kölln-Straße 30, D-22527 Hamburg
{heitmann,koehler}@informatik.uni-hamburg.de

Abstract. Elementary object systems (EOS for short) are Petri nets in which tokens may be Petri nets again. Originally proposed by Valk for a two levelled structure, the formalism was later generalised for arbitrary nesting structures.
However, even if restricted to a nesting depth of two, EOS are Turing-complete and thus many problems like reachability and liveness are undecidable for them. Nonetheless, since they are useful to model many practical applications a natural question is how to restrict the formalism in such a way, that the resulting restricted formalism is still helpful in a modelling context, but so that important verification problems like reachability become quickly decidable.
In the last years several structural and dynamic restrictions for EOS have therefore been investigated. These investigations have been central to the first author's recent PhD thesis and have been published in past CS&P conferences. In this paper we add several new results and present them together with the old in a unified fashion highlighting the central message of these investigations.

1 Introduction

Object Petri Nets are Petri Nets whose tokens may be Petri Nets again and thus may have an inner structure and activity. This approach is useful to model mobile systems and other systems arising in Computer Science which enjoy a certain nesting of structures (cf. [13] and [14]).

This approach, which is also called the *nets-within-nets* paradigm, was proposed by Valk [27, 28] for a two levelled structure and generalised in [15, 16] for arbitrary nesting structures. By now many related approaches like recursive nets [6], nested nets [24], adaptive workflow nets [25], AHO systems [11], PN^2 [10], Mobile Systems [23], and many others are known. Another line of research also dealing with nesting, but not in the field of Petri nets, is concerned with process calculi. Arguably most prominently there are the Ambient Calculus of Gordon and Cardelli [1] and the Seal Calculus [2] among many others. See [18] and [7] for a detailed discussion.

Unfortunately even if restricted to a two level structure as in elementary object systems, the formalism is Turing-complete. A natural question is how to restrict the formalism in such a way, that the resulting restricted formalism is still helpful in a modelling context, but so that important verification problems like reachability become quickly decidable. This "borderline" between modelling power and fast algorithms is in the case of p/t nets usually drawn at free choice Petri nets.

In the following we survey several structural restrictions for Eos and give results concerning the complexity of the reachability problem. Since we conclude, that even in very restricted cases the reachability problem becomes practically hard to decide, we then survey dynamic restrictions of Eos, most notably a safeness notion. Here, too, we focus on the complexity on the reachability problem.

The following section gives fundamental definitions of Eos. In Section 3 we survey results on structural restrictions and in Section 4 we survey results on dynamic restrictions. The paper ends with a summary of these results and a conclusion.

In the following we assume basic knowledge of Petri nets, see e.g. [26] and of Eos, see e.g. [18]. We do not define all notions rigorously here due to space restrictions, but all notions and an in-depth study can be found in [7].

2 Fundamentals

An elementary object system (Eos) is composed of a system net and a set of object nets where each of these nets is a p/t net. While the object nets use the usual black tokens, the system net's places are marked with either black tokens or object nets. For this each place of the system net is typed with an object net with the meaning the only object nets of these type may rest on that place. Additionally each transition may be labelled with a channel. The meaning here is that transitions with the same label may only fire synchronously.

Definition 1 (EOS). *An elementary object system (*Eos*) is a tuple* $OS = (\widehat{N}, \mathcal{N}, d, l)$ *such that:*

1. $\widehat{N}$ *is a p/t net, called the* system net.
2. $\mathcal{N}$ *is a finite set of disjoint p/t nets, called* object nets.
3. $d : \widehat{P} \to \mathcal{N}$ *is the typing of the system net places.*
4. $l = (\widehat{l}, (l_N)_{N \in \mathcal{N}})$ *is the labelling.*

An Eos *with initial marking is a tuple* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *where* $\mu_0 \in \mathcal{M}$ *is the initial marking.*

A system net transition may be labelled with a channel for each object net. Where an object net transition is only labelled with one channel. If for example a system net transition $\widehat{t}$ is labelled with channel c_1 for the object net N_1 and with channel c_2 for object net N_2, then $\widehat{t}$ may only fire, if it is possible to *synchronously* fire a transition in N_1 which is labelled with channel c_1 and if it is possible to

synchronously fire a transition in N_2 which is labelled with channel c_2. Firing may also happen system-autonomously (an unlabelled system net transition fires independently from any object net transition) or object-autonomously (firing of an unlabelled object net transition)

A formal treatment can be found in e.g. [9]. We only give an example here to illustrate the main points of the firing rule.

Example 1. Figure 1 shows an Eos with the system net $\widehat{N}$ and the object nets $\mathcal{N} = \{N_1, N_2\}$. The system has four net-tokens: two on place p_1 and one on p_2 and p_3 each. The net-tokens on p_1 and p_2 share the same net structure, but have independent markings.

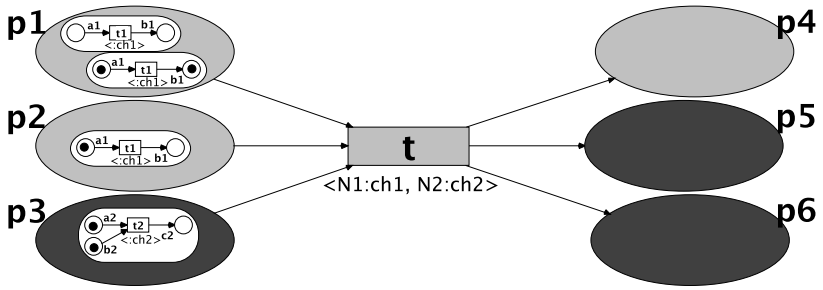


Fig. 1. An Elementary Object Net System

Formally we have the system net $\widehat{N} = (\widehat{P}, \widehat{T}, \mathbf{pre}, \mathbf{post})$ with the places and transitions given by $\widehat{P} = \{p_1, \ldots, p_6\}$ and $\widehat{T} = \{t\}$, the object net $N_1 = (P_1, T_1, \mathbf{pre}_1, \mathbf{post}_1)$ with $P_1 = \{a_1, b_1\}$ and $T_1 = \{t_1\}$ and the the object net $N_2 = (P_2, T_2, \mathbf{pre}_2, \mathbf{post}_2)$ with $P_2 = \{a_2, b_2, c_2\}$ and $T_2 = \{t_2\}$. The typing is given by $d(p_1) = d(p_2) = d(p_4) = N_1$ and $d(p_3) = d(p_5) = d(p_6) = N_2$.

We have two channels ch_1 and ch_2. The labelling function $\widehat{l}$ of the system net is defined by $\widehat{l}(t)(N_1) = ch_1$ and $\widehat{l}(t)(N_2) = ch_2$. The labelling l_{N_1} of the first object net is defined by setting $l_{N_1}(t_1) = ch_1$. Similarly, l_{N_2} is defined by $l_{N_2}(t_2) = ch_2$.

There is only one (synchronous) event: $\Theta = \Theta_l = \{t[N_1 \mapsto t_1, N_2 \mapsto t_2]\}$. The event is also written shortly as $t[t_1, t_2]$.

The initial marking μ has two net-tokens on p_1, one on p_2, and one on p_3:

$$\mu = p_1[a_1 + b_1] + p_1[\mathbf{0}] + p_2[a_1] + p_3[a_2 + b_2]$$

Note that for Figure 1 the structure is the same for the three net-tokens on p_1 and p_2 but the net-tokens' markings are different.

The marking μ enables $t[N_1 \mapsto t_1, N_2 \mapsto t_2]$ in the mode (λ, ρ), where

$$\begin{aligned}
\mu &= p_1[\mathbf{0}] + p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2] = p_1[\mathbf{0}] + \lambda \\
\lambda &= p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2] \\
\rho &= p_4[a_1 + b_1 + b_1] + p_5[\mathbf{0}] + p_6[c_2]
\end{aligned}$$

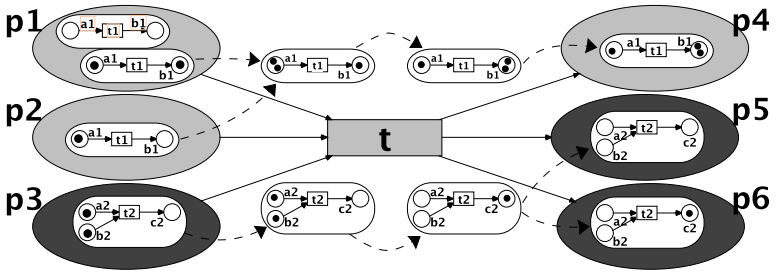


Fig. 2. The EOS of Figure 1 illustrating the projections $\Pi^2_N(\lambda)$ and $\Pi^2_N(\rho)$

The net-tokens' markings are added by the projections Π^2_N resulting in the markings $\Pi^2_N(\lambda)$. The sub-synchronisation generate $\Pi^2_N(\rho)$. (The results are shown above and below the transition t in Figure 2.) After the synchronisation we obtain the successor marking μ' with net-tokens on p_4, p_5, and p_6 as shown in Figure 2:

$$\begin{aligned}\mu' &= (\mu - \lambda) + \rho = p_1[\mathbf{0}] + \rho \\ &= p_1[\mathbf{0}] + p_4[a_1 + b_1 + b_1] + p_5[\mathbf{0}] + p_6[c_2]\end{aligned}$$

For general Eos the following theorem holds due to Köhler-Bußmeier [12].

Theorem 1 (Köhler 2007). Eos *can simulate* 2*-counter machines. Important problems like reachability and liveness are thus undecidable.*

In the following two sections we focus on introducing structural and dynamic restrictions that result in a decidable reachability problem.

3 Structural Restrictions

The main reason why Eos are Turing-complete is a null-test that is possible due to the firing rule. In Figure 3 the system net transition $\widehat{t}_1$ is not able to fire: the object net in the preset has a token on place a_1. This token would have to be distributed among the places of the same object net type in the postset of $\widehat{t}_1$, but there is no such place. The transition $\widehat{t}_2$ to the right may fire. The object net's marking is $\mathbf{0}$ and so there are no tokens that need to be distributed.

Conservative Eos avoid this by demanding that each object net type that appears in the preset of a system net transition also appears in the postset of that transition.

Definition 2 (Conservative Eos). *A typing is called* conservative *iff*

$$(d({}^\bullet\widehat{t}) \cup \{\bullet\}) \subseteq (d(\widehat{t}^\bullet) \cup \{\bullet\}),$$

i.e. each object net type that appears in the preset of $\widehat{t}$ *also appears in its postset.*

An Eos *is* conservative *iff its typing* d *is.*

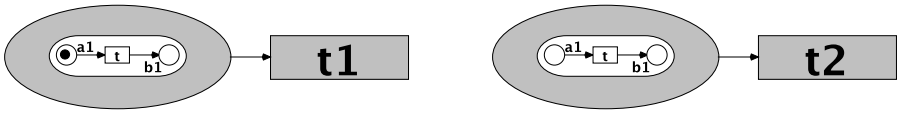


Fig. 3. The transition $\widehat{t}_1$ is disabled, $\widehat{t}_2$ is enabled.

While boundedness and coverability become decidable for conservative Eos, reachability and liveness remain undecidable. This was proven in [12] and [21].

Theorem 2. *For conservative* Eos *boundedness and coverability are decidable, while reachability and liveness are undecidable.*

The idea in the definition of conservative Eos can be strengthen further by demanding that each object net type appears exactly once in the preset and the postset of a system net transition or does not appear at all. This leads to *generalised state machines* (GSM) which are suitable to model many practical applications, since each object net can be seen as a physical entity.

Definition 3 (Generalised State Machines). *Let* $G = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *be an* Eos*.* G *is a* generalised state machine *(GSM) iff for all* $N \in \mathcal{N} \setminus \{N_\bullet\}$

1. $\forall \widehat{t} \in \widehat{T} : |\{\widehat{p} \in {}^\bullet\widehat{t} \mid d(\widehat{p}) = N\}| = |\{\widehat{p} \in \widehat{t}^\bullet \mid d(\widehat{p}) = N\}| \leq 1$
2. $\sum_{\widehat{p} \in \widehat{P}, d(\widehat{p}) = N} \Pi^1(\mu_0)(\widehat{p}) \leq 1$

holds.

In [17]) it was shown that for each GSM a p/t net, called reference net, can be easily constructed from which decidability results follow (see [12] and [17]).

Theorem 3. *The reachability problem is decidable for generalised state machines.*

While this is a first positive result, the set of transitions is defined by the set of events of the GSM and the size of this set can become exponential in the size of the GSM.

Lemma 1 ([8], [9]). *Let* $|T| := \max\{|T_N| \mid N \in \mathcal{N}\}$ *then* $|\Theta| \leq |\widehat{T}| \cdot |T|^{|\mathcal{N}|}$.

Given a GSM it might thus be very expensive to construct its reference net. This exponential blow up stems from the fact that in a GSM a $m : n$-synchronisation between the system net and the object nets exists, i.e. if there are m system net transitions labelled with channel c for object net N_1 and if in N_1 n transitions are labelled with channel c then each of the m system net transitions may fire synchronously with each of the n object net transitions, resulting in $m \cdot n$ different events.

To prevent this, we introduced deterministic GSMs and Eos in [8] (see also [9]).

Definition 4 (Deterministic and Strongly Deterministic Eos). *A* Eos *OS is called* deterministic *if for each $N \in \mathcal{N}$ and every two transitions $t, t' \in T_N$, $t \neq t'$ with $l_N(t) \neq \tau \neq l_N(t')$, $l_N(t) \neq l_N(t')$ holds, i.e. if the labels for all all $t \in T_N$ with $l_N(t) \neq \tau$ are pairwise different.*

OS is strongly deterministic *if OS is deterministic and additionally for all $\widehat{t}$ and N with $\widehat{l}(\widehat{t})(N) \neq \tau$ the labels $\widehat{l}(\widehat{t})(N)$ are pairwise different.*

Thus, in a deterministic Eos or GSM each channel is used at most once in each object net (resulting in a m : 1-synchronisation). In a strongly deterministic Eos or GSM each channel is additionally used at most once in the system net (resulting in a 1 : 1-synchronisation).

For Eos the definition of determinism does not significantly reduce the power of the formalisms introduced so far, namely of Eos or conservative Eos.

Theorem 4 ([7]). *The reachability problem for strongly deterministic, deterministic and general* Eos *is undecidable - even if the* Eos*is conservative.*

However, for GSMs the size of the events and thus the size of the reference net is reduced considerably.

Lemma 2 ([7]). *Let $G = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ be a deterministic or strongly deterministic GSM, then $|\Theta|$ is bounded above by $|\widehat{T}| + \sum_{N \in \mathcal{N}} |T_N|$.*

However, strongly deterministic GSMs can still simulate p/t nets and thus the reachability problem remains ExpSpace-hard.

Theorem 5 ([7]). *Every p/t net system $\mathcal{N}$ can be simulated by a strongly deterministic GSM $G_{\mathcal{N}}$.*

Corollary 1 ([7]). *The reachability problem for strongly deterministic, deterministic and general GSMs is* ExpSpace*-hard.*

Due to this results further structural restrictions are necessary to reduce the complexity of the reachability problem. Several structural restrictions known from p/t nets are carried over to GSMs and investigated in [8], [9], [21], and [7].

The results are summarised in Table 1. ttGSMs, ppGSMs, ptGSMs, and tpGSMs are GSMs where the system net and/or the object nets are restricted to be T-nets or P-nets. In ptGSMs the system net is a P-net and the object nets are T-nets and in tpGSMs the system net is a T-net and the object nets are P-nets. Despite being rather simple in the case of p/t nets, it is evident in Table 1, that the reachability problem becomes hard in the case of object nets due to the synchronisation between the system net and the object nets. Most notably in the case of ppGSMs where all participating nets are P-nets and thus similar to finite automata, the reachability problem is already PSpace-complete.

acGSMs are GSMs where all nets are acylic and fcGSMs are GSMs where all nets are free-choice nets. For cfGSMs the definition of conflict-freedom has to be adapted, because it is not enough to demand a similar structural restrictions as for p/t nets if one wants to structurally rule out conflicts (cf. [8] and [7]).

Table 1. Complexity of the reachability problem for various formalisms.

	strongly deterministic	deterministic	general
ttGSM	P	?	?
ppGSM	PSpace-complete	PSpace-complete	PSpace-complete
ptGSM	NP-hard	PSpace-hard	PSpace-hard
tpGSM	?	?	?
acGSM	NP-hard	NP-hard	NP-hard
cfGSM	NP-complete	NP-hard	NP-hard
fcGSM	ExpSpace-hard	ExpSpace-hard	ExpSpace-hard
GSM	ExpSpace-hard	ExpSpace-hard	ExpSpace-hard
cEos	undecidable	undecidable	undecidable
Eos	undecidable	undecidable	undecidable

Definition 5 (Conflict-Free GSMs ([8])). *A GSM* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *is* conflict-free *if*

1. $\forall N \in \mathcal{N} \cup \{\widehat{N}\}\ \forall p \in \widehat{P} \cup P_{\mathcal{N}} : |p^\bullet| > 1 \Rightarrow p^\bullet \subseteq {}^\bullet p$
2. $\forall N \in \mathcal{N}\ \forall p \in P_N : (\exists t \in p^\bullet\ \exists \widehat{t}_1, \widehat{t}_2 \in \widehat{T}\ \exists \widehat{p} \in \widehat{P}\ \exists c \in C : \widehat{t}_1 \neq \widehat{t}_2 \wedge \widehat{p} \in {}^\bullet\widehat{t}_1 \cap {}^\bullet\widehat{t}_2 \wedge d(\widehat{p}) = N \wedge l_N(t) = \widehat{l}(\widehat{t}_1)(N) = \widehat{l}(\widehat{t}_2)(N) = c) \Rightarrow p \in t^\bullet$

holds. We also say that OS is a cfGSM.

Proofs for the results in Table 1 can be found in [7] with pointers to the literature where the results where first proven.

Most notably in Table 1 are the results for ppGSMs and for fcGSMs. They show that even very strong structural restrictions as in the case of ppGSMs lead to an already hard to solve reachability problem and that more openly structural restrictions as in the case of fcGSMs where the formalism would be suitable for modelling purposes lead to complexity results that render algorithms practically unusable. Thus other restrictions than structural restrictions are necessary if one aims at solving the reachability problem quickly.

4 Dynamics Restrictions

In [7] unary and persistent Eos are introduced where in unary Eos in each reachable marking at most one event is enabled and in persistent Eos conflicts are dynamically ruled out. For both formalisms, however, the reachability problem remains undecidable. These dynamical restrictions are thus not strong enough for our purpose.

In general, the state space of an Eos is of infinite size which is a source for undecidability results or strong lower complexity bounds. In [19] we therefore introduced four different notions of safeness for Eos, safe(1), safe(2), safe(3), and safe(4), to adapt the notion of safeness for p/t nets to Eos. For p/t nets 1-safeness guarantees not only finiteness of the state space size but also that each reachable marking can be seen as a set. This set idea is adapted in [19]. Furthermore, a safe(4) Eos is also a safe(3) Eos which is in turn a safe2 Eos

and so on. Positive results concerning the solvability of the reachability problem thus carry over from safe(i) to safe($i+1$) and negative results carry over from safe($i+1$) to safe(i).

However, despite the fact that the markings are sets, safe(1) and safe(2) Eosstill have an infinite state space and the reachability problem remains undecidable for them even if the Eos is additionally structurally restricted.

Theorem 6 ([19], [7]). *The reachability problem is undecidable for safe(1) or safe(2)* Eos *- even in the case of strongly deterministic and conservative* Eos.

In the following we concentrate on safe(3) Eos which have a finite state space.

Definition 6 (Safeness). *An* Eos *OS is* safe(3) *or simply* safe *iff for all reachable markings there is at most one token on each system net place and each net-token is safe:*

$$\forall \mu \in RS(OS) : \forall \widehat{p} \in \widehat{P} : \Pi^1(\mu)(\widehat{p}) \leq 1 \wedge \\ \forall N \in \mathcal{N} : \forall p \in P_N : \forall \widehat{p}[M] \leq \mu : M(p) \leq 1$$

Theorem 7 ([19], [7]). *If an* Eos *is safe(3) or safe(4), then its set of reachable markings is finite.*

Indeed, an upper bound for the state space size if given in [7]. Let $k := |\widehat{P}|$ and $l := \max\{|P_N| \mid N \in \mathcal{N}\}$, then there are at most $(1+2^l)^k$ different markings.

For safe Eos two very strong result can be shown. Not only are reachability and liveness decidable, but every property that can be expressed in the temporal logics LTL or CTL can be decided in polynomial space in the size of the Eos and the formula. The the problems are PSpace-hard follows directly from similar results for safe p/t nets (see e.g. [5]). It is thus surprising that this bound can also be met from above in the case of safe Eos despite their quite huge state space.

Theorem 8 ([19], [7]). *Given a safe(3) or safe(4)* Eos *OS and an LTL formula ϕ, checking whether OS satisfies ϕ can be done in polynomial space in the size of OS and ϕ, that is, there is a polynomial p, independent of OS and ϕ, such that the algorithm uses $O(p(|OS| + |\phi|))$ space.*

Corollary 2. *The reachability problem for safe* Eos *is* PSpace*-complete.*

Theorem 9 ([20], [7]). *Given a safe(3) or safe(4)* Eos *OS and a CTL formula ϕ checking whether OS satisfies ϕ can be done in $O(|OS|^4 \cdot |\phi|)$ space.*

Corollary 3. *The liveness problem for safe* Eos *is* PSpace*-complete.*

Both model checking algorithms are an adaption of a technique from Esparza for 1-safe p/t nets [5]. The LTL model checking algorithms additionally uses techniques from Vardi from automata theory [29].

The proofs of the above theorems are very involved. Detailed discussions and proofs can be found in [19], [20], and [7].

In addition to the results above it can also be decided in polynomial space if an Eos is safe(3), which is helpful from a modelling point of view.

Theorem 10 ([7]). *Given an* Eos *OS it is* PSpace*-complete to decide if OS is safe.*

Safe Eos can in addition be structurally restricted. In some cases as in the case of conflict-free Eos this reduces the complexity of the reachability problem (cf. Table 2). However, the formalisms where this happens seem to be too restricted to be useful in a modelling context.

Table 2. Complexity of the reachability problem for *safe* Eos with further structural restrictions.

	strongly deterministic	deterministic	general
ttGSM	P	PSpace	PSpace
ppGSM	PSpace-complete	PSpace-complete	PSpace-complete
ptGSM	NP-hard, PSpace	PSpace-complete	PSpace-complete
tpGSM	PSpace	PSpace	PSpace
acGSM	PSpace	PSpace	PSpace
cfGSM	P	PSpace	PSpace
fcGSM	PSpace-complete	PSpace-complete	PSpace-complete
GSM	PSpace-complete	PSpace-complete	PSpace-complete
cEos	PSpace-complete	PSpace-complete	PSpace-complete
Eos	PSpace-complete	PSpace-complete	PSpace-complete

A discussion of structural restrictions of safe(1) and safe(2) Eos can be found in [7].

5 Conclusion and Outlook

In summary we have investigated formalisms which are useful to model mobility, interaction, and nesting of structures. We then focused on object nets, but it is possible to adapt these results for other formalisms mentioned in the introduction. Since in their general form object nets are Turing-complete it was our goal to restrict the formalism of elementary object systems such that modelling capabilities mostly remain and interesting applications can still be conveniently modelled, and also such that properties can be automatically and quickly verified.

To achieve this goal we have restricted the formalism structurally and dynamical. We introduced restrictions natural for object nets like determinism and strong determinism and carried over restrictions known from p/t nets to object nets. We then focused on the complexity of the reachability problem to evaluate the formalisms. The results are summarised in Table 1 and 2.

As a conclusion structural restrictions alone are not enough even if the possibility to synchronise is additionally restricted. The restriction to safe(3) Eos,however, allows for a quick verification of important properties and is also still useful from a modelling point of view.

In [7] EOS and GSMs are furthermore generalised to an arbitrary but fixed nesting depth $k > 2$. In this case a safeness definition can also be introduced which allows to carry over the results for CTL and LTL model checking. These problems are then solvable in polynomial space, too, albeit the polynomial worsens.

Also in [7] object net systems are introduced, which allow a vertical transport of net tokens, i.e. a transport of net tokens between nesting levels. For this systems different safeness definitions are introduced in [7]. In particular, it is not enough to demand that on each place resides at most one (net) token. Due to the arbitrary nesting depth, the state space might still be infinite. However, for the strongest safeness definition for object net systems in [7], which among other things does not allow the creation and destruction of net tokens, again PSPACE-completeness results for LTL- and CTL-model-checking can be established.

Open question in the context presented here are, obviously, to match upper and lower bounds in the tables above. While this might be interesting from a theoretical point of view, the effect for the question tackled here are limited. The open cases are for formalisms which are structurally restricted in such a way, that using them in a modelling context is doubtful. Thus even if the reachability problem is solvable more quickly than PSPACE the formalisms will not be practically useful.

Another open question concerns the borderline between safe(1) and safe(2) EOS and the borderline between safe(3) and safe(4) EOS. While it is known that e.g. for safe(3) and safe(4) EOS the reachability problem is PSPACE-complete, the polynomial will almost surely be smaller for safe(4) EOS. Again this question is more from a theoretical nature.

A question with much practical relevance is how to improve the LTL- and CTL-model-checking algorithms uses so far in this context. In [7] the algorithm are rather direct and not optimised. It is to be expected that these algorithms can be improved. Afterwards, it would be nice to implement these algorithms in a tool which can then be used by modellers. We believe that such a tool might be very useful in practice, since many applications can be modelled more intuitively with object nets and such a tool would allow to verify properties of the model automatically and with modest time and space requirements.

Questions not tackled so far which would open up whole new directions are *compositionality* and *adaptivity*. The goal in compositionality is to find properties and restrictions such that properties of the whole system can than be deduced from properties of components treated in isolation. This might reduce the complexity considerably. First results concerning compositionality with regard to nested nets are published in [3] and [4].

Adaptivity means to introduce formalisms which do not only allow a transport of net tokens as presented here, but also to manipulated these net tokens during run-time. First formalisms which allow this are adaptive workflow nets [25] and higher order nets [22].

This two questions, how to introduce compositionality and adaptivity for object nets, are the questions we want to tackle in the future. We believe that

they will be of high practical relevance for run-time analysis of systems in general and workflows in particular.

References

1. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
2. G. Castagna, J. Vitek, and F. Zappa Nardelli. The seal calculus. *Information and Computation*, 201:1–54, 2005.
3. Leonid W. Dworzański and Irina A. Lomazova. On compositionality of boundedness and liveness for nested Petri nets. In Marcin Szczuka, Ludwik Czaja, Andrzej Skowron, and Magdalena Kacprzak, editors, *Concurrency, Specification and Programming (CS&P 2011), Proceedings*, Pułtusk, Poland, 2011. Białystok University of Technology.
4. Leonid W. Dworzański and Irina A. Lomazova. On compositionality of boundedness and liveness for nested Petri nets. *Fundamenta Informaticae*, 120(3–4):275–293, 2012.
5. Javier Esparza. Decidability and complexity of petri net problems – an introduction. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.
6. Serge Haddad and Denis Poitrenaud. Theoretical aspects of recursive Petri nets. In S. Donatelli and J. Kleijn, editors, *Application and Theory of Petri Nets*, volume 1639 of *Lecture Notes in Computer Science*, pages 228–247. Springer-Verlag, 1999.
7. Frank Heitmann. *Algorithms and Hardness Results for Object Nets*. PhD thesis, University of Hamburg, 2013.
8. Frank Heitmann and Michael Köhler-Bußmeier. On defining conflict-freedom for object nets. In B. Farwer and M. Köhler-Bußmeier, editors, *Proceedings of the Second International Workshop on Logic, Agents, and Mobility (LAM 2011)*, 2011.
9. Frank Heitmann and Michael Köhler-Bußmeier. P- and t-systems in the nets-within-nets-formalism. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets. 33rd International Conference, PETRI NETS 2012. Hamburg, Germany, June 2012. Proceedings*, volume 7347 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, 2012.
10. Kunihiko Hiraishi. PN^2: An elementary model for design and analysis of multi-agent systems. In Farhad Arbab and Carolyn L. Talcott, editors, *Coordination Models and Languages, COORDINATION 2002*, volume 2315 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag, 2002.
11. Kathrin Hoffmann, Hartmut Ehrig, and Till Mossakowski. High-level nets with nets and rules as tokens. In *Application and Theory of Petri Nets and Other Models of Concurrency*, volume 3536 of *Lecture Notes in Computer Science*, pages 268 – 288. Springer-Verlag, 2005.
12. Michael Köhler. The reachability problem for object nets. *Fundamenta Informaticae*, 79(3-4):401 – 413, 2007.
13. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modeling the behaviour of Petri net agents. In J. M. Colom and M. Koutny, editors, *Application and Theory of Petri Nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.

14. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling mobility and mobile agents using nets within nets. In W. v. d. Aalst and E. Best, editors, *Application and Theory of Petri Nets*, volume 2679 of *Lecture Notes in Computer Science*, pages 121–140. Springer-Verlag, 2003.
15. Michael Köhler and Heiko Rölke. Concurrency for mobile object-net systems. *Fundamenta Informaticae*, 54(2-3), 2003.
16. Michael Köhler and Heiko Rölke. Properties of Object Petri Nets. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets*, volume 3099 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 2004.
17. Michael Köhler and Heiko Rölke. Reference and value semantics are equivalent for ordinary object petri nets. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets*, volume 3536 of *Lecture Notes in Computer Science*, pages 309–328. Springer-Verlag, 2005.
18. Michael Köhler-Bußmeier. Decidability results for elementary object systems. Report of the department of informatics, Universität Hamburg, Fachbereich Informatik, 2011.
19. Michael Köhler-Bußmeier and Frank Heitmann. Safeness for object nets. *Fundamenta Informaticae*, 101(1-2):29–43, 2010.
20. Michael Köhler-Bußmeier and Frank Heitmann. Liveness of safe object nets. *Fundamenta Informaticae*, 112(1):73–87, 2011.
21. Michael Köhler-Bußmeier and Frank Heitmann. Conservative elementary object systems. *Fundamenta Informaticae*, 120(3–4):325–339, 2012.
22. Michael Köhler-Bußmeier and Frank Heitmann. Complexity results for elementary hornets. In José-Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency. 34th International Conference, PETRI NETS 2013. Milan, Italy, June 24-28, 2013. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 150–169. Springer-Verlag, 2013.
23. Charles Lakos. A Petri net view of mobility. In *Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 174–188. Springer-Verlag, 2005.
24. Irina A. Lomazova. Nested Petri nets – a formalism for specification of multi-agent distributed systems. *Fundamenta Informaticae*, 43(1-4):195–214, 2000.
25. Irina A. Lomazova, Kees M. van Hee, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. Nested nets for adaptive systems. In *Petri Nets and Other Models of Concurrency - ICATPN 2006. 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, Turku, Finland, June 26-30, 2006. Proceedings*, volume 4024 of *Lecture Notes in Computer Science*, pages 241–260. Springer-Verlag, 2006.
26. Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
27. Rüdiger Valk. Modelling concurrency by task/flow EN systems. In *3rd Workshop on Concurrency and Compositionality*, number 191 in GMD-Studien, St. Augustin, Bonn, 1991. Gesellschaft für Mathematik und Datenverarbeitung.
28. Rüdiger Valk. Object Petri nets: Using the nets-within-nets paradigm. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advanced Course on Petri Nets 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, 2003.
29. Moshe Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

Causal Structures for General Concurrent Behaviours

Ryszard Janicki[1], Jetty Kleijn[2], Maciej Koutny[3], and Łukasz Mikulski[3,4]

[1] McMaster University, Canada
[2] LIACS, Leiden University, The Netherlands
[3] Newcastle University, U.K.
[4] Nicolaus Copernicus University, Poland

Abstract. Non-interleaving semantics of concurrent systems is often expressed using posets, where causally related events are ordered and concurrent events are unordered. Each causal poset describes a unique concurrent history, i.e., a set of executions, expressed as sequences or step sequences, that are consistent with it. Moreover, a poset captures all precedence-based invariant relationships between the events in the executions belonging to its concurrent history. However, concurrent histories in general may be too intricate to be described solely in terms of causal posets. In this paper, we introduce and investigate generalised mutex order structures which can capture the invariant causal relationships in any concurrent history consisting of step sequence executions. Each such structure comprises two relations, viz. interleaving/mutex and weak causality. As our main result we prove that each generalised mutex order structure is the intersection of the step sequence executions which are consistent with it.
Keywords: concurrent history, causal poset, weak causal order, mutex relation, interleaving, step sequence, causality semantics.

1 Introduction

In order to design and validate complex concurrent systems, it is essential to understand the fundamental relationships between events occurring in their executions. However, looking at sequential descriptions of executions in the form of sequences or step sequences is insufficient when it comes to providing faithful information about causality and independence between events. To address this drawback, one may resort to using partially ordered sets of events providing explicit representation of causality in the executions of a concurrent system. In particular, the order in which independent events are observed may be accidental and those executions which only differ in the order of occurrences of independent events may be regarded as belonging to the same *concurrent history*, underpinned by a causal poset [1, 13, 16, 17, 21].

In general, concurrent behaviours can be investigated at the level of individual executions as well as at the level of order structures, like causal posets, capturing the essential invariant dependencies between events. The key link between these

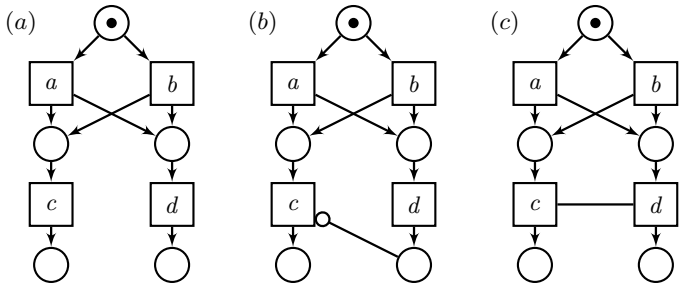


Fig. 1. A safe Petri net (a), extended with an inhibitor arc implying that when c is executed the output place of d must be empty (b), and extended with a mutex arc implying that c and d cannot be executed simultaneously (c).

two levels is the notion of a concurrent history [6], an *invariant closed* set Δ of executions. The latter means that Δ is fully determined by invariant relationships over X, its set of events: causality ($e \prec_\Delta f$ if, in all executions of Δ, e precedes f); weak causality ($e \sqsubset_\Delta f$ if, in all executions of Δ, e either precedes or is simultaneous with f); and interleaving/mutex ($e \rightleftharpoons_\Delta f$ if, in all executions of Δ, e is not simultaneous with f). In the case of safe Petri nets with sequential executions, $\prec_\Delta$ is the only invariant we need (as then, e.g., $\prec_\Delta = \sqsubset_\Delta$ and $\rightleftharpoons_\Delta = \prec_\Delta \cup \prec_\Delta^{-1}$). In particular, Δ is the set of all sequential executions corresponding to the linearisations of $\prec_\Delta$. The soundness of this approach is validated by Szpilrajn's Theorem [20] which states that each poset is equal to the intersection of its linearisations.

In this paper, executions are observed as step sequences, i.e., sequences of finite sets (steps) of simultaneously executed events. As an example, consider the safe Petri net depicted in Figure 1(a) which generates three step sequences involving a, c and d, viz. $\sigma = \{a\}\{c,d\}$, $\sigma' = \{a\}\{c\}\{d\}$ and $\sigma'' = \{a\}\{d\}\{c\}$. They can be seen as forming a single concurrent history $\Delta = \{\sigma, \sigma', \sigma''\}$ underpinned by a causal poset $\prec_\Delta$ satisfying $a \prec_\Delta c$ and $a \prec_\Delta d$. Moreover, such a Δ adheres to the following *true concurrency paradigm*:

> Given two events (c and d in Δ), they can be observed as simultaneous (in σ) $\Longleftrightarrow$ they can be observed in both orders (c before d in σ', and d before c in σ''). (TrueCon)

Concurrent histories adhering to TrueCon are underpinned by *causal partial orders*, in the sense that each such history comprises *all* step sequence executions consistent with a unique causal poset on events involved in the history.

In [6] fundamental concurrency paradigms are identified, including (TrueCon). Another paradigm is characterised by (TrueCon) with $\Longleftrightarrow$ replaced by $\Longleftarrow$. This paradigm has a natural system model interpretation provided by safe Petri nets with inhibitor arcs. Figure 1(b) depicts such a net generating two step sequences involving a, c and d, viz. $\sigma = \{a\}\{c,d\}$ and $\sigma' = \{a\}\{c\}\{d\}$. They form

a concurrent history $\Delta' = \{\sigma, \sigma'\}$ adhering to the paradigm that unorderedness implies simultaneity, but *not* to the true concurrency paradigm as Δ' has no step sequence in which d precedes c although in σ, c and d occur in a single step.

As a result, histories adhering to the weaker paradigm are *not* underpinned by causal partial orders, but rather by causality structures $(X, \prec, \sqsubset)$ introduced in [7] — called *stratified order structures* (SO-structures) — based on causality and an additional weak causality ('not later than') relation. A version of Szpilrajn's Theorem can be shown to hold also for SO-structures and the concurrent histories they generate. Stratified order structures were independently introduced in [3] (as 'prossets'). Their comprehensive theory was developed in e.g. [8, 9, 12, 15]. As shown in this paper, SO-structures can be represented in a one-to-one manner by mutex order structures, or MO-structures, $(X, \rightleftharpoons, \sqsubset)$ based on interleaving/mutex and weak causality. The first, symmetric, relation defines the events that never occur simultaneously. Hence strict event precedence (causality) can be captured as a combination of mutex and weak causality.

This paper focuses on the least restrictive paradigm. i.e., there are no constraints imposed on concurrent histories. It admits all (invariant closed) concurrent histories comprising step sequence executions. As shown in [6], it is now sufficient to consider only two invariant relations, viz. mutex and weak causality. Figure 1(c) depicts a safe Petri net with mutex arcs (see [11]) generating two step sequences involving a, c and d, viz. $\sigma' = \{a\}\{c\}\{d\}$ and $\sigma'' = \{a\}\{d\}\{c\}$. We first observe that they form a concurrent history $\Delta'' = \{\sigma', \sigma''\}$ in which the executions of c and d interleave, and are both preceded by a; in other words, $c \rightleftharpoons_{\Delta''} d$, $a \sqsubset_{\Delta''} c$, $a \sqsubset_{\Delta''} d$ and $c \rightleftharpoons_{\Delta''} a \rightleftharpoons_{\Delta''} d$. That Δ'' is a concurrent history then follows from the observation that Δ'' contains *all* step sequences involving a, c and d which obey these invariant relationships. However, Δ'' does *not* conform to the two earlier considered paradigms as there is no step sequence in Δ'' in which c and d occur simultaneously. To summarise, a nonempty set Δ of step sequence executions over a common set of events X, is a concurrent history iff Δ consists of all step sequences σ over X such that for all $e, f \in X$: $e \rightleftharpoons_\Delta f$ implies that e and f are not simultaneous in σ, and $e \sqsubset_\Delta f$ implies that e precedes or is simultaneous with f in σ.

The aim of this paper is to provide a structural characterisation of general concurrent histories (consisting of step sequence executions). An early attempt to describe structures of this kind was made in [4]. The there proposed generalised stratified order structures (or GSO-structures) do however not always capture all implied invariant relationships involving the mutex relation. Here, we will show that *generalised mutex order structures* (or GMO-structures) describe exactly all general concurrent histories. The main result is a version of Szpilrajn's Theorem, formulated and proven to hold for GMO-structures and concurrent histories. For this we develop a notion of GMO-closure which is the GMO-structure counterpart of transitive closure of an acyclic relation.

First, we recall key notions and notations used throughout the paper. In Section 3, we introduce MO-structures and establish their relationship with stratified order structures. Then, Section 4 introduces GMO-structures and proves

their main properties, including GMO-closure and the GMO-structure version of Szpilrajn's Theorem. Section 5 presents concluding remarks.
Proofs of all the results can be found in the Technical Report available at http://www.cs.ncl.ac.uk/publications/trs/papers/1378.pdf.

2 Preliminary Definitions

Throughout the paper we use the standard notions of set theory and formal language theory. In particular, $\biguplus$ denotes disjoint set union. The identity relation on a set X is defined as $Id_X = \{\langle a, a\rangle \mid x \in X\}$, the index X may be omitted if it is clear from the context.

Composing relations. The composition of two binary relations, R and Q, over X is given by $R \circ Q = \{\langle a, b\rangle \mid \exists x \in X : \ aRx \wedge xQb\}$. Moreover, if $P \subseteq X \times X$, then we define $R \circ_P Q = \{\langle a, b\rangle \mid \exists \langle x, y\rangle \in P : \ aRxQb \wedge aRyQb\}$ (see Figure 2).

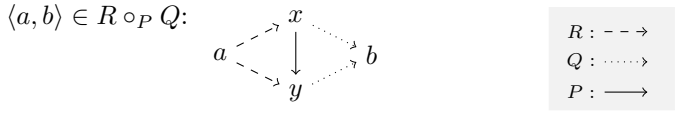


Fig. 2. A visualisation of $\circ_P$ composition.

Note that $\circ = \circ_{Id}$, and the associativity of relation composition holds for the extended notion. We will also denote $a_1 \ldots a_k R b_1 \ldots b_m$ whenever $a_i R b_j$, for all i, j. For example, $aRbcQd$ means that $aRbQd$ and $aRcQd$.

Given a relation $R \subseteq X \times X$, $R^0 = Id$ and $R^n = R^{n-1} \circ R$, for all $n \geq 1$. Then: (i) the reflexive closure of R is defined by $R \cup Id$; (ii) the transitive closure by $R^+ = \bigcup_{i \geq 1} R^i$; (iii) the reflexive transitive closure by $R^* = Id \cup R^+$; and (iv) the irreflexive transitive closure by $R^\lambda = R^+ \setminus Id = R^* \setminus Id$. Moreover, the inverse of R is given by $R^{-1} = \{\langle a, b\rangle \mid \langle b, a\rangle \in R\}$, and the symmetric closure by $R^{sym} = R \cup R^{-1}$.

Order relations. A relation $R \subseteq X \times X$ is: (i) symmetric if $R = R^{-1}$; (ii) antisymmetric if $R \cap R^{-1} \subseteq Id$; (iii) reflexive if $Id \subseteq R$; (iv) irreflexive if $Id \cap R = \varnothing$; (v) transitive if $R \circ R \subseteq R \cup Id$; and (vi) total if $R \cup R^{-1} = X \times X$.

A relation $R \subseteq X \times X$ is: (i) an *equivalence relation* if it is symmetric, transitive and reflexive; (ii) a *pre-order* if it is transitive and irreflexive; (iii) a *partial order* if it is an antisymmetric pre-order; and (iv) a *total order* if it is a partial order and $R \cup Id$ is total; (v) a *stratified order* if it is a partial order such that $X \times X \setminus R^{sym}$ is an equivalence relation.

Every irreflexive relation $R \subseteq X \times X$ induces a least pre-order containing R defined by R^λ. Following E. Schröder [19], we define the largest equivalence relation contained in R^* as $R^{\circledast} = R^* \cap (R^*)^{-1} = (R^\lambda \cap (R^\lambda)^{-1}) \uplus Id$.

For a stratified order $R \subseteq X \times X$ we define two relations, $\sqsubset_R$ and $\rightleftharpoons_R$, such that, for all distinct $a, b \in X$:

$$a \sqsubset_R b \iff \neg(bRa) \quad \text{and} \quad a \rightleftharpoons_R b \iff \neg(a \sqsubset_R^{\circledast} b) \iff aRb \vee bRa\ .$$

Intuitively, if R represents a stratified order execution, aRb means '*a occurred earlier than b*'. In such a case $a \sqsubset_R b$ means '*a occurred not later than b*', $a \rightleftharpoons_R b$ means '*a did not occur simultaneously with b*', and $a \sqsubset_R^{\circledast} b$ means '*a occurred simultaneously with b*'.

Relational structures. A tuple $S = (X, R_1, R_2, \ldots, R_n)$, where $n \geq 1$ and each $R_i \subseteq X \times X$ is a binary relation on X, is an (n-ary) *relational* structure. By the *domain* of a relational structure S we mean the set X. An *extension* of S is any relational structure $S' = (X, R'_1, R'_2 \ldots, R'_n)$ satisfying $R_i \subseteq R'_i$, for every $1 \leq i \leq n$. We denote this by $S \subseteq S'$. The *intersection* of a nonempty family $\mathcal{R} = \{(X, R_1^i, \ldots, R_n^i) \mid i \in I\}$ of relational structures with the same domain and arity is given by $\bigcap \mathcal{R} = (X, \bigcap_{i\in I} R_1^i, \ldots, \bigcap_{i\in I} R_n^i)$. In what follows, we consider only relational structures that contain two relations, while the set X is finite.

A relational structure $S = (X, Q, R)$ is: (i) *separable* if $Q \cap R^{\circledast} = \varnothing$, Q is symmetric and R is irreflexive; and (ii) *saturated* in a family of relational structures $\mathscr{X}$ if it belongs to $\mathscr{X}$ and for every extension $S' \in \mathscr{X}$ of S, we have $S = S'$. It is easily seen that an intersection of separable relational structures is also separable. Intuitively, if Q represents 'mutex' and R 'weak precedence', then separability means that simultaneous events cannot be in the mutex relation.

A *stratified order structure* (or SO-structure) is defined as a relational structure $sos = (X, \prec, \sqsubset)$, where $\prec$ and $\sqsubset$ are binary relations on X such that, for all $a, b, c \in X$:

$$\begin{array}{lll} S1: a \not\sqsubset a & & S3: \ a \sqsubset b \sqsubset c \ \wedge \ a \neq c \implies a \sqsubset c \\ S2: a \prec b \implies a \sqsubset b & & S4: \ a \sqsubset b \prec c \ \vee \ a \prec b \sqsubset c \implies a \prec c\ . \end{array}$$

A *generalized stratified order structure* [4] (or GSO-structure) is a relational structure $gsos = (X, \rightleftharpoons, \sqsubset)$ such that $\sqsubset$ is irreflexive, $\rightleftharpoons$ is irreflexive and symmetric, and $(X, \rightleftharpoons \cap \sqsubset, \sqsubset)$ is an SO-structure. A comprehensive treatment of GSO-structures can be found in [5].

Properties. For every binary relation $R \subseteq X \times X$ and all $a, b \in X$, we have:

$$(R \cup \langle a, b\rangle)^* = R^* \cup \{\langle c, d\rangle \mid cR^*a \wedge bR^*d\}\ . \tag{1}$$

$$\neg(bR^*a) \implies (R \cup \langle a, b\rangle)^{\circledast} = R^{\circledast} \tag{2}$$

$$R^{\circledast} = (R^{\circledast})^{-1} \subseteq R^* \tag{3}$$

$$(R^{\curlywedge})^{\curlywedge} = R^{\curlywedge} \qquad (R^{\curlywedge})^* = R^* \qquad (R^*)^* = R^* \qquad (R^{\curlywedge})^{\circledast} = R^{\circledast} \tag{4}$$

$$R^{\circledast} \circ R^{\circledast} = R^{\circledast} \qquad R^* \circ R^{\circledast} = R^{\circledast} \circ R^* = R^* \circ R^* = R^* \tag{5}$$

If $R \subseteq X \times X$ is a stratified order, then $\rightleftharpoons_R$ is irreflexive and symmetric, while $\sqsubset_R$ is a pre-order such that:

$$\sqsubset_R = \sqsubset_R^+ \setminus Id \ = \sqsubset_R^{\curlywedge} \quad \text{and} \quad \sqsubset_R^{\circledast} \setminus Id \ = \sqsubset_R \cap \sqsubset_R^{-1}\ . \tag{6}$$

Moreover, for all distinct $a, b \in X$, we have:

$$\neg(a \rightleftharpoons_R b) \iff a \sqsubset_R b \ \wedge \ b \sqsubset_R a \tag{7}$$
$$\neg(a \sqsubset_R b) \implies b \sqsubset_R a \tag{8}$$
$$aRb \iff a \rightleftharpoons_R b \ \wedge \ a \sqsubset_R b \ . \tag{9}$$

and exactly one of the following holds:

$$\begin{array}{lllllll} a & \rightleftharpoons & b & \rightleftharpoons & a & \sqsubset & b \not\sqsubset a \\ a & \rightleftharpoons & b & \rightleftharpoons & a & \not\sqsubset & b \sqsubset a \\ a & \not\rightleftharpoons & b & \not\rightleftharpoons & a & \sqsubset & b \sqsubset a \ . \end{array} \tag{10}$$

Intuitively, (9) means that 'a occurred earlier than b' iff 'a and b were not simultaneous' and 'a occurred not later than b'.

3 Separable Order Structures

In this section we take another look at the stratified order structures, substantially different from that of, e.g., [8, 12, 15]. We provide for them a new representation more suitable for further generalisation. The new representation replaces causal orders by mutex relations between events. While SO-structures allow for more compact representation (strict precedence involves fewer pairs of events than mutex), the new order structures are easier to generalise to cater for general interleaving/mutex requirements and their properties.

In the rest of this paper, we will be concerned with order structures of the form $S = (X, \rightleftharpoons, \sqsubset)$. Intuitively, X is a set of events involved in some history of a concurrent system, $\rightleftharpoons$ is a 'mutex' (or 'interleaving') relation which identifies pairs of events which cannot occur simultaneously, and $\sqsubset$ is a 'weak precedence' relation between events. The latter means, in particular, that if $a \sqsubset b \sqsubset a$ then a and b must occur simultaneously in any execution belonging to the history represented by S; in other words, S must be *separable* (i.e., $\rightleftharpoons \cap \sqsubset^{\circledast} = \varnothing$).

Mutex order structures The definition of the first class of order structures based on mutex and weak precedence relations is motivated by the observation that the 'precedence' (or 'causality') relation is nothing but 'mutex'+'weak precedence', c.f. (9). Therefore, the axioms defining stratified order structures can easily be rendered in terms of the latter relations.

Definition 1 (mutex order structure). *A* mutex order structure *(*MO*-structure) is a relational structure mos* $= (X, \rightleftharpoons, \sqsubset)$*, where* $\rightleftharpoons$ *and* $\sqsubset$ *are binary relations on* X *such that, for all* $a, b, c \in X$*:*

$$\begin{array}{ll} M1: & a \rightleftharpoons b \implies b \rightleftharpoons a \\ M2: & a \not\sqsubset a \\ M3: & a \rightleftharpoons b \implies a \sqsubset b \ \vee \ b \sqsubset a \\ M4: & a \sqsubset b \sqsubset c \ \wedge \ a \neq c \implies a \sqsubset c \\ M5: & a \sqsubset b \sqsubset c \ \wedge (a \rightleftharpoons b \ \vee \ b \rightleftharpoons c) \implies a \rightleftharpoons c \ . \end{array}$$

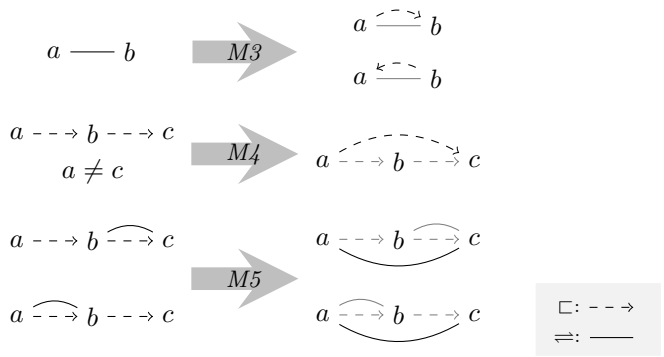


Fig. 3. A visualisation of axioms *M3* − *M5*.

Axioms *M3* − *M5* are illustrated in Figure 3, and some relevant properties of MO-structures are given below.

Proposition 1. *Let* $mos = (X, \rightleftharpoons, \sqsubset)$ *be an* MO*-structure. Then* mos *is separable and, for all* $a, b, c, d \in X$*, we have:*

$$a \not\rightleftharpoons a \tag{11}$$

$$a \sqsubset b \sqsubset a \ \wedge\ a \rightleftharpoons c \implies b \rightleftharpoons c \tag{12}$$

$$a \sqsubset c \sqsubset b \ \wedge\ a \sqsubset d \sqsubset b \ \wedge\ c \rightleftharpoons d \implies a \rightleftharpoons b\,. \tag{13}$$

The next results demonstrate that MO-structures are in a one-to-one relationship with SO-structures. Below, we use two mappings between these two classes of order structures. For every SO-structure $sos = (X, \prec, \sqsubset)$, we define $\mathsf{so2mo}(sos) = (X, \prec^{sym}, \sqsubset)$, and for every MO-structure $mos = (X, \rightleftharpoons, \sqsubset)$, we define $\mathsf{mo2so}(mos) = (X, \rightleftharpoons \cap \sqsubset, \sqsubset)$.

Theorem 1. *The mappings* mo2so *and* so2mo *are inverse bijections.*

Layered order structures In general, order structures like MO-structures are not saturated, and may capture histories comprising several executions (like a single causal partial order may have numerous total order linearisations). However, there is also a class of order structures which correspond in a one-to-one way to step sequences.

Definition 2. *Let* $R \subseteq X \times X$ *be a stratified order. Then* $los = (X, \rightleftharpoons_R, \sqsubset_R)$ *is the* layered order structure *(or* LO*-structure) induced by* R*.*

For a separable relational structure $sr = (X, \rightleftharpoons, \sqsubset)$, we will denote by $\mathsf{sr2los}(sr)$ the set of all LO-structures los extending sr, i.e., $sr \subseteq los$. With this notation, a nonempty set LOS of LO-structures is a concurrent history if $LOS = \mathsf{sr2los}\big(\bigcap LOS\big)$.

Proposition 2. *Every layered order structure is separable and saturated in the set of all separable order structures.*

Proposition 3. *Every* LO*-structure is an* MO*-structure.*

An MO-structure is linked with LO-structures (step sequences) through the set $\mathsf{sr2los}(mos)$ of all LO-structures extending mos. Similarly, for every SO-structure sos we can define $\mathsf{so2los}(sos) = \mathsf{sr2los}(\mathsf{so2mo}(sos))$. It can then be seen ([8]) that $\mathsf{so2los}(sos)$ is a nonempty set and (in the notation used in this paper):

$$sos = \bigcap \mathsf{mo2so}(\mathsf{so2los}(sos)) \ . \tag{14}$$

That result corresponds to Szpilrajn's Theorem that any partial order is the intersection of its linearisations (c.f. [5, 8]). Such a result extends to MO-structures and we obtain

Theorem 2. *For every* MO*-structure* mo, $\mathsf{sr2los}(mo) \neq \varnothing$ *and* $mo = \bigcap \mathsf{sr2los}(mo)$.

We can therefore conclude that the saturated extensions of an MO-structure mos form a concurrent history represented by mos. It is then important to ask which concurrent histories can be derived in this way; in other words, which concurrent histories can be represented by MO-structures.

Consider now a nonempty set $LOS = \{(X, \rightleftharpoons_i, \sqsubset_i) \mid i \in I\}$ of LO-structures forming a concurrent history, and their intersection $S = \bigcap LOS = (X, \rightleftharpoons, \sqsubset)$. Since every LO-structure is also an MO-structure, we immediately obtain that S is an order structure satisfying axioms *M1*, *M2*, *M4* and *M5*. However, *M3* in general does not hold although it holds for histories in which the possibility of executing two events in either order always implies also simultaneous execution, meaning that, for all distinct $a, b \in X$,

$$\left.\begin{array}{l} \exists i \in I : \ \langle a, b\rangle \ \in \rightleftharpoons_i \cap \sqsubset_i \\ \exists j \in I : \ \langle b, a\rangle \ \in \rightleftharpoons_j \cap \sqsubset_j \end{array}\right\} \implies \exists k \in I : \ \langle a, b\rangle \ \in \sqsubset_j^{sym} \ .$$

One might now wonder what happens if we do not assume any special properties of a concurrent history. As we will show in the rest of the paper, Proposition 1 in combination with the observation that it always holds for $S = \bigcap LOS$, yields axioms for order structures underpinning general concurrent histories.

4 Generalised Order Structures

In this section, we provide a complete characterisation of general concurrent histories where executions are represented by layered order structures; in other words, histories comprising step sequence executions. We achieve this by retaining all those MO-structure axioms which hold in general, and then replacing the only dropped axiom *M3* by Proposition 1.

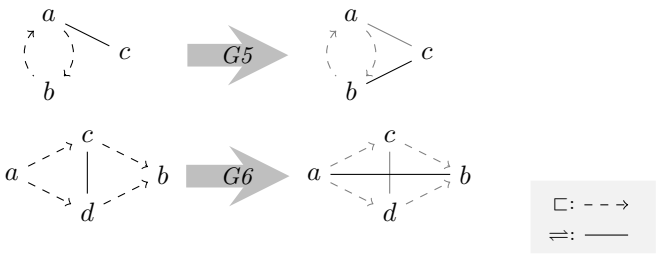


Fig. 4. A visualisation of axioms *G5* and *G6*.

Definition 3 (generalised mutex order structure). *A* generalised mutex order structure *(or* GMO*-structure) is a relational structure* $gmos = (X, \rightleftharpoons, \sqsubset)$*, where* $\rightleftharpoons$ *and* $\sqsubset$ *are binary relations on* X *such that, for all* $a, b, c, d \in X$*:*

$$
\begin{array}{lll}
G1: & a \rightleftharpoons b \implies b \rightleftharpoons a & M1 \\
G2: & a \not\sqsubset a \ \wedge \ a \neq a & M2 \ \& \ (11) \\
G3: & a \sqsubset b \sqsubset c \ \wedge \ a \neq c \implies a \sqsubset c & M4 \\
G4: & a \sqsubset b \sqsubset c \ \wedge (a \rightleftharpoons b \ \vee \ b \rightleftharpoons c) \implies a \rightleftharpoons c & M5 \\
G5: & a \sqsubset b \sqsubset a \ \wedge \ a \rightleftharpoons c \implies b \rightleftharpoons c & (12) \\
G6: & a \sqsubset c \sqsubset b \ \wedge \ a \sqsubset d \sqsubset b \ \wedge \ c \rightleftharpoons d \implies a \rightleftharpoons b & (13)
\end{array}
$$

The set of axioms in Definition 3 is minimal (see Figure 5). Moreover, GMO-structures enjoy a number of useful properties.

Proposition 4. *Let* $gmos = (X, \rightleftharpoons, \sqsubset)$ *be a* GMO*-structure. Then gmos is separable and, for all* $a, b \in X$*, we have:*

$$a \sqsubset^{\lambda} b \implies a \sqsubset b \qquad\qquad a \sqsubset b \sqsubset a \implies a \neq b \,.$$

Proposition 5. *Each* MO*-structure is a* GMO*-structure.*

The converse of Proposition 5 does not hold; for example, as *M3* does not hold, $(\{a, b\}, \{\langle a, b\rangle, \langle b, a\rangle\}, \varnothing)$ is a GMO-structure but *not* an MO-structure.

Proposition 6. *If* $gmos = (X, \rightleftharpoons, \sqsubset)$ *is a* GMO*-structure, then* $(X, \rightleftharpoons \cap \sqsubset, \sqsubset)$ *is an* SO*-structure.*

Proposition 6 states that every GMO-structure is a GSO-structure. We observe that the converse is not true, with suitable counterexamples provided by the GSO-structures S_{G5} and S_{G6} in Figure 5.

Closure operator for generalised mutex order structures We will now provide a method for deriving valid GMO-structures from other relational structures.

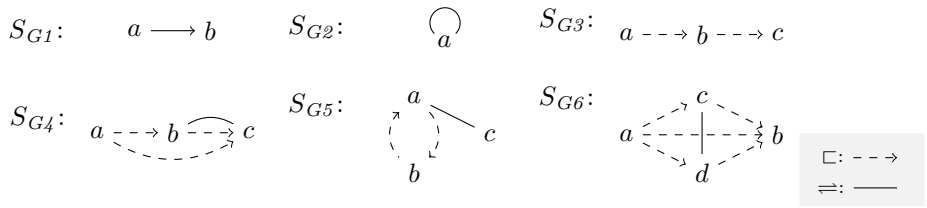


Fig. 5. Examples showing that the set of axioms in Definition 3 is minimal. Each relational structure S_{Gi} satisfies all axioms except for Gi.

Definition 4 (GMO-closure). *Let $S = (X, Q, R)$ be a relational structure and $Q^{[R]} = R^{\circledast} \circ \big(Q \cup (R^* \circ_Q R^*)^{sym}\big) \circ R^{\circledast}$. Then the* GMO*-closure of S is given by:*

$$S^{\blacklozenge} = \big(X \ , \ Q^{[R]} \ , \ R^{\curlywedge}\big) \ .$$

The GMO-closure operator introduced here can be seen as related to two different closure operators: (i) the transitive closure operator for acyclic reflexive binary relations; and (ii) the $\Diamond$-operator for $\Diamond$-acyclic order structures introduced in [7] in order to obtain SO-structures. It is also be seen as a generalisation of the GSO-closure introduced in [10] for GSO-acyclic structures in order to obtain GSO-structures.

The main property we want from the notion of GMO-closure is that whenever $S = (X, Q, R)$ is a separable relational structure, $S^{\blacklozenge}$ is a GMO-structure. Furthermore, if S is already a GMO-structure, then we want $S^{\blacklozenge} = S$. The form of $Q^{[R]}$ follows from the requirement that $S^{\blacklozenge}$ should be a GMO-structure and the axioms for GMO-structures. In particular the factor $(R^* \circ_Q R^*)^{sym}$ follows from axioms *G4* and *G6*, while the factor $R^{\circledast} \circ_Q R^{\circledast}$ corresponds to *G5*.

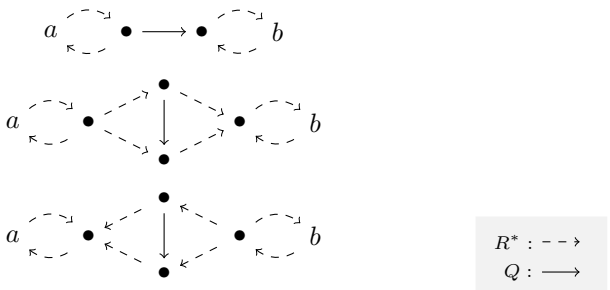


Fig. 6. A visualisation of the three cases of $\langle a, b\rangle \in Q^{[R]}$.

The next four results respectively correspond to saying that: (i) the transitive closure of an acyclic relation is also acyclic; (ii) GMO-closure is a closure operation in the usual sense; (iii) the transitive closure of an acyclic relation yields a poset; and (iv) posets are transitively closed.

Proposition 7. *If S is a separable relational structure, then $S^{\blacklozenge}$ is also separable, $S \subseteq S^{\blacklozenge}$ and $(S^{\blacklozenge})^{\blacklozenge} = S^{\blacklozenge}$. Moreover, $S^{\blacklozenge}$ is a* GMO*-structure.*

Proposition 8. *If gmos is a* GMO*-structure, then $gmos^{\blacklozenge} = gmos$.*

As layered order structures and mutex order structures are special cases of generalised mutex order structures, we obtain an immediate

Corollary 1. *Let los be an* LO*-structure and mos be an* MO*-structure. Then $los^{\blacklozenge} = los$ and $mos^{\blacklozenge} = mos$.*

The following technical lemma describes a single stage of the saturation process for a GMO-structure leading to a LOS-structure. In such a process, we may add an arbitrary link between two elements that do not yet satisfy (10). We only need to remember that in the case of extending the relation Q, together with $\langle a, b\rangle$ we have to add $\langle b, a\rangle$. After such an addition, we get a separable order structure that may be closed. As a result, we obtain one of possible extensions of an initial *gmos*. The above process terminates after obtaining an LO-structure and it is central to the proof of the main Theorem 3.

In what follows, we denote $R_{xy} = R \cup \{\langle x, y\rangle\}$ and $Q_{xyx} = Q \cup \{\langle x, y\rangle, \langle y, x\rangle\}$.

Lemma 1. *Let $gmos = (X, Q, R)$ be a* GMO*-structure, $a, b \in X$ and $a \neq b$.*

$$\begin{array}{lcll} \langle a, b\rangle \notin R \ \wedge \ \langle b, a\rangle \notin R & \Longrightarrow & (X, Q, R_{ab})^{\blacklozenge} & \textit{is a GMO-structure} \\ \langle a, b\rangle \notin R \ \wedge \ \langle a, b\rangle \notin Q & \Longrightarrow & (X, Q, R_{ab})^{\blacklozenge} & \textit{is a GMO-structure} \\ \langle a, b\rangle \notin R \ \wedge \ \langle a, b\rangle \notin Q & \Longrightarrow & (X, Q_{aba}, R)^{\blacklozenge} & \textit{is a GMO-structure}\,. \end{array}$$

To complete the properties of the saturation process described in Lemma 1 and used in the proof of Theorem 3, we formulate the following

Lemma 2. *Let $gmos = (X, Q, R)$ be a* GMO*-structure such that $a, b \in X$, $a \neq b$, $\langle a, b\rangle \notin R$ and $\langle a, b\rangle \notin Q$ and $S' = (X, Q, R_{ab})^{\blacklozenge} = (X, Q', R^{\lambda}_{ab})$. Then $\langle a, b\rangle \notin Q'$.*

In Lemmas 1 and 2 we have captured a method of saturating GMO-structures that are not LO-structures. It moreover allows us to formulate an immediate

Corollary 2. *Every relational structure saturated among all separable relational structures is a layered order structure.*

General concurrent histories We now return to our original goal which was to provide a structural characterisation of all histories comprising step sequence executions. Recall that sr2los(*gmos*) is the set of all LO-structures associated with a GMO-structure *gmos*. Then we obtain a result corresponding to Szpilrajn's Theorem:

Theorem 3. *For every* GMO*-structure gmos,*

$$\mathsf{sr2los}(gmos) \neq \varnothing \quad \textit{and} \quad gmos = \bigcap \mathsf{sr2los}(gmos) \ .$$

Together with the fact that, for every nonempty set LOS of LO-structures with the same domain, $\bigcap LOS$ is a GMOS-structure, this leads to the conclusion that all concurrent histories are represented by GMO-structures.

5 Concluding Remarks

We can finally clarify the relationship between GSO-structures and GMO-structures. In general, in order to accept an order structure $os = (X, \rightleftharpoons, \sqsubset)$ as an invariant representation of a concurrent history, we require that

$$\mathsf{sr2los}(os) \neq \varnothing \ \text{ and } \ os = \bigcap \mathsf{sr2los}(os) \ .$$

We demonstrated that this property holds whenever os is a GMO-structure, and that it may fail to hold for a GSO-structure. We have further shown that GMO-structures are GSO-structures, but that the converse does not hold. However, what is the case is that each GSO-structure $gsos$ is separable, and so its GMO-closure $gsos^{\blacklozenge}$ is a GMO-structure satisfying $\mathsf{sr2los}(gsos^{\blacklozenge}) = \mathsf{sr2los}(gsos)$. In other words, concurrent histories described by separable order structures and their GMO-closures are the same. The importance of GSO-structures comes from the fact that they paved the way for GMO-structures, by exposing the fundamental property that causal ordering is a combination of mutex and weak ordering.

A key motivation for the research presented in this paper comes from concurrent behaviours as exhibited by safe Petri nets with mutex arcs. The resulting semantical approach — which has been meticulously worked out above — is based on GMO-structures which characterise all concurrent histories comprising step sequence executions. A natural direction for further work is to provide a compatible language-theoretic representation of concurrent histories, by generalising Mazurkiewicz traces [13] which correspond to causal posets, and comtraces [7] which correspond to SO-structures (or MO-structures). This development would also allow to link the dynamic notions of mutex and weak causality with the static properties of Petri nets with mutex arcs. The resulting semantics can also support efficient verification techniques [2, 14, 18].

Acknowledgements

We would like to thank the anonymous reviewers for useful comments and suggestions. This research was supported by a fellowship funded by the "Enhancing Educational Potential of Nicolaus Copernicus University in the Disciplines of Mathematical and Natural Sciences" Project POKL.04.01.01-00-081/10, the EPSRC GAELS and UNCOVER projects, and by an NSERC of Canada grant.

References

1. Best, E., Devillers, R.: Sequential and concurrent behaviour in Petri net theory. TCS 55(1), 87–136 (1987)
2. Esparza, J., Heljanko, K.: Unfoldings: A Partial-Order Approach to Model Checking. Monographs in Theoretical Computer Science, Springer (2008)
3. Gaifman, H., Pratt, V.R.: Partial order models of concurrency and the computation of functions. In: LICS. pp. 72–85 (1987)
4. Guo, G., Janicki, R.: Modelling concurrent behaviours by commutativity and weak causality relations. In: AMAST. pp. 178–191 (2002)
5. Janicki, R.: Relational structures model of concurrency. Act. Inf. 45, 279–320 (2008)
6. Janicki, R., Koutny, M.: Structure of concurrency. TCS 112(1), 5–52 (1993)
7. Janicki, R., Koutny, M.: Semantics of inhibitor nets. Inf.&Comp. 123(1), 1 – 16 (1995)
8. Janicki, R., Koutny, M.: Fundamentals of modelling concurrency using discrete relational structures. Acta Inf. 34, 367–388 (1997)
9. Juhás, G., Lorenz, R., Mauser, S.: Causal semantics of algebraic Petri nets distinguishing concurrency and synchronicity. Fundam. Inf. 86(3), 255–298 (2008)
10. Kleijn, J., Koutny, M.: The mutex paradigm of concurrency. In: Petri Nets. pp. 228–247 (2011)
11. Kleijn, J., Koutny, M.: Mutex causality in processes and traces. Fundam. Inf. 122, 119–146 (2013)
12. Le, D.T.M.: On three alternative characterizations of combined traces. Fundam. Inf. 113(3), 265–293 (2011)
13. Mazurkiewicz, A.: Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University (1977)
14. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: CAVC. pp. 164–177 (1992)
15. Mikulski, Ł., Koutny, M.: Hasse diagrams of combined traces. In: Brandt, J., Heljanko, K. (eds.) ACSD. pp. 92–101 (2012)
16. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, Part I. TCS 13, 85–108 (1981)
17. Pratt, V.R.: Some constructions for order-theoretic models of concurrency. In: Logic of Programs. pp. 269–283 (1985)
18. Rodriguez, C., Schwoon, S., Khomenko, V.: Contextual merged processes. In: Petri Nets. LNCS, vol. 7927, pp. 29–48 (2013)
19. Schröder, E.: Vorlesungen über die Algebra der Logik (Exakte Logik). Dritter Band: Algebra und Logik der Relative. B. G. Teubner (1895)
20. Szpilrajn, E.: Sur l'extension de l'ordre partiel. Fundam. Math. 16, 386–389 (1930)
21. Winkowski, J., Maggiolo-Schettini, A.: An algebra of processes. Journal of Computer and System Sciences 35(2), 206–228 (1987)

Interactive Complex Granules

Andrzej Jankowski[1], Andrzej Skowron[2], and Roman Swiniarski[3⋆]

[1] Institute of Computer Science, Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
a.jankowski@ii.pw.edu.pl
[2] Institute of Mathematics, The University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
skowron@mimuw.edu.pl
[3] Department of Computer Science, San Diego State University
5500 Campanile Drive San Diego, CA 92182, USA
and
Institute of Computer Science Polish Academy of Sciences
Jana Kazimierza 5, 01-248 Warsaw, Poland
rswiniarski@mail.sdsu.edu

As far as the laws of mathematics refer to reality,
they are not certain; and as far as they are certain,
they do not refer to reality.
– *Albert Einstein* ([2])

Constructing the physical part of the theory and unifying it
with the mathematical part should be considered as one of
the main goals of statistical learning theory
– *Vladimir Vapnik*
([24], *Epilogue: Inference from sparse data, p. 721*)

Abstract. Information granules (infogranules, for short) are widely discussed in the literature. In particular, let us mention here the rough granular computing approach based on the rough set approach and its combination with other approaches to soft computing. However, the issues related to interactions of infogranules with the physical world and to perception of interactions in the physical world by infogranules are

⋆ This work was supported by the Polish National Science Centre grants 2011/01/B/ST6/03867, 2011/01/D/ST6/06981, and 2012/05/B/ST6/03215 as well as by the Polish National Centre for Research and Development (NCBiR) under the grant SYNAT No. SP/I/1/77065/10 in frame of the strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information" and the grant No. O ROB/0010/ 03/001 in frame of the Defence and Security Programmes and Projects: "Modern engineering tools for decision support for commanders of the State Fire Service of Poland during Fire & Rescue operations in the buildings"

not well elaborated yet. On the other hand the understanding of interactions is the critical issue of complex systems. We propose to model complex systems by interactive computational systems (ICS) created by societies of agents. Computations in ICS are based on complex granules (c-granules, for short). In the paper we concentrate on some basic issues related to interactive computations based on c-granules performed by agents in the physical world.

Key words: granular computing, rough set, interaction, information granule, physical object, complex granule, interactive computational system

1 Introduction

Granular Computing (GC) is now an active area of research (see, e.g., [16]). Objects we are dealing with in GC are *information granules* (or *infogranules*, for short). Such granules are obtained as the result of information granulation [26, 28]:

> *Information granulation can be viewed as a human way of achieving data compression and it plays a key role in implementation of the strategy of divide-and-conquer in human problem-solving.*

The concept of granulation is rooted in the concept of a linguistic variable introduced by Lotfi Zadeh in 1973 [25]. Information granules are constructed starting from some elementary ones. More compound granules are composed of finer granules that are drawn together by indistinguishability, similarity, or functionality [27].

Computations on granules should be interactive. This requirement is fundamental for modeling of complex systems [3]. For example, in [13] this is expressed as follows

> *[...] interaction is a critical issue in the understanding of complex systems of any sorts: as such, it has emerged in several well-established scientific areas other than computer science, like biology, physics, social and organizational sciences.*

Interactive Rough Granular Computing (IRGC) is an approach for modeling interactive computations (see, *e.g.*, [17, 19–23]). Computations in IRGC are progressing by interactions represented by interactive information granules. In particular, interactive information systems (IIS) are dynamic granules used for representing the results of the agent interaction with the environments. IIS can be also applied in modeling of more advanced forms of interactions such as hierarchical interactions in layered granular networks or generally in hierarchical modeling. The proposed approach [17, 19–23] is based on rough sets but it can be combined with other soft computing paradigms such as fuzzy sets or evolutionary computing, and also with machine learning and data mining techniques.

The notion of the highly interactive granular system is clarified as the system in which intrastep interactions [4] with the external as well as with the internal environments take place. Two kinds of interactive attributes are distinguished: perception attributes, including sensory ones and action attributes.

In this paper we extend the existing approach by introducing complex granules (c-granules) making it possible to model interactive computations performed by agents. Any c-granule consists of three components, namely soft_suit, link_suit and hard_suit. These components are making it possible to deal with such abstract objects from soft_suit as infogranules as well as with physical objects from hard_suit. The link_suit of a given c-granule is used as a kind of c-granule interface for expressing interaction between soft_suit and and hard_suit. Any agent operates in a local world of c-granules. The agent control is aiming to control computations performed by c-granules from this local world for achieving the target goals. Actions (sensors or plans) from link_suits of c-granules are used by the agent control in exploration and/or exploitation of the environment on the way to achieve their targets. C-granules are also used for representation of perception by agents of interactions in the physical world. Due to the bounds of the agent perception abilities usually only a partial information about the interactions from physical world may be available for agents. Hence, in particular the results of performed actions by agents can not be predicted with certainty.

In Section 2 a general structure of c-granules is described and some illustrative examples are included. Moreover, some preliminary concepts related to agents performing computations on c-granules are discussed. In Section 3 the agent architecture is outlined. Societies of agents and communication languages are discussed shortly in Section 4.

This paper is a step in the realization of the Wisdom Technology (WisTech) programme [6–8].

2 Complex Granules and Physical World

We define the basic concepts related to c-granule relative to a given agent ag. We assume that the agent ag has access to a local clock making it possible to use the local time scale. In this paper we consider discrete linear time scale.

We distinguish several kinds of objects in the environment in which the agent ag operates:

- *physical objects* (called also as *hunks of matter*, or *hunks*, for short) [5] such as physical parts of agents or robots, specific media for transmitting information; we distinguish hunks called as artifacts used for labeling other hunks or stigmergic markers used for indirect coordination between agents or actions [9]; note that hunks may change in time and are perceived by the agent ag as dynamic (systems) processes; any hunk h at the local time t of ag is represented by the state $st_h(t)$; the results of perception of hunk states by agent ag are represented by value vector of relevant attributes (features);
- *complex granules* (c-*granules*, for short) consisting of three parts: *soft_suit*, *link_suit*, and *hard_suit* (see Figure 1); c-granule at the local time t of ag is

denoted by G; G receives some inputs and produces some outputs; inputs and outputs of c-granule G are c-granules of the specified admissible types; input admissible type is defined by some input preconditions and the output admissible type is defined by some output postconditions, there are distinguished inputs (outputs) admissible types which receive (send) c-granules from (to) the agent ag control;

- G soft_suit consists of
  1. G name, describing the behavioral pattern description of the agent ag corresponding to the name used by agent for identification of the granule,
  2. G type consisting of the types of inputs and outputs of G c-granule,
  3. G status (*e.g.*, active, passive),
  4. G information granules (infogranules, for short) in mental imagination of the agent consisting, in particular of G specification, G implementation and manipulation method(s); any implementation distinguished in infogranule is a description in the agent ag language of transformation of input c-granules of relevant types into output c-granules of relevant types, *i.e.*, any implementation defines an interactive computation which takes as input c-granules (of some types) and produces some c-granules (of some types); inputs for c-granules can be delivered by the agent ag control (or by other c-granules), we also assume that the outputs produced by a given c-granule depend also on interactions of hunks pointed out by link_suite as well as some other hunks from the environment - in this way the semantics of c-granules is established;
- G link_suit consists of
  1. a representation of configuration of hunks at time t (*e.g.*, mereologies of parts in the physical configurations perceived by the agent ag);
  2. links from different parts of the configuration to hunks;
  3. G links and G representations of elementary actions; using these links the agent ag may perform sensory measurement or/and actions on hunks; in particular, links are pointing to the sensors or effectors in the physical world used by the considered c-granule; using links the agent ag may, *e.g.*, fix some parameters of sensors or/and actions, initiate sensory measurements or/and action performance; we also assume that using these links the agent ag may encode some information about the current states of the observed hunks by relevant information granules;
- G hard_suit is created by the environment of interacting hunks encoding G soft_suit, G link_suit and implementing G computations;
- soft_suit and link_suit of G are linked by G links for interactions between the G hunk configuration representation and G infogranules;
- link_suit and hard_suit are linked by G links for interactions between the G hunk configuration representation and hunks in the environment.

The interactive processes during transforming inputs of c-granules into outputs of c-granules is influenced by

1. interaction of hunks pointed out by link_suit;
2. interaction of pointed hunks with relevant parts of configuration in link_suit.

Agent can establish, remember, recognize, process and modify relations between c-granules or/and hunks.

A general structure of c-granules is illustrated in Figure 1.

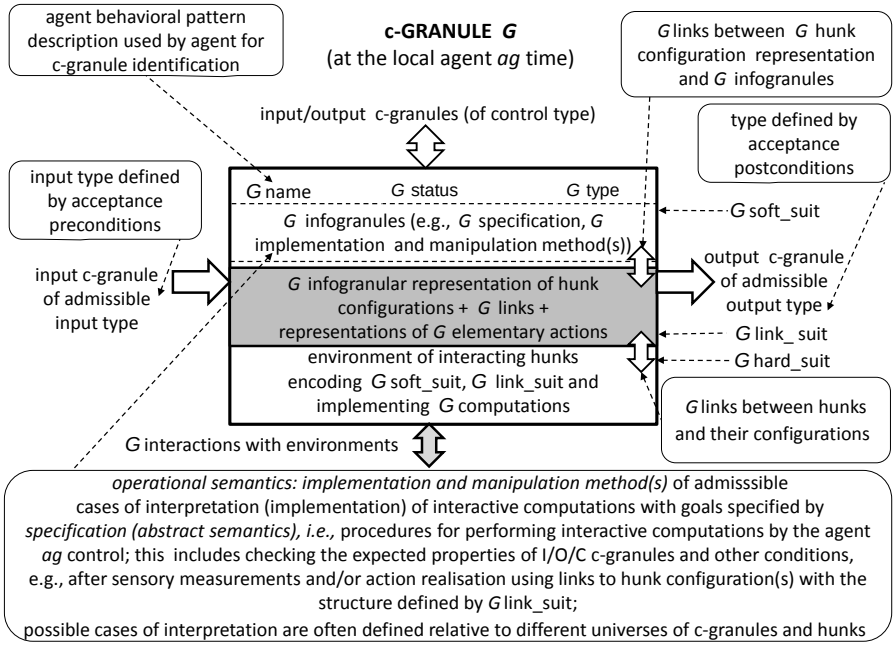


Fig. 1. General structure of c-granules

In Figure 2 we illustrate how the abstract definition of operation from soft_link interacts with other suits of c-granule. It is necessary to distinguish two cases. In the first case, the results of operation $\otimes$ realized by interaction of hunks are consistent with the specification in the suit_link. In the second case, the result specified in the soft_suit can be treated only as an estimation of the real one which may be different due to the unpredictable interactions in the hard_suit. Figure 3 illustrates c-granules corresponding to sensory measurement. Note that in this case, the parameters fixed by the agent control may concern sensor selection, selection of the object under measurement by sensor and selection of sensor parameters. They are interpreted as actions selected from the link_suit. In the perception of configuration of hunks of c-granule are distinguished infogranules representing sensor, object under measurement and the configuration itself. The links selected by the agent control represent relations between states of hunks and infogranules corresponding to them in the link_suit.

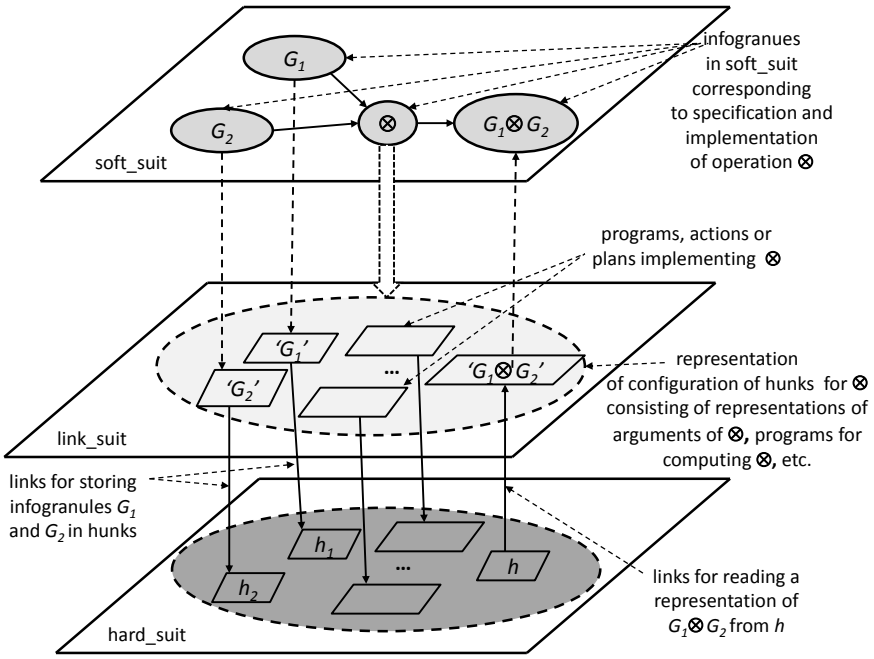


Fig. 2. Explanation of roles of different suits of a c-granule for operation ⊗

Figure 4 illustrates how an interactive information (decision) system is created and updated during running of c-granule implementation according to scenario(s) defined in the soft_suit and related G links. Such information (decision) systems are used for recording information about the computation steps during c-granule implementation run. Note that the structure of this information system is different from the classical definition [14, 15, 18]. In particular, this system is open because of links to physical objects as well as interactions are changing (often in unpredictable way) in time. In our approach, the agent can be also interpreted as c-granule. However, this is a c-granule of higher order with embedded control. One can also consider another situation when the c-granules are autonomous but this is out of scope of this article. Instead of this one can consider interactions in societies of agents. We assume that for any agent ag there is distinguished a family of *elementary c-granules* and constructions on c-granules leading to more compound c-granules. The agent ag is using the constructed granules for modeling attention and interaction with the environment. Note that for any new construction on elementary granules (such as network of c-granules) should be defined the corresponding c-granule. This c-granule should have appropriate soft_suit, link_suit and hard_suit so that the constructed c-granule will satisfy the conditions of the new c-granule construction specification. Note that one of the constraints on such construction may follow from the interactions which the agent ag will have at the disposal in the uncertain environment.

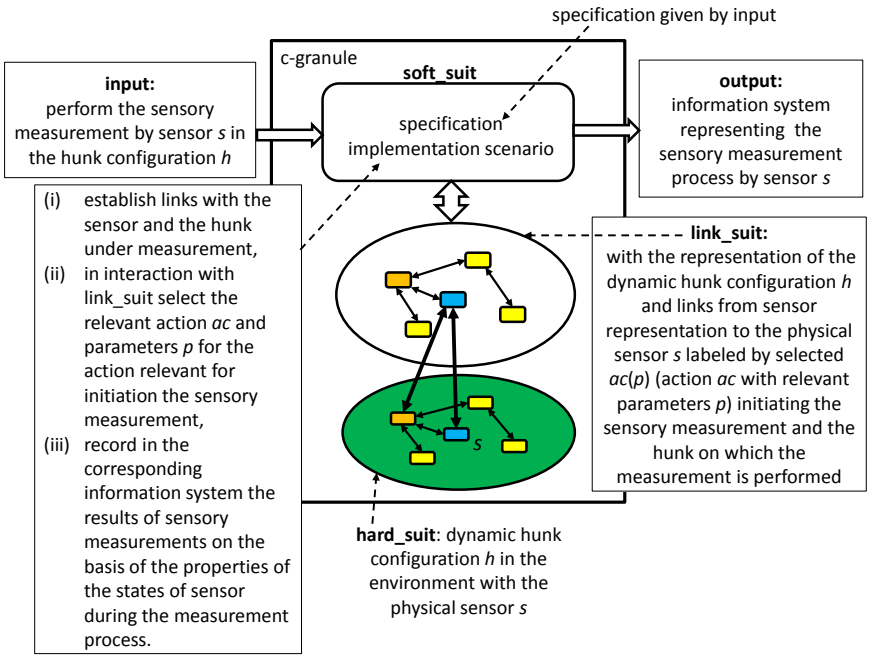


Fig. 3. Interactions caused by sensors

3 Agent Architecture Framework

Agents may be treated as generalized c-granules with embedded control structure.

Any agent ag is defined over several classes of c-granules. Among them are:

- senbot (sensory bot) - class of c-granules representing possible states of the agent sensory measurements with at most one distinguished c-granule at the local time moment t of agent ag;
- imbot (imagination bot)- class of all possible c-granules which can be constructed by the agent ag from sensory measurements with at most one distinguished c-granule at the local time moment t of agent ag;
- embot (emotional bot)- subclass of imbot class representing emotional concepts of the agent ag;
- nebot (needs bot)- subclass of imbot class representing concepts of the agent ag needs;
- enabot (environment action bot) - subclass of imbot class specifying the agent ag elementary actions in the environment;
- imobot (imagination operation bot) - subclass of imbot class specifying the agent ag elementary operations (different from elementary actions) on c-granules from imbot;

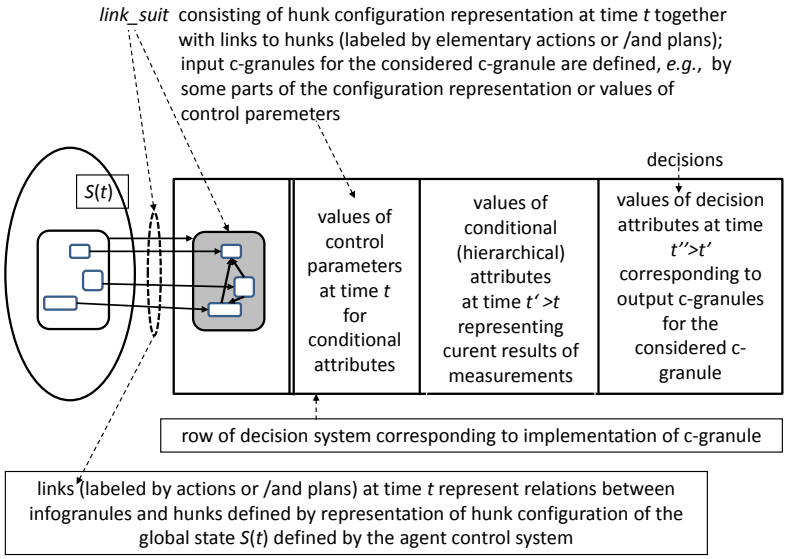


Fig. 4. Example: Row of interactive information (decision) system corresponding to registration of computation of c-granule according to implementation scenario

- abot (attention bot) - subclass of imbot class representing c-granules currently under attention by the agent ag;
- activebot - subclass of imbot class representing c-granules currently active;
- passivebot - subclass of imbot class representing c-granules currently passive;
- metbot (method bot) - subclass of imbot representing methods of manipulation on c-granules (construction, destruction, modification, join, classifiers construction);
- metabot (method adaptation bot) subclass of imbot representing c-granules used for adaptation or/and modification of the given methods of manipulation on c-granules.

The language of c-granule names consists of

- set of names of existing c-granules;
- set of names of new generated c-granules.

Types of objects relative to c-granules in imbot:

- set of types of existing c-granules;
- set of types of new generated c-granules.

There are some distinguished c-granules of the agent ag:

- Meaning relation (Mean) - a distinguished c-granule representing a relation between c-granules and their names.

- Type relation (TypeMean) - a distinguished c-granule representing a relation between c-granules and their types.
- Reference relation (Ref) - a distinguished c-granule representing a relation between c-granules and 'related' names.
- Jbot (Judgment bot) - a distinguished c-granule representing actual collection of strategies of approximate reasoning used by the agent ag for judgment and risk assessment in the current environment and agent situation.
- Cobot (control bot) - a distinguished c-granule representing actual collection of strategies of approximate reasoning used by the agent ag for control, adaptation, and modification of all the agent ag c-granules.
- Metacobot (meta-control bot) - a distinguished c-granule representing actual collection of strategies of approximate reasoning used by the agent ag for cobot control, adaptation, and modification.

The generalized c-granules corresponding to agents are defined using also the above classes of c-granules for defining corresponding suits of such generalized c-granules. The details of such construction will be presented in our next papers. Here, we would like to note only that there is a quite general approach for defining new c-granules from the simpler already defined.

Figure 5 illustrates an idea of transition relation related to a given agent ag. The relation is defined between configurations of ag at time t and the measurement time next to t. A configuration of ag at time t consists of all configurations of c-granules existing at time t. A configuration of c-granule G at time t consists of G itself as well as all c-granules selected on the basis of links in the link_suit of G at time t. These are, in particular all c-granules pointed by links corresponding to the c-granules stored in the computer memory during the computation process realised by c-granule as well as c-granules corresponding to perception at time t of the configuration of hunks at time t.

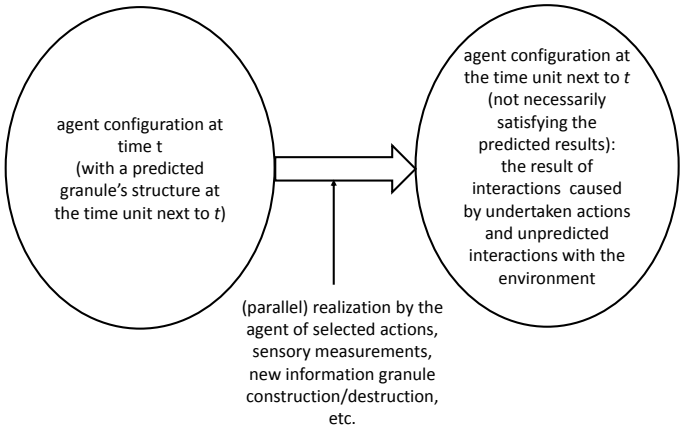


Fig. 5. Transiton relation of the agent ag

4 Societies of Agents and Communication Languages

We assume that the agents can perceive behavioral patterns of other agents of their groups and based on this they can try to exchange some messages [10]. It is worthwhile to mention that at the beginning agents do not have common understanding of the meaning of such messages. In the consequence, this leads to misunderstanding, not comfortable situation for agents (in terms of hierarchy of their needs represented by nebot). However, after series of trials they have a chance to set up common meaning of some behavioral patterns. In other words, they start to create common c-granules which use agreed links to other hunks or infogranules and also descriptions of some details about actions related to meaning or methods of implementation of the infogranule contents. For example, at the beginning the messages could be linked to warning situations or to identifications of some sources for satisfiability of some agent needs. This kind of simple messages could be passed by very simple behavioral pattern. Next, based on these very simple behavioral patterns the agents can develop more compound messages related to c-granules corresponding to common plans of cooperation of group of agents or/and competition with other groups of agents. This very general framework could be implemented in many ways using different AI paradigms. Especially, many models from Natural Computing could be quite helpful (*e.g.*, modification of cellular automata or evolutionary programming). However, our proposal is to implement this general scheme by agents having soft_suit and link_suit built up on the hierarchies of interactive information (decision) systems linked to configurations of hunks. Starting from the simplest case when we have just one attribute and one message to be passed up to quite complex system this approach based on rough sets is quite convenient for implementation by computers well prepared for manipulation on tables of data.

It has to be underlined that the behavioral patterns are complex vague concepts. Hence, some advanced methods for approximation of these concepts should be used. Usually these methods are based on hierarchical learning (see, e.g., [11, 1]). Note that often in satisfiability checking for vague concept, actions or/and plans are used. In the rough set approach it is important to remember that the attribute values are given only for some examples from reality. Moreover, if we use a large number of attributes or/ and hierarchical learning this will not guarantee the exact description of reality in terms of perceived vague concepts.

Languages of agents consist of partial descriptions of situations (or their indiscernibility or similarity classes) perceived by agents as well as description of approximate reasoning schemes about the situations and their changes by actions and /or plans. The situations may be represented in hierarchical modeling by structured objects (*e.g.*, relational structures over attribute value vectors or/and indiscernibility (similarity classes) of such structures). In reasoning about the situation changes one should take into account that the predicted actions or/and planes may depend not only on the changes of past situations but also on the performed actions and plane in the past. This is strongly related to the idea of perception pointed out in [12]:

The main idea of this book is that perceiving is a way of acting. It is something we do. Think of a blind person tap-tapping his or her way around a cluttered space, perceiving that space by touch, not all at once, but through time, by skillful probing and movement. This is or ought to be, our paradigm of what perceiving is.

Figure 6 illustrates this idea.

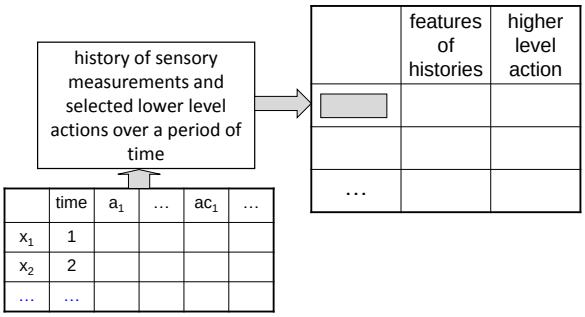


Fig. 6. Action in perception.

Note that the expression of the language may be used without its 'support' in corresponding link_suit and hard_suit of c-granules under the assumption that there are fixed coding methods between expressions and hunks by the agent. However, the languages should contain more general expressions for communication usually requiring the usage of expressions representing classes of hunks rather than single hunks. This follows from the fact that the agents have bounded abilities for discernibility of perceived objects. In our approach the situations and reasoning schemes about situations are represented by c-granules.

Note that different behavioral patterns may be indiscernible relative to the set of attributes used by the agent. Hence, it follows that the agents perceive objects belonging to the same indiscernibility or/and similarity class in the same way. This is an important feature making it possible to use generalization by agents. For example, the situations classified by a given set of characteristic functions of induced classifiers (used as attributes) may be indiscernible. On the other hand, a new situation unseen so far may be classified to indiscernibility classes which allows agents to make generalizations. The new names created by agents are names of new structured objets or their indiscernibility (similarity) classes.

Agents should be equipped with adaptation strategies for discovery of new structured objects and their features (attributes). This is the consequence of the fact that the agents are dealing with vague concepts. Hence, the approximations of these concepts represented by the induced classifiers evolve with changes in uncertain data and imperfect knowledge.

5 Conclusions and Future Research

The outlined research on the nature of interactive computations is crucial for understanding complex systems. Our approach is based on complex granules (c-granules) performing computations through interaction with physical objects (hunks). Computations of c-granules are controlled by the agent control. More compound c-granules create agents and societies of agents. Other issues outlined in this paper such as interactive computations performed by societies for agents, especially communication language evolution and risk management in interactive computations will be discussed in more detail in our next papers.

References

1. J. Bazan. Hierarchical classifiers for complex spatio-temporal concepts. Transactions on Rough Sets IX: Journal Subline LNCS 5390 (2008) 474–750.
2. A. Einstein. Geometrie und Erfahrung (Geometry and Experience). Julius Springer, Berlin, 1921.
3. D. Goldin, S. Smolka, P. Wegner (Eds.). Interactive Computation: The New Paradigm. Springer, 2006.
4. Y. Gurevich. Interactive algorithms 2005. In: Goldin et al. [3], pp. 165–181.
5. M. Heller. The Ontology of Physical Objects. Four Dimensional Hunks of Matter. Cambridge Studies in Philosophy, Cambridge University Press, Cambridge, UK, 1990.
6. A. Jankowski, A. Skowron. A wistech paradigm for intelligent systems. Transactions on Rough Sets VI: Journal Subline LNCS 4374 (2007) 94–132.
7. A. Jankowski, A. Skowron. Wisdom technology: A rough-granular approach. In: M. Marciniak, A. Mykowiecka (Eds.), Bolc Festschrift, Springer, Heidelberg, Lectures Notes in Computer Science, vol. 5070. 2009, pp. 3–41.
8. A. Jankowski, A. Skowron. Practical Issues of Complex Systems Engineering: Wisdom Technology Approach. Springer, Heidelberg, 2014. (in preparation).
9. L. Marsh. Stigmergic epistemology, stigmergic cognition. Journal Cognitive Systems 9 (2008) 136–149.
10. M. Minsky. Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Mind. Simon & Schuster, New York, 2006.
11. S. H. Nguyen, J. Bazan, A. Skowron, H. S. Nguyen. Layered learning for concept synthesis. Transactions on Rough Sets I: Journal Subline LNCS 3100 (2004) 187–208.
12. A. Noë. Action in Perception. MIT Press, 2004.
13. A. Omicini, A. Ricci, M. Viroli. The multidisciplinary patterns of interaction from sciences to computer science. In: Goldin et al. [3], pp. 395–414.
14. Z. Pawlak. Information systems - theoretical foundations. Information Systems 6 (1981) 205–218.
15. Z. Pawlak. Rough Sets: Theoretical Aspects of Reasoning about Data, System Theory, Knowledge Engineering and Problem Solving, vol. 9. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.
16. W. Pedrycz, S. Skowron, V. Kreinovich (Eds.). Handbook of Granular Computing. John Wiley & Sons, Hoboken, NJ, 2008.
17. A. Skowron, J. Stepaniuk, R. Swiniarski. Modeling rough granular computing based on approximation spaces. Information Sciences 184 (2012) 20–43.

18. A. Skowron, Z. Suraj (Eds.). Rough Sets and Intelligent Systems. Professor Zdzislaw Pawlak in Memoriam. Series Intelligent Systems Reference Library, Springer, 2013.
19. A. Skowron, M. Szczuka. Toward interactive computations: A rough-granular approach. In: J. Koronacki, Z. Raś, S. Wierzchoń, J. Kacprzyk (Eds.), Advances in Machine Learning II: Dedicated to the Memory of Professor Ryszard S. Michalski, Springer, Heidelberg, Studies in Computational Intelligence, vol. 263. 2009, pp. 23–42.
20. A. Skowron, P. Wasilewski. Information systems in modeling interactive computations on granules. In: M. Szczuka, M. Kryszkiewicz, S. Ramanna, R. Jensen, Q. Hu (Eds.), Proceedings of RSCTC 2010, Springer, Heidelberg, Lectures Notes in Computer Science, vol. 6086. 2010, pp. 730–739.
21. A. Skowron, P. Wasilewski. Information systems in modeling interactive computations on granules. Theoretical Computer Science 412(42) (2011) 5939–5959.
22. A. Skowron, P. Wasilewski. Toward interactive rough-granular computing. Control & Cybernetics 40(2) (2011) 1–23.
23. A. Skowron, P. Wasilewski. Interactive information systems: Toward perception based computing. Theoretical Computer Science 454 (2012) 240–260.
24. V. Vapnik. Statistical Learning Theory. John Wiley & Sons, New York, NY, 1998.
25. L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. on Systems, Man and Cybernetics SMC-3 (1973) 28–44.
26. L. A. Zadeh. Fuzzy sets and information granularity. In: Advances in Fuzzy Set Theory and Applications, North-Holland, Amsterdam. 1979, pp. 3–18.
27. L. A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. Fuzzy Sets and Systems 90 (1997) 111–127.
28. L. A. Zadeh. A new direction in AI: Toward a computational theory of perceptions. AI Magazine 22(1) (2001) 73–84.

Identification of Formal Fallacies in a Natural Dialogue

Magdalena Kacprzak[1] and Anna Sawicka[2]

[1] Faculty of Computer Science, Białystok University of Technology, Poland
[2] Faculty of Computer Science, Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract. This paper is a continuation of the work presented at CS&P 2012 where the LND dialogue system was proposed. It brings together and unifies two traditions in studying dialogue as a game: the dialogical logic introduced by Lorenzen and persuasion dialogue games as specified by Prakken. The aim of the system LND is to recognize and verify formal fallacies in dialogues. Now we extend this system with a new protocol which allows for reconstruction of natural dialogues in which parties can be committed to formal fallacies. Finally, we show the implementation of the applied protocols.

Keywords: protocol for dialogue games, natural dialogue, formal fallacy, Lorenzen Natural Dialogue

1 Introduction

In [19], Yaskorska, Budzynska and Kacprzak proposed a protocol, called LND, for verifying during a dialogue whether a propositional formula is a tautology. This protocol is based on Lorenzen's dialogical logic [7, 8, 15]. The aim of this protocol is to use it in a game simulating natural dialogue as an inference scheme validator. Participants of dialogues perform a variety of actions. Some of them can be recognized as justification of player's standpoint. Such an argumentation may refer to argumentation schemes based on propositional tautologies. The LND game tests propositional formulas and thereby decides whether a corresponding inference is correct. The contribution of the present work is to introduce a dialogue system, called PND, in which players are allowed to make formal fallacies, that is, those that use schemes which are not equivalent to a valid formulas of the underlying logic. In our approach, we limit ourselves to propositional calculus and use as a departing point the general framework for dialogues for argumentation proposed by Prakken [13, 14]. We also define rules which determine how LND games can be nested in PND games. The result is a dialogue system in which players can be committed to formal fallacies, and can identify, verify and debug them.

In many of initially studied dialogue systems like for example the one proposed by Hamblin [6] as well as currently defined systems, arguments can be

constructed only in accordance with the assumed logical base. As a result, players do not have the possibilities of making a formal error in contrast to the participants of natural, real-life dialogues. This work complements the gap. The idea of combining Lorenzen's games with natural persuasion dialogues was also studied by Walton and Krabbe [17]. They proposed Complex Persuasion Dialogue (CPD) which embeds Lorenzen-like dialogue into Hamblin-like dialogue. However, their motivation was to allow players to help their opponents to infer conclusions logically following from their commitments rather then identify formal fallacies.

The study of the argumentation dialogues is of particular interest in areas such as artificial intelligence and multi-agent systems [1, 11]. The research program that combines dialogue theory and argumentation theory with the new four-valued approach to modeling multi-agent inter-actions is guided by a Polish group of scientists. Their discussion on an implementation of four speech acts: assert, concede, request and challenge in a paraconsistent framework is presented in [4]. In the paper [3], they show how speech acts and agents' reasoning rules naturally combine in the framework of 4QL [16], leading to intuitive conclusions while maintaining tractability. Thereby they justify that this four-valued approach can be applied to modelling complex dialogues and argumentations between agents, reasoning in uncertain and dynamic environments. The semantics of speech acts which are applied in deliberation dialogues and thereby are used for modelling communication in teamwork is studied in [5].

The rest of the article is organized as follows. Section 2 presents Prakken's general framework for argumentation dialogue games. In Sect. 3 the extension of this framework, called PND, is introduced. It allows for modelling dialogues in which inference rules used by players are publicly declared and can be challenged. In Sect. 4 the LND system for testing propositional tautologies during a dialogue is described. In Sect. 5 the rules for embedding LND into PND are defined. Finally, in Sect. 6 and 7 the implementation of our protocols and conclusion remarks, respectively, are discussed. The Appendix provides locution, protocol and effect rules for LND games.

2 Formal Framework for Dialogue Games

In [13], Prakken proposes a dialogue game which defines principles of argumentation dialogues, i.e., rules governing the meaning and the use of speech acts. In this system, dialogue utterances are treated as moves in a game and rules of their appropriateness are formulated as rules of the game. Prakken's system provides a basis for our new argumentation dialogue system. The reason why we choose this system from other dialog systems [10, 12, 17] is that it covers a class of argumentation dialogues rather then one selected kind of a dialogue. Moreover this system is flexible with respect to the applied underlying logics, alternative sets of locutions and more or less strict locution rules. Thereby, it offers a nice basis for further research and extensions.

Below, the main terms and definitions of a formal framework of dialogue games for argumentation, introduced by Prakken in [13], are quoted. In the next section, the modification of this system is shown.

All dialogues of the system are assumed to be for two parties arguing about a single dialogue topic t, the *proponent* P who defeats t and the *opponent* O who challenges t. Both proponent and opponent are equipped with a set of commitments that are understood as publicly incurred standpoints. Commitments are expected to be defended upon a challenge.

Definition 1. *A* dialogue system *for argumentation is a pair* $(\mathcal{L}, \mathcal{D})$*, where* $\mathcal{L}$ *is a logic for argumentation and* $\mathcal{D}$ *is a dialogue system proper.*

The elements of the above top level definition are in turn defined as follows.

Definition 2. *A* logic for argumentation $\mathcal{L}$ *is a tuple* $(L_t, R, Args, \rightarrow)$*, where* L_t *is a logical language called the topic language,* R *is a set of inference rules over* L_t*,* $Args$ *is a set of arguments, and* $\rightarrow$ *is a binary relation of defeat defined on* $Args$*.*

For any argument $A \in Args$, $prem(A)$ is the set of premises of A and $conc(A)$ is the conclusion of A.

Definition 3. *An argumentation theory* T_F *within* $\mathcal{L}$ *(where* $F \subset L_t$*) is a pair* $(A, \rightarrow_{/A})$ *where* A *consists of all arguments in Args with only premises and conclusions from* F *and* $\rightarrow_{/A}$ *is* $\rightarrow$ *restricted to* $A \times A$*.* T_F *is called finitary if none of its arguments has an infinite number of defeaters.*

The idea of an argumentation theory is that it contains all arguments that are constructible on the basis of a certain theory.

Definition 4. *A dialogue system proper is a triple* $\mathcal{D} = (L_c, Pr, C)$ *where* L_c *is a communication language,* Pr *is a protocol for* L_c*, and* C *is a set of effect rules of locutions in* L_c*.*

Below the elements of a dialogue system proper are specified.

Definition 5. *A communicating language* L_c *is a set of locutions.*

The most frequently considered locutions are: $claim(\alpha)$ – the speaker asserts that α is the case, $why(\alpha)$ – the speaker challenges α and asks for reasons why it would be the case, $concede(\alpha)$ – the speaker admits that α is the case, $retract(\alpha)$ – the speaker declares that he is not committed (any more) to α, $argue(A)$ – the speaker provides an argument A, where $\alpha \in L_t$ and $A \in Args$.

The protocol for L_c is defined in terms of the notion of a dialogue, which in turn is defined with the notion of a move. The set M of *moves* is defined as $\mathbb{N} \times \{P, O\} \times L_c \times \mathbb{N}$, where the four elements of a move m are respectively denoted by: $id(m)$ - the *identifier* of the move, $pl(m)$ - the *player* of the move, $s(m)$ - the *speech act* (locution) performed in m, $t(m)$ - the *target* of m.

The set of *dialogues*, denoted by $M^{\leq\infty}$, is the set of all sequences $m_1, \ldots, m_i, \ldots$ from M such that each i^{th} element in the sequence has identifier i, $t(m_1) = 0$, for all $i > 1$ it holds that $t(m_i) = j$ for some m_j preceding m_i in the sequence. The set of *finite dialogues*, denoted by $M^{<\infty}$, is the set of all finite sequences that satisfy these conditions. When d is a dialogue and m a move, then (d, m) will denote the continuation of d with m.

A protocol also assumes a turntaking rule. A *turntaking function* T is a function $T : M^{<\infty} \to 2^{\{P,O\}}$ such that $T(\emptyset) = \{P\}$. A *turn* of a dialogue is a maximal sequence of stages in the dialogue where the same player moves. This definition allows that more than one speaker has the right to speak next.

The key notion for the dialogue system is the protocol.

Definition 6. *A protocol on the set of moves M is a set $Pr \subseteq M^{<\infty}$ satisfying the condition that whenever d is in Pr, so are all initial sequences that d starts with.*

A partial function $\overline{Pr} : M^{<\infty} \to 2^M$ is derived from Pr as follows: $\overline{Pr}(d) =$ undefined whenever $d \notin Pr$; $\overline{Pr}(d) = \{m : (d, m) \in Pr\}$ otherwise. The elements of the domain $dom(\overline{Pr})$ are called the *legal finite dialogues*. The elements of $\overline{Pr}(d)$ are called the moves allowed after d. If d is a legal dialogue and $\overline{Pr}(d) = \emptyset$, then d is said to be a *terminated* dialogue.

All protocols of Prakken's system are assumed to satisfy the following conditions for all moves m and all legal finite dialogues d. If $m \in \overline{Pr}(d)$, then:

PP1 $pl(m) \in T(d)$,
PP2 If $d \neq d_0$ and $m \neq m_1$, then $s(m)$ is a reply to $s(t(m))$ according to L_c,
PP3 If m replies to m', then $pl(m) \neq pl(m')$,
PP4 If there is an m' in d such that $t(m) = t(m')$, then $s(m) \neq s(m')$,
PP5 For any $m' \in d$ that surrenders to $t(m)$, m' is not an attacking counterpart of m.

Rule **PP1** says that a move is legal only if moved by the player-to-move. **PP2** says that a replying move must be a reply to its target according to L_c. **PP3** says that one cannot reply to one's own moves. Rule **PP4** states that if the player backtracks, the new move must be different from the first one. Finally, **PP5** says that surrenders should not be 'revoked'.

Every utterance from L_c can influence participants' commitments. Results of utterances are determined by commitment rules which are specified as a commitment function.

Definition 7. *A commitment function is a function: $C : M^{<\infty} \times \{P, O\} \to 2^{L_t}$, such that $C(\emptyset, i) = \emptyset$ for $i \in \{P, O\}$.*

$C(d, i)$, for a participant $i \in \{P, O\}$ and a stage of a dialogue $d \in M^{<\infty}$, denotes a player i's commitments at the stage of a dialogue d.

3 Prakken Natural Dialogue

The framework for dialogue games proposed by Prakken implements the intention of all argumentation dialogue games, that is, to define rules that allow participants to play the dialogue in a way that could lead to an agreement. In a persuasion dialogue game it is understood as an opportunity to convince the opponent to change its position, and consequently resolve a conflict of opinion. Therefore, much attention has been devoted to establishing conditions under which such an agreement can be achieved. In Prakken's system it is assumed that all participants agree to the topic language and the set of rules under which valid arguments are defined. What is more, all the rules are correct in the assumed logic. Unquestionably, this is the basis for the agreement. Observe, however, that participants of real life dialogues very rarely determine among themselves their knowledge base or rules they use. Moreover, they are often committed to wrong inferences. The case when the participants apply different, not necessarily correct, inference schemes cannot be modelled in Prakken's system. This is why we propose some modifications of his system. In Prakken's general framework, players can argue using one of the possible arguments. All the arguments are constructed over inference rules of the assumed logical system. Thereby they are correct. Our proposition is to allow players to perform locutions in which incorrect argumentation is provided.

The new dialogue system, is called Prakken Natural Dialogue (PND) and is a pair $(\mathcal{L}, \mathcal{D})$. A logic for argumentation $\mathcal{L}$ is a tuple $(L_t, R, Args, \rightarrow)$ where L_t is a propositional logic and R is as set of inference rules of L_t. To realize our goal, we need to distinguish in the topic language two sentences: (a) "The formula θ is a propositional tautology" and (b) "The formula θ is obtained from the formula ψ by some substitution". For convenience, we introduce the following abbreviations. Let $Taut(\theta)$ will be short for "θ is a propositional tautology". This sentence should be true or false. We do not state here that actually θ is a tautology. Moreover, if $\theta(q_1, \ldots, q_n)$ is a propositional formula build under propositions $q_1, \ldots, q_n$ and $\alpha_1, \ldots, \alpha_n$ are propositional formulas, we write $\theta(\alpha_1/q_1, \ldots, \alpha_n/q_n)$ for a formula θ in which the proposition q_i is replaced with the formula α_i for $i = 1, \ldots, n$. It is obvious that if $Taut(\theta(q_1, \ldots, q_n))$ is true (i.e. $\theta(q_1, \ldots, q_n)$ is a tautology), then $Taut(\theta(\alpha_1/q_1, \ldots, \alpha_n/q_n))$ is also true (i.e. $\theta(\alpha_1/q_1, \ldots, \alpha_n/q_n)$ is a tautology too). We will also write $\theta(\alpha_1(p_1, \ldots, p_k)/q_1, \ldots, \alpha_n(p_1, \ldots, p_k)/q_n) = \psi(p_1, \ldots, p_k)$ if the formula ψ is obtained from θ by substitution q_i for α_i $(i = 1, \ldots, n)$.

The set of arguments $Args$ is a set of pairs $A = (Prem(A), Conc(A))$ where $prem(A)$ is a set of premises (a finite set of propositional formulas) and $conc(A)$ is a conclusion (a propositional formula) such that the formula $\bigwedge_{a \in prem(A)} \Rightarrow conc(A)$ is a propositional tautology. Since in this paper we do not focus on attacks and counterattacks on arguments, we omit here the specification of the defeat relation.

A dialogue system proper for PND, $\mathcal{D} = (L_c, P, C_P, C_O)$ is defined by locution, protocol, and effect rules presented in the next subsections. Taking into account the structural properties [9], the protocol for PND is:

- *unique-move*, i.e., the turn switches after each move,
- *multi-reply*, i.e., players can return to earlier choices and try alternative moves to the other player's moves,
- *immediate-reply*, i.e., each player must immediately respond to the move of the other player.

3.1 Locution rules

The communication language of PND assumes the following locutions:

PL1 Claim $claim(\varphi)$ is performed when a player asserts that sentence φ is true and his antagonist does not have this sentence in his commitment base.
PL2 Concession $concede(\varphi)$ is performed when a player asserts that sentence φ is true and his antagonist has this sentence in his commitment base.
PL3 Challenge $why(\varphi)$ is performed when a player asks about a proof for φ.
PL4 Argumentation (φ) *since* $(\psi_1, \ldots, \psi_n, Taut(\theta))$ is performed when a player justifies statement φ with a set of premises $\psi_1, \ldots, \psi_n$ and the inference rule corresponding to the formula θ. A player can use in this locution a rule which is not correct and does not correspond to a tautology.
PL5 Retraction $retract(\varphi)$ is performed when a player resigns from the statement that sentence φ is true.

3.2 Protocol rules

The protocol for PND satisfies the protocol rules **PP1-PP5** of Prakken's general framework and adds the following, where $s \in \{P, O\}$, $d \in Pr$, $m \in \overline{Pr}(d)$, $\varphi, \psi_1, \ldots, \psi_n, \Theta \in L_t$:

PP6 if $d = \emptyset$, then $s(m)$ is of the form
  (a) $claim(\varphi)$ or
  (b) (φ) *since* $(\psi_1, \ldots, \psi_n, Taut(\theta))$,
PP7 if m concedes the conclusion of an argument moved in m', then m' does not reply to a *why* move,
PP8 if $s(m)$ is $claim(\varphi)$, then $s(m')$ for $m' \in \overline{Pr}((d, m))$ is of the form
  (a) $why(\varphi)$ (attack) or
  (b) $concede(\varphi)$ (surrender),
PP9 if $s(m)$ is $why(\varphi)$, then $s(m')$ for $m' \in \overline{Pr}((d, m))$ is of the form
  (a) (φ) *since* $(\psi_1, \ldots, \psi_n, Taut(\psi_1 \wedge \ldots \wedge \psi_n \Rightarrow \varphi))$ (attack) or
  (b) (φ) *since* $(Taut(\theta(q_1, \ldots, q_n)), Taut(\theta(\alpha_1/q_1, \ldots, \alpha_n/q_n)) = \beta)$ for some formulas $\alpha_1, \ldots, \alpha_n$ if $\varphi = Taut(\beta)$ (attack) or
  (c) $retract(\varphi)$ (surrender),
PP10 if $s(m)$ is (φ) *since* $(\psi_1, \ldots, \psi_n, Taut(\theta))$, then $s(m')$ for $m' \in \overline{Pr}((d, m))$ is of the form
  (a) $why(\alpha)$ where $\alpha \in \{\psi_1, \ldots, \psi_n, Taut(\theta)\}$ (attack) or
  (b) $(\neg\varphi)$ *since* $(\beta_1, \ldots, \beta_n, Taut(\beta_1 \wedge \ldots \wedge \beta_n \Rightarrow \neg\varphi))$
  (c) $concede(\alpha)$ where $\alpha \in \{\varphi, \psi_1, \ldots, \psi_n, Taut(\theta)\}$ (surrender),

PP11 if $s(m)$ is $concede(\varphi)$ or $retract(\varphi)$, then $s(m')$ for $m' \in \overline{Pr}((d,m))$ is
- **(a)** a reply (attack or surrender) to some earlier move of the other player or
- **(b)** $\overline{Pr}((d,m)) = \emptyset$.

Rules **PP6** and **PP7** are inspired by liberal dialogues (see [13]). **PP6** says that each dialogue begins with either a claim or an argument. The initial claim or, if a dialogue starts with an argument, its conclusion is the topic of the dialogue. **PP7** restricts concessions of an argument conclusion to conclusions of counterarguments. Rules **PP8**-**PP11** describe possible moves after specific locution. Observe that every move replies to some earlier move of the antagonist and it is either attack or surrender.

3.3 Effect rules

In PND there are two participants. Therefore, we need to define a commitment functions for both of them:

$$C_s : M^{<\infty} \times \{P, O\} \to 2^{L_t}$$

where $s \in \{P, O\}$. However, locution rules do not depend on the role which the performer of the locution plays. Effects on the commitment sets after execution specific moves are described below, where s denotes the speaker, $(m_0, \ldots, m_n)$ is a legal dialogue, and $\varphi, \psi_1, \ldots, \psi_n, Taut(\Theta)) \in L_t$.

PE1 if $s(m_n) = claim(\varphi)$, then $C_s(m_0, m_1, \ldots, m_n) = C_s(m_0, m_1, \ldots, m_{n-1}) \cup \{\varphi\}$, i.e. after $claim(\varphi)$ the formula φ is added to the s's commitment set,

PE2 if $s(m_n) = concede(\varphi)$, then $C_s(m_0, \ldots, m_n) = C_s(m_0, \ldots, m_{n-1}) \cup \{\varphi\}$, i.e., after $concede(\varphi)$ the formula φ is added to the s's commitment set,

PE3 if $s(m_n) = why(\varphi)$, then $C_s(m_0, m_1, \ldots, m_n) = C_s(m_0, m_1, \ldots, m_{n-1})$, i.e., after $why(\varphi)$ the s's commitment set does not change,

PE4 if $s(m_n) = (\varphi)$ *since* $(\psi_1, \ldots, \psi_n, Taut(\theta))$, then $C_s(m_0, m_1, \ldots, m_n) = C_s(m_0, m_1, \ldots, m_{n-1}) \cup \{\varphi, \psi_1, \ldots, \psi_n, Taut(\theta)\}$, i.e., after this locution the formulas $\varphi, \psi_1, \ldots, \psi_n, Taut(\theta)$ are added to s's commitment set,

PE5 if $s(m_n) = retract(\varphi)$, then $C_s(m_0, m_1, \ldots, m_n) = C_s(m_0, m_1, \ldots, m_{n-1}) \backslash \{\varphi\}$, i.e., after $retract(\varphi)$ the formula φ is deleted from the s's commitment set.

3.4 Turntaking

In a PND game, P makes the first move, then O and P take turns in performing moves. Thus the turntaking function is defined as follows:

$$T(m_0, m_1, \ldots, m_n) = \begin{cases} P \text{ iff } n \text{ is even} \\ O \text{ iff } n \text{ is odd} \end{cases}.$$

4 Lorenzen Natural Dialogue

In [18], Yaskorska, Budzynska, and Kacprzak proposed a dialogue system, LND, that allows communicating agents to prove that a formula used in an argument is a *classical propositional tautology*, and, as a result, to identify and eliminate *classical propositional formal fallacies* committed during a natural dialogue. This is achieved through a combination of a system for representing natural dialogues with a system for representing formal dialogues. In the first case, the framework proposed by Prakken [14] was used, since it provides a generic and formal specification of the main elements of dialogue systems for persuasion. For handling formal fallacies in a dialogue, the dialogical logic introduced by Lorenzen [7, 8, 15] was applied. Lorenzen's dialogue games allow the players to prove that a formula is a tautology of classical propositional logic, if the proponent has a winning strategy in a given game. The aim of this system is not to jointly built an argument: φ, therefore ψ, as in inquiry dialogues (see e.g. [2]), but to allow the participants to play against each other starting with opposing viewpoints on an argument validity and determining which player wins.

The dialogical logic communication language and structure are different from systems for natural dialogues. For example, in Lorenzen's system the only moves available to speakers are: X *attacks* φ and X *defends* φ, while, according to Prakken's specification, in systems for natural dialogues the legal locutions can be: *claim* φ, *why* φ, *concede* φ, *retract* φ, φ *since* S, *question* φ. Therefore the main challenge was to introduce a new description of the dialogical logic which meets the requirements of Prakken's generic specification. The correspondence result between the original and the new version of the dialogical logic is presented in [19] where it is proved that a winning strategy for a proponent in the original version of the dialogical logic means a winning strategy for a proponent in the new one, and conversely.

The locution, protocol and effect rules of the LND system are presented in the Appendix.

5 Embedding Dialogues

The aim of dialogical logic introduced by Lorenzen is to define logical connectives in terms of attacks and defences, and then determined whether the formula under discussion is valid in the given logical system. The goal of argumentation dialogue systems is to define rules of the game in which participants can challenge and provide reasons for their claims and positions. Our intention is to combine these two approaches. This object is achieved by proposing the system LND and the system PND and by defining rules for their embedding. The main advantage of combining the systems LND and PND is to obtain a uniform system that allows modelling of dialogues in which participants can be committed to formal fallacies and may discuss the patterns of inferences, in terms of their correctness, according to a given logical framework. In this work it is propositional logic.

In the new dialogue system which is a combination of LND with PND, we take the following assumptions:

- in the PND (LND) part of the game, players use the same topic and communication languages,
- players have different commitment sets in PND game and hypothetical commitment sets in LND game,
- players can use correct and incorrect inference rules and correct and incorrect arguments constructed under these rules,
- players can challenge claims and inference rules of the other player,
- a correct rule is a rule which corresponds to some propositional tautology,
- correctness of inference rules is examining during Lorenzen's game, i.e., if a player challenges some rule, its antagonist starts Lorenzen's game and takes the role of the proponent,
- if the proponent of Lorenzen's game looses, then he must retract from the commitment which says that the inference rule under discussion is correct.

Below the rules for embedding LND into PND are given. Two new locutions are introduced: *InitLor* and *EndLor*.

PL6 Initialization The locution $InitLor(\theta)$ breaks the natural dialogue and initializes the DL-like dialogue for formula θ. The player who performed $InitLor(\theta)$ becomes the proponent for θ in the embedded DL-like dialogue.

PL7 Ending The locution $EndLor(\theta)$ ends the DL-like dialogue for θ and resumes the broken natural dialogue.

In the approach it is assumed that DL-like dialogue for a formula θ starts when one of the players challenges this formula. Then, the players examine θ in accordance with the rules of DL-like games. Protocol rules for embedding a formal dialogue into a natural one is described in **PP12** - **PP15**.

PP12 The locution $InitLor(\theta)$ can be performed as a reply to the locution $why(Taut(\theta))$ or the locution $claim(\neg Taut(\theta))$ executed in a PND game.

PP13 After the locution $InitLor(\theta)$ players can perform the same actions which are allowed to execute after $claim(\theta)$ according to the protocol rules **P1-P8** of the system LND (see Appendix).

PP14 The locution $EndLor(\theta)$ can be performed by a player X if X has no legal move according to the protocol rules **P1-P8** of the system LND (see Appendix).

PP15 After the locution $EndLor(\theta)$, (1) if P is the performer, then P executes $retract(Taut(\theta))$ in the broken PND game, (2) if O is the performer, then O executes $concede(Taut(\theta))$ in the broken PND game.

6 Implementation

The implementation consists of two applications: LNDGame and PNDGame. The program LNDGame is intended to implement the dialogue construction based on the LND game. All the basic notions are modelled in the program in a direct way. According to the adopted formal definitions, a dialogue $D(\theta)$, for

a formula θ, is a set of dialogue games consisting of sequences of moves. The initial move is performed by the proponent, which claims formula θ. During the dialogue game, each participant makes moves, one after the other, according to the rules of the protocol.

In the program, a move is defined by locution type and the formula, it also has a reference to the locution it responds to. A formula is represented as a tree of subformulas, even though for the user it appears rather as a sequence of symbols. Initial formula proposed by the proponent can be given in two ways: by providing a sequence of symbols or by recursively constructing subformulas using GUI. A hypothetical commitment set is associated with each participant. It is an increasing set of formulas previously committed.

The application enables two modes. In the first, interactive mode, the user has a possibility to choose each move on the behalf of one of the participants. The move can be chosen from the list of possible moves updated after each move. The result is a sequence of moves chosen by a participant at each step of one of the dialogue games, and the result of the game, i.e., whether the proponent wins the game or not. In the second, auto mode, program can scan all possible dialogue games for the specified initial formula θ. The result contains a sequences of moves, one for each theoretically possible dialogue game, with information about the other available moves at each step. If a proponent wins all the possible dialogue games, it wins the dialogue $D(\theta)$ (in this case formula θ is a tautology). Certainly, we have to take into account the complexity of such a scan for more complex formulas. The current version is adapted to handle real-life size formulas and illustrates that even for small formulas used in short dialogues $D(\theta)$ can be very large.

The second application, PNDGame, is intended to implement the dialogue games based on PND. During the dialogue game, understood in the same way as above, each of the participants can use different inference schemes, some of them can be even incorrect. Each participant can also challenge another participant's rule of inference using Lorenzen-style Natural Dialogue (and the same mechanism as proposed in LNDGame application). To prove that the rule is incorrect, challenging participant has to start a Lorenzen game and to win it. Then, the proponent of the rule has to withdraw this rule from his set of inference rules.

The implementation was made in Java language, which will facilitate further development of application, and also software portability. This choice helps us to avoid from the restrictions of other protocols than analyzed in this paper. The participants of the dialog, as well as game manager, are implemented as a separated, eventually distributed over the network classes. The aim of this implementation is dialogue game simulation and a decision making support during such a game. It allows to verify the validity of formulas used in dialogues as tautologies and to identify formal fallacies. It also enables recording of conducted dialogues games, which makes later analysis possible. Subsequent versions of the application prepared by the authors will correspond to the efforts to combine several formal systems for modelling natural dialogues in terms of games and analyzing properties of such dialogue games.

7 Conclusions

This work provides a unified dialogue system for argumentation which combines two approaches: Lorenzen's dialogical logic (DL) with a modified Prakken's framework for dialogue games for argumentation. The idea of dialogical logic was applied in Lorenzen Natural Dialogue (LND) where the structural and particle rules of DL were reconstructed and defined in terms of locution, commitment and protocol rules of dialogue games [19]. Prakken's system was extended with specific locutions which allows players to use incorrect arguments, to show directly the inferences on which these arguments are based and to challenge them. The result is a Prakken Natural Dialogue. Finally, the rules for embedding LND into PND are defined.

The main advantage of the new system is that in the course of a dialogue the participants can verify their sets of rules and create new arguments. Thereby, this idea allows a study argumentation systems in which participants have the ability to learn. The dynamic nature of dialogues and frequent change of information may be reflected not only in revising beliefs and commitments of players but also in changing the way in which they argue and reason.

The proposed system can be used both as a simulation of natural dialogues conducted in artificial intelligence systems, and as a tool for argumentation and persuasion communication in multi-agent systems.

References

1. Amgoud, L., Maudet, N., Parsons, S.: Modelling dialogues using argumentation. In: Proc. of the Fourth Int. Conf. on Mulit-Agent Systems (2000) 31–38
2. Black, E., Hunter, A.: An inquiry dialogue system. Autonomous Agent Multi-Agent Systems **18** (2009) 173–209
3. Dunin-Kęplicz, B., Strachocka, A., Szałas, A., Verbrugge, R.: A paraconsistent approach to speech acts. In: Proc. of ArgMAS (2012) 59–78
4. Dunin-Kęplicz, B., Strachocka, A., Szałas, A., Verbrugge, R.: Perceiving speech acts under incomplete and inconsistent information. Frontiers in Artificial Intelligence and Applications **252** (2013) 255 – 264
5. Dunin-Kęplicz, B., Strachocka, A., Verbrugge, R.: Deliberation dialogues during multiagent planning. LNCS **6804** (2011) 170–181
6. Hamblin, C.: Fallacies. Methuen, London (1970)
7. Keiff, L.: Dialogical logic. The Stanford Encyclopedia of Philosophy (2011)
8. Lorenz, K., Lorenzen, P.: Dialogische logik. WBG. Darmstadt (1978)
9. Loui, R.: Process and policy: resource-bounded non-demonstrative reasoning. Computational Intelligence **14** (1998) 1–38
10. Mackenzie, J.D.: Question begging in non-cumulative systems. Journal of Philosophical Logic **8** (1979) 117–133
11. McBurney, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language and Information **13** (2002) 315–343
12. Parsons, S., Wooldridge, M., Amgoud, L.: Properties ans complexity of some formal inter-agent dialogues. Journal of Logic and Computation **13** (2003) 347–376

13. Prakken, H.: Coherence and flexibility in dialogue games for argumentation. Journal of Logic and Computation **15** (2005) 1009–1040
14. Prakken, H.: Formal systems for persuasion dialogue. The Knowledge Engineering Review **21** (2006) 163–188
15. Rahman, S., Tulenheimo, T.: From games to dialogues and back: towards a general frame for validity. Games: Unifying Logic, Language, and Philosophy (2006)
16. Szałas, A.: How an agent might think. Logic J. IGPL **21(3)** (2013) 515–535
17. Walton, D.N., Krabbe, E.C.W.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. State University of N.Y. Press (1995)
18. Yaskorska, O., Budzynska, K., Kacprzak, M.: Rules for formal and natural dialogues in agent communication. In: Proc. of CS&P (2012) 416-427
19. Yaskorska, O., Budzynska, K., Kacprzak, M.: Proving propositional tautologies in a natural dialogue. Fundamenta Informaticae (2013)

Appendix

In LND game the *set of players* consists of two elements $\{\mathbf{O}, \mathbf{P}\}$. *Topic language* L_t is assumed to be that of classical propositional logic. The *dialogue system proper* is specified by the locution, protocol and effect rules.

Locution rules. The locution rules for LND are specified as follows: [**L1**] **Claim** *claim* φ is performed when a player: (1) attacks $\neg A$, then φ is a formula A, (2) defends $A \wedge B$, then φ is a formula A or a formula B, (3) attacks $A \rightarrow B$, then φ is a formula A, (4) defends $A \rightarrow B$, then φ is a formula B; [**L2**] **Concession** *concede* φ can be performed only by a proponent **P**, and this locution is performed when φ is an atomic formula and the performer: (1) attacks $\neg A$, then φ is a formula A, (2) defends $A \wedge B$, then φ is a formula A or a formula B, (3) attacks $A \rightarrow B$, then φ is a formula A, (4) defends $A \rightarrow B$, then φ is a formula B; [**L3**] **Argumentation** φ *since* ψ is performed when a player defends $A \vee B$, then φ is a formula $A \vee B$ and ψ is a set which includes the formula A or the formula B; [**L4**] **Challenge** The challenge *why* φ is performed when a player attacks $A \vee B$, then φ is a formula $A \vee B$; [**L5**] **Question** The question *question* φ is performed when a player attacks $A \wedge B$, then φ is a formula A or a formula B.

Protocol rules. The LND protocol descries a formal dialogue game $\triangle = m_0, \ldots, m_n$ on a topic A, which is called a DL-like game. Let **D'**(A) be DL-like dialogue for A, i.e. a set of DL-like games for A. The protocol is specified as follows: [**P1**] In the first move **P** performs *claim* φ where φ is the topic A; next players perform one locution at each turn; [**P2**] A player **P** cannot perform *claim* φ where φ is a proposition; he can state that φ is true executing *concede* φ but this move can be performed only if **O** claimed φ in some previous move; [**P3**] After *claim* φ a player can perform: (1)c*claim* ψ, if (a) φ is a negation of the formula and ψ is a contradiction to φ, (b) φ is the implication and ψ is the antecedent of φ, (c) φ is the antecedent of an implication under the attack and ψ is the consequent of this implication (in P3.1, P has to follow the restriction described in P2), (2) *concede* ψ, if **P** is the player and ψ is a proposition, and (a) φ is a negation of the formula and ψ is a contradiction to φ, (b) if φ is the implication under the attack and ψ is its consequent, (3) *question* ψ, if φ is a conjunction and ψ is one of its operands, (4) *why* φ, if φ is a disjunction, (5) attack or defence of any formula uttered before, if **P** is the player, (6) no move, if (a) *claim* φ is an attack on negation and φ is a proposition, (b) *claim* φ is a defence executed by **P**, and **O** has attacked this defence before; [**P4**] After *concede* φ performed by **P**, where φ is a proposition, **O** has no move; [**P5**] After φ *since* Ψ, where $\Psi = \{\psi\}$ the player can

perform: (1) *claim* φ, if (a) ψ is a negation of the formula and φ is a contradiction to ψ, (b) if ψ is the implication φ is its antecedent (in P5.1, P has to follow the restriction described in P2), (2) *concede* φ, if **P** is the player and φ is a proposition, and (a) ψ is a negation of the formula and φ is a contradiction to ψ, (b) if ψ is the implication under the attack and φ is its consequent, (3) *question* φ, if ψ is a conjunction and φ is one of its operands, (4) *why* ψ, if ψ is a disjunction, (5) attack or defence of any formula uttered before, if **P** is the player, (6) no move, if φ *since* Ψ is a defence executed by **P**, and **O** has attacked this defence before; [**P6**] After *why* φ a player can perform: (1)cφ *since* ψ (**P** has to follow the restriction described in **P2**), (2) attack or defence of any formula uttered before, if **P** is the player; [**P7**] After *question* φ a player can perform: (1)c*claim* φ (**P** has to follow the restriction described in **P2**), (2) *concede* φ, if **P** is the player and φ is a proposition, (3) attack or defence of any formula uttered before, if **P** is the player; [**P8**] If **O** loses a game $\triangle$ which involves the *propositional choice* made by **O** (see DL-rule **SR-2**), then **O** can start a sub-game $\triangle\prime$. There are three types of sub-games $\triangle\prime$ possible: **(I)** Assume that **P** executes *claim* φ in $\triangle$, where φ is $\psi \wedge \psi\prime$, and **O** attacks the conjunction by stating: *question* ψ (the *propositional choice* step). If they continue to play the game $\triangle$ according to the LND rules and **P** makes the last available move, then **O** can extend $\triangle$ with a sub-game $\triangle\prime$ by attacking the conjunction one more time using the locution: *question* $\psi\prime$. **(II)** Assume that **O** executes *claim* φ in $\triangle$, where φ is $\psi \vee \psi\prime$. In the next moves, **P** attacks the disjunction by stating: *why* φ, and **O** defends it by stating: φ *since* ψ (the *propositional choice* step). If they continue to play the game $\triangle$ according to the LND rules and **P** makes the last available move, then **O** can extend $\triangle$ with a sub-game $\triangle\prime$ by defending the disjunction one more time with the locution: φ *since* $\psi\prime$. **(III)** Assume that in a game $\triangle$, **O** executes *claim* φ, where φ is $\psi \rightarrow \psi\prime$, and **P** attacks the implication by stating: *claim* ψ. There are two possible sub-cases: (1) Let **O** respond to this attack by defending the implication, i.e., he performs: *claim* $\psi\prime$ (the *propositional choice* step). If they continue to play the game $\triangle$ according to the LND rules and **P** makes the last available move, then **O** can extend $\triangle$ with a sub-game $\triangle\prime$ by responding to **P**'s attack one more time and attacking the propositional content of **P**'s attack, ψ, accordingly to its logical form. (2) Let **O** respond to **P**'s attack by attacking its content, ψ, accordingly to its logical form (the *propositional choice* step). If they continue to play the game $\triangle$ according to the LND rules and **P** makes the last available move, **O** can extend $\triangle$ with a sub-game $\triangle\prime$ by responding to **P**'s attack one more time and defend the implication using the locution: *claim* $\psi\prime$. In all cases **P8.I-P8.III**, during $\triangle\prime$ the players may use all the LND rules with a limitation on the **P2** rule such that **P** cannot perform *concede* ϕ if **O** did not introduce a proposition ϕ in $\triangle$ before the propositional choice step and did not introduce a proposition ϕ in $\triangle\prime$.

Effect rules The dynamics of participants' commitments in LND formal games is showed by a **hypothetical commitment base**. During the game, new formulas are added to this base and no formulas are deleted. For a formal game $\triangle = m_0, \ldots, m_n \in$ **D'**(A), the rules for hypothetical commitment base C'_s of a player $s \in \{\mathbf{O}, \mathbf{P}\}$ are specified below, where $s(m)$ denotes a move of a player s and $\varphi, \psi \in L_t$ are propositional formulas: [**E1**] if $s(m_n) = claim(\varphi)$, then $C'_s(m_0, \ldots, m_n) = C'_s(m_0, \ldots, m_{n-1}) \cup \{\varphi\}$, i.e. after $claim(\varphi)$ the formula φ is added to the hypothetical commitment base, [**E2**] if $s(m_n) = why(\varphi)$, then $C'_s(m_0, \ldots, m_n) = C'_s(m_0, \ldots, m_{n-1})$, [**E3**] if $s(m_n) = concede(\varphi)$, then $C'_s(m_0, \ldots, m_n) = C'_s(m_0, \ldots, m_{n-1}) \cup \{\varphi\}$, [**E4**] if $s(m_n) = (\varphi \vee \psi)$ *since* φ, then $C'_s(m_0, \ldots, m_n) = C'_s(m_0, \ldots, m_{n-1}) \cup \{\varphi\}$, i.e. after $(\varphi \vee \psi)$ *since* φ the formula φ is added to s's hypothetical commitment base, [**E5**] if $s(m_n) = question(\varphi)$, then $C'_s(m_0, \ldots, m_n) = C'_s(m_0, \ldots, m_{n-1})$.

Discovery of Cancellation Regions within Process Mining Techniques*

A. A. Kalenkova and I. A. Lomazova

[1] National Research University Higher School of Economics, Moscow, Russia
{akalenkova, ilomazova}@hse.ru
[2] Program Systems Institute of the Russian Academy of Sciences,
Pereslavl-Zalessky, Russia

Abstract. Process mining is a relatively new field of computer science which deals with process discovery and analysis based on event logs. In this work we consider the problem of discovering workflow nets with cancellation regions from event logs. Cancellations occur in the majority of real-life event logs. In spite of huge amount of process mining techniques little has been done on cancellation regions discovery. We show that the state-based region algorithm gives labeled Petri nets with overcomplicated control flow structure for logs with cancellations. We propose a novel method to discover cancellation regions from the transition systems built on event logs and show the way to construct equivalent workflow net with reset arcs to simplify the control flow structure.

1 Introduction

Process mining technology [1] provides us with a variety of methods for discovering business processes from event logs. These methods are commonly used when formal process description is not available or description does not correspond to the real-life process behavior. One of the goals of the process discovery is the retrieving of readable process models. The majority of real-life event logs contain information about cancellations which occur during the process execution. These cancellations can be expressed by means of workflow languages (BPMN [2], YAWL [3]) and formal models such as Reset workflow nets (RWF-net) [4]. In [5] an approach for discovery of cancellations from event log has been presented. This approach constructs a workflow net (WF-net) with the state-based region algorithm [6]. After that it replays the log on this model, for all remaining tokens reset arcs are added to the WF-net, as a result RWF-net is produced.

State-based region algorithm [6] used within process mining techniques was developed on the basis of well-known algorithms for the construction of a Petri net (PN) from a transition system (TS) [7–9], taking into account that minor transformations of TS will allow to retrieve WF-net instead of arbitrary PN. An algorithm was given by [7] to synthesize a PN from the elementary transition

* This study was carried out within the National Research University Higher School of Economics' Academic Fund.

system (ETS) in such a manner that reachability graph (RG) of the PN is isomorphic to the TS (or the minimized TS if it is not minimal). Algorithm proposed in [8, 9] generate a labeled PN for an arbitrary TS such that RG of the PN is isomorphic or split-isomorphic [8] to the TS or its minimization. This algorithm effectively checks whether TS is elementary (by verifyng the excitation closure property [8]), if TS is not elementary then TS and target PN are splitted.

In this paper we propose a method which not just adds reset arcs, but makes a target model more compact and readable. We prove that the straightforward applying of a state-based region algorithm to an event logs with cancellations leads to the generation of a labeled Petri net with overcomplicated control flow structure. Then we present an algorithm for discovering cancellations and constructing an RWF-net with a more compact and transparent structure. We prove correctness of the proposed algorithm.

The paper is organized as follows. In Section 2 a motivating example of the booking process with cancellations is presented, it gives a ground for the development of a novel cancellation discovery method. Section 3 contains some basic definitions and notions, including logs, Petri nets and transition systems. In Section 4 we formally prove that cancellations in a log in the presence of parallel branches lead to the generation of labeled Petri nets with complicate control-flow structure. We also present an algorithm for construction of a RWF-net from TS and give the proof of the algorithm correctness. Section 5 contains some conclusions.

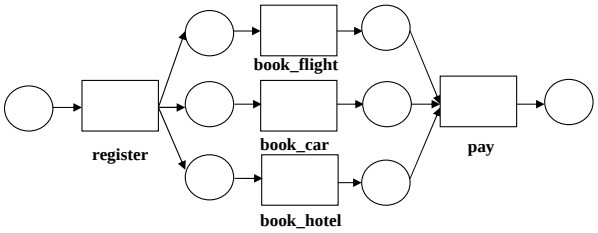


Fig. 1. A WF-net for a regualer booking process (no cancellations)

2 Motivating example

Let us consider a simple model of booking the trips process from [5]. One needs to book a hotel, a car and a flight (Fig. 1). This model formalizes the booking process which can't be canceled, while a real-life process might be canceled in consequence of internal booking errors. If we take a look at a log of some real-life booking process, we may notice that it contains traces of process execution

failures along with traces of standard booking process executions. A sample of such a real-life log L is presented in Fig. 2.

$$
\begin{aligned}
L = \{ & < register, book\_flight, book\_hotel, book\_car, pay >, \\
& < register, book\_flight, book\_car, book\_hotel, pay >, \\
& < register, book\_car, book\_flight, book\_hotel, pay >, \\
& < register, book\_car, book\_hotel, book\_flight, pay >, \\
& < register, book\_hotel, book\_car, book\_flight, pay >, \\
& < register, book\_hotel, book\_flight, book\_car, pay >, \\
& < register, book\_flight, book\_hotel\_NOK, cancel >, \\
& < register, book\_flight, book\_car, book\_hotel\_NOK, cancel >, \\
& < register, book\_car, book\_flight, book\_hotel\_NOK, cancel >, \\
& < register, book\_car, book\_hotel\_NOK, cancel >, \\
& < register, book\_hotel\_NOK, cancel >\}.
\end{aligned}
$$

Fig. 2. An event log for booking process with cancellations

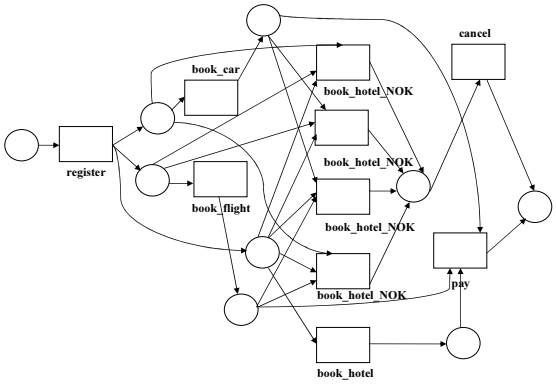


Fig. 3. A WF-net for booking process with cancellations discovered from event log in Fig. 2 by the standard region-based method.

In this log an additional event with the label 'book_hotel_NOK' occurs and denotes the situation when the booking hotel step failures. The 'book_hotel_NOK' event causes cancellation of parallel branches of booking a car and a flight and execution of a catching task (which is represented in the log as an event with the

label 'cancel'). Applying the state-based regions method for process discovery to this sample log gives us a labeled WF-net with a complicated control-flow structure (Fig. 3). This model is rather confounded. The clear structure of the core process (Fig. 1) can be hardly recognized. So, it is very important to find a method for discovering clear and readable workflow net with cancellations.

3 Logs, Petri nets and Transition systems. Definitions

Let S be a finite set. A *multiset* m over a set S is a mapping $m : S \to Nat$, where Nat is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

For two multisets m, m' we write $m \subseteq m'$ iff $\forall s \in S : m(s) \leq m'(s)$ (the inclusion relation). The sum of two multisets m and m' is defined as usual: $\forall s \in S : (m + m')(s) = m(s) + m'(s)$, the difference is a partial function: $\forall s \in S$ such that $m(s) \geq m(s') : (m - m')(s) = m(s) - m'(s)$. By $\mathcal{M}(S)$ we denote the set of all finite multisets over S. Non-negative integer vectors are often used to encode multisets. Actually, the set of all multisets over finite S is a homomorphic image of $Nat^{|S|}$.

Definition 1 (Event log). *Let A be a set activities. A trace σ can be described as a sequence of activities, i.e., $\sigma \in A^*$. An event log L is a multiset of traces, i.e., $L \in \mathcal{M}(A^*)$.*

Definition 2 (Petri net). *Let P and T be disjoint sets of* places *and* transitions *and $F : (P \times T) \cup (T \times P) \to Nat$. Then $N = (P, T, F)$ is a* Petri net.

Let Σ be a finite alphabet. A labeled PN is a PN with a labeling function $\lambda : T \to \Sigma$ which maps every transition to a symbol (called a label) from Σ.

A *marking* in a Petri net is a function $m : P \to Nat$, mapping each place to some natural number (possibly zero). Thus a marking may be considered as a multiset over the set of places. Pictorially, P-elements are represented by circles, T-elements by boxes, and the flow relation F by directed arcs. Places may carry tokens represented by filled circles. A current marking m is designated by putting $m(p)$ tokens into each place $p \in P$.

For a transition $t \in T$ an arc (x, t) is called an *input arc*, and an arc (t, x) — an *output arc*; the *preset* ${}^\bullet t$ and the *postset* $t^\bullet$ are defined as the multisets over P such that ${}^\bullet t(p) = F(p, t)$ and $t^\bullet(p) = F(t, p)$ for each $p \in P$.

A transition $t \in T$ is *enabled* in a marking m iff $\forall p \in P\ m(p) \geq F(p, t)$. An enabled transition t may *fire* yielding a new marking $m' =_{\text{def}} m - {}^\bullet t + t^\bullet$, i. e. $m'(p) = m(p) - F(p, t) + F(t, p)$ for each $p \in P$ (denoted $m \xrightarrow{t} m'$, $m \xrightarrow{\lambda(t)} m'$, or just $m \to m'$). We say that m' is reachable from m iff there is a (possibly empty) sequence of firings $m = m_1 \to \cdots \to m_n = m'$ and denote it by $m \xrightarrow{*} m'$.

Workflow nets (WF-nets) is a special subclass of Petri nets designed for modeling workflow processes. A workflow net has one initial and one final place, and every place or transition in it is on a directed path from the initial to the final place.

Definition 3 ((Labeled) workflow net). *A (labeled) Petri net N is called a (labeled)* workflow net (WF-net) *iff*

1. *There is one source place $i \in P$ and one sink place $f \in P$ s. t. ${}^\bullet i = f^\bullet = \emptyset$;*
2. *Every node from $P \cup T$ is on a path from i to f.*
3. *The initial marking in N contains the only token in its source place.*

By abuse of notation we denote by i both the source place and the initial marking in a WF-net. Similarly, we use f to denote the final marking in a WF-net N, defined as a marking containing the only token in the sink place f. Fig. 1 gives an example of a WF-net.

Definition 4 (Reachability graph). *A reachability graph (RG) for a PN N is a graph with vertices corresponding to markings in N and with arcs defined as follows: (m_1, m_2) is an arc in the RG iff $m_1 \to m_2$ in N.*

For a labeled PN its RG has arcs labeled with the corresponding transitions labels.

Definition 5 (Transition system). *A* transition system *(TS) is a tuple $TS = (S, E, T, s_{in})$, where S is a finite non-empty set of states, E is a set of events, $T \subset S \times E \times S$ is a transition relation, and s_{in} is an initial state. Elements of T are called transitions and (by abuse of notation) will be denoted by $s \xrightarrow{e} s'$. A state s is reachable from a state s' iff there is a possibly empty sequence of transitions leading from s to s' (denoted by $s \xrightarrow{*} s'$). Each TS must satisfy the following basic axioms:*

1. *No self-loops: $\forall (s \xrightarrow{e} s') \in T : s \neq s'$;*
2. *No multiple arcs between a pair of states: $\forall (s \xrightarrow{e_1} s_1), (s \xrightarrow{e_2} s_2) \in T : [s_1 = s_2$ implies $e_1 = e_2]$;*
3. *Every event has an occurrence: $\forall e \in E : \exists (s \xrightarrow{e} s') \in T$;*
4. *Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.*

We write $s \xrightarrow{e}$, or $\xrightarrow{e} s$ iff $\exists s' : s \xrightarrow{e} s'$, or $\exists s' : s' \xrightarrow{e} s$ correspondingly.

Now we define the notion of a region.

Definition 6 (Region). *Let $TS = (S, E, T, s_{in})$ be a transition system and $S' \subseteq S$ be a subset of states. S' is a region iff for each event $e \in E$ one of the following conditions hods:*

- *all the transitions $s_1 \xrightarrow{e} s_2$ enter S', i.e. $s_1 \notin S'$ and $s_2 \in S'$,*
- *all the transitions $s_1 \xrightarrow{e} s_2$ exit S', i.e. $s_1 \in S'$ and $s_2 \notin S'$,*
- *all the transitions $s_1 \xrightarrow{e} s_2$ do not cross S', i.e. $s_1, s_2 \in S'$ or $s_1, s_2 \notin S'$.*

Each TS has two *trivial regions*: the set of all states, and the empty set. For each state $s \in S$ we define the set of non-trivial regions, containing s (denoted by R_s). A region r' is said to be a subregion of a region r iff $r' \subseteq r$. A region r is called a minimal region iff it does not have any other subregions. A region r is a *pre-region* of an event e iff there is a transition labeled with e which exits r.

Now we define a notion of elementary transition system [8, 9].

Definition 7 (Elementary transition system(ETS)). *A* $TS = (S, E, T, s_{in})$ *is called elementary iff in addition to* 1-4 *it satisfies the following two axioms:*

5. *State separation property: two different states must belong to different sets of regions:*
   $\forall s, s' \in S : [(R_s = R_{s'})$ *implies* $(s = s')]$*;*
6. *Forward closure property: if state* s *is included in all pre-regions of event* e*, then* e *must be enabled by* s*:*
   $\forall s \in S \forall e \in E : [({}^{o}e \subseteq R_s)$ *implies* $(s \xrightarrow{e})]$.

A set S of states is called a *generalized excitation region* for an event a (denoted by $GER(a)$) iff S is a maximal (a maximal connected) set of states such that for every state $s \in S$ there is a transition $s \xrightarrow{a}$. An *excitation closure* condition is satisfied iff for each event $a : \bigcap_{r \in {}^{o}a} = GER(a)$.

4 Discovering a WF-net with cancellation regions

In this section we present a new method for process discovering, which allows constructing clear and readable process models with cancellations. To increase transparency and readability of process models with cancellations many languages for modeling business processes (such as BPMN, YAWL and others) use so called cancellation regions. A cancellation region is a subset of places in a model associated with a transition. Firing of this transition empties the region, i.e. removes all tokens happen to remain in its places.

In WF-nets cancellation regions can be naturally represented with the help of reset arcs. Now we define Petri nets with reset arcs and workflow nets with reset arcs (RWF-nets).

Definition 8 (Reset net). *A reset net is a tuple* (P, T, F, R)*, where*

- (P, T, F) *is a classical PN with places* P*, transitions* T*, and flow relation* F*,*
- $R : T \to 2^P$ *is a function mapping transitions to (possibly empty) subsets of places.*

For a transition $t \in T$, $R(t)$ is a subset of places, emptied by firing of t. When $p \in R(t)$, we say that (p, t) is a reset arc.

As in classical Petri nets a transition $t \in T$ is *enabled* in a marking m iff $\forall p \in P \; m(p) \geq F(p, t)$. An enabled transition t may *fire* yielding a new marking $m'(p) = \pi_{P \setminus R(t)}(m(p) - F(p, t)) + F(t, p)$ for each $p \in P$. Here $\pi_{P \setminus R(t)}$: $Nat^{|P|} \to Nat^{|P|}$ is a 'projection' function, which maps markings to markings by removing all tokens in reset places $R(t)$.

Definition 9 (Reset workflow net). *A reset net* N *is called a* reset workflow net (RWF-net) *iff*

1. *There is one source place* $i \in P$ *and one sink place* $f \in P$ *s. t.* ${}^{\bullet}i = f^{\bullet} = \emptyset$*;*
2. *Every node from* $P \cup T$ *is on a path from* i *to* f.

3. *The initial marking in N contains the only token in its source place.*
4. *There is no reset arc connected to the sink place, i.e., $\forall t \in T : o \notin R(t)$.*

An example of a RWF-net is shown in Fig. 7, reset arcs are denoted by double-headed arrows.

Given a log we construct a TS, states of which are formed on the basis of event sets. This can be done a standard way. After that we execute the procedure of merging the states with identical outflow. An example of a TS with states merged by outflow is presented in Fig. 4. Note that there might be situations when some dummy states and transitions should be added to a TS in order to derive a WF-net which has one source and one sink place [5].

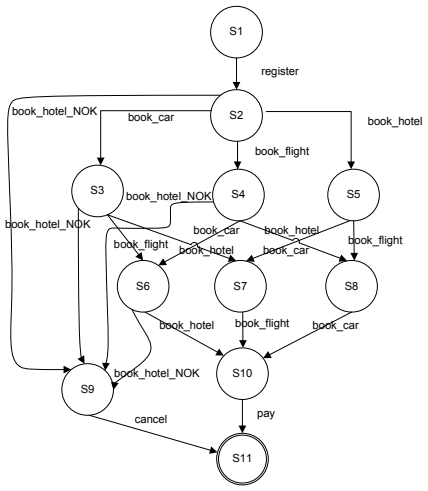


Fig. 4. Transition system for the log in Fig. 2

The state-based region algorithm [7–9] constructs a target PN in such a way that a TS is covered by its minimal regions and after that every minimal region is transformed to a place in a PN.

Our approach is based on the following assumption: *failure events* are always followed by some *catching event* in a log. In our example 'book_hotel_NOK' is such a *failure event*, and 'cancel' is a *catching event*.

Failure events inform us about errors during the process execution. A *cancellation state* is a state reached by a process after an occurrence of some *failure event*. *Catching events* inform about the work of a handler. And a *cancellation set* is a set of states which might be canceled in consequence of some error. To formalize these heuristics we now give the following definition.

Definition 10 (Cancellation state). *Let $TS = (S, E, T, s_{in})$ be a transition system. A state $s_c \in S$ is a cancellation state iff the following conditions hold:*

1. $\forall e \in E$ *s.t.* $(\overset{e}{\rightarrow} s_c)$ *we have* $(\forall s \in S : [\overset{e}{\rightarrow} s$ *implies* $(s = s_c)]$;
2. $\forall e \in E$ *s.t.* $(s_c \overset{e}{\rightarrow})$ *we have* $(\forall s \in S : [s \overset{e}{\rightarrow}$ *implies* $(s = s_c)]$;
3. $\forall e \in E$ *s.t.* $\overset{e}{\rightarrow} s_c$ *we have* $(\exists s_1, s_2 \in S : [((s_1, e, s), (s_2, e, s) \in T) \wedge (s_1 \neq s_2)])$;
4. $\exists! e : [s_c \overset{e}{\rightarrow}]$.

Definition 11 (Catching event). *Let* $TS = (S, E, T, s_{in})$ *be a transition system. An event* $e \in E$ *is a catching event iff* $(s_c \overset{e}{\rightarrow})$ *for some cancellation state* $s_c \in S$.

Definition 12 (Failure event). *Let* $TS = (S, E, T, s_{in})$ *be a transition system, a state* $s_c \in S$ *is a cancellation state,* e_f *is a failure event. A set of states* S *is a cancellation set for* e_f *(denoted by* $CS(e_f)$*) iff* $\forall s \in S$: $(s \overset{e}{\rightarrow} s_c)$.

All incoming and, correspondingly, outgoing transitions of a *cancellation state* are labeled with some separated events in TS. Events which label incoming transitions are called *failure events*. There is only one event which labels all outgoing transitions — a *catching event*. There are not less than two transitions for each *failure event*. The set of states which have outgoing transitions labeled with some *failure event* e_f is called a *cancellation set*, and is denoted by $CS(e_f)$.

In our example s_9 is a *cancellation state*, $\{s_2, s_3, s_4, s_6\}$ is a *cancellation set*, 'book_hotel_NOK' is a *failure event* and 'cancel' is a *catching event* (Fig. 4).

Note, that in our example each process terminates after the completion of the transition 'cancel', but in a general case 'cancel' may be followed by some other transitions.

Now we show that occurrence of cancellations in a log frequently leads to a complicated WF-net. This is also valid for our example (Fig. 3).

There could be such a situation when one of events occurs independently of the potential failures. See for example an event 'book_flight', booking flight procedure may start before the failure (and therefor can be interrupted) or after the successful completion of the booking hotel procedure without any possibility of being interrupted. The transitions labeled with 'book_flight' event connect states in the *cancellation set* and states which are not in the *cancellation set* at the same time. We now prove that if a TS contains a *cancellation state* as well as an event, which occurs independently of potential failures (independent parallel branches), then the generated labeled WF-net will contain transitions with identical labels.

Theorem 1. *Let* $TS = (S, E, T, s_{in})$ *be a transition system. A state* $s_c \in S$ *is a cancellation state,* $e_f \in E$ *is a failure event and* $CS(e_f)$ *is a corresponding cancellation set. Let* $e \in E$ *be an event for which the following conditions hold:*

- $e \neq e_f$;
- $\exists s^1 \in CS(e_f) : [(s^1 \overset{e}{\rightarrow})]$ – *there is a state from the cancellation set with an outgoing transition labeled by e;*
- $\exists s^2 \in CS(e_f) : [\neg(s^2 \overset{e}{\rightarrow})]$ – *the cancellation set contains a state that does not have outgoing transitions labeled by e;*

– $\exists s^3 \in S, s^3 \notin CS(e_f) : [(s^3 \xrightarrow{e})]$ – *there is a state not from the cancellation set which has outgoing transition labeled by e.*

Then the excitation closure condition for the event e is not satisfied.

The theorem conditions are illustrated by Fig. 5.

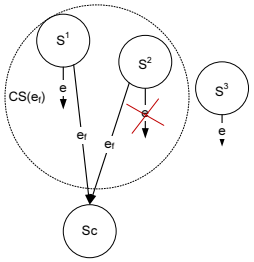


Fig. 5. Transition system for which excitation closure condition is not satisfied

Proof. We have to prove that $\bigcap_{r \in {}^o e} = GER(e)$ is not satisfied. Let us consider an arbitrary pre-region of e - r. According to the definition of a pre-region there is an exit transition labeled by e, that means that r contains all states with outgoing transitions labeled by e: $s^1, s^3 \in r$. Let e_f be a label for transitions which 'do not cross' the region r. Then $s_c \in r$, and hence $s^2 \in r$. If a transition e_f exits r, then we also have $s^2 \in r$. It means that every pre-region of e contains s^2 which does not have any outgoing transition labeled with e. In this case the *excitation closure* property $\bigcap_{r \in {}^o e} = GER(e)$ is not satisfied. □

By now we have proved that it is impossible to construct an equivalent labeled WF-net with transitions having unique labels in the presence of *cancellation state* and an event which occurs independently of the potential failures (existence of independent parallel branches), because in this case the *excitation closure* condition is not satisfied. If the *excitation closure* condition is not satisfied, then TS is not elementary and the target PN is splitted [8, 9]. It means that almost always an overcomplicated WF-net is obtained from logs with cancellations.

To overcome this problem we propose a new method of discovering a RWF-net from an event log. We start with discovering a regular structure of the process. For that we first construct a TS based on given event log. Then we delete all *cancellation states* together with their incident arcs from the TS (Fig. 6) and apply one of existing discovery algorithms to obtain a WF-net, representing the regular (without cancellations) behavior. The WF-net generated from this TS presented in Fig. 1. Then we add places, transitions and reset arcs needed for representing cancellations.

One may notice that according to the definition, the *cancellation state* forms a region itself, and hence it is transformed to a place of the target PN. So, this

place should be added to the target WF-net and connected by outgoing flows in a way it was connected in a WF-net generated from the TS with cancellation. The question remains, how to connect this place by incoming flows with other WF-net elements to achieve control flow simplicity and preserve semantics of the initial TS.

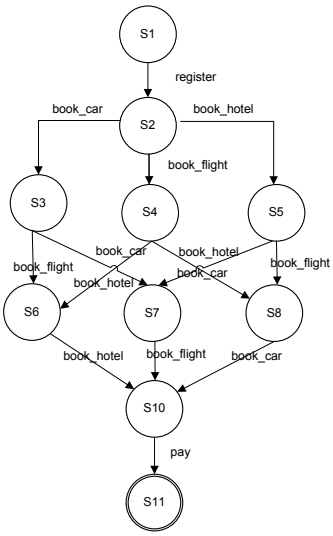


Fig. 6. Transition system without cancellations

We use an assumption that after deleting of the *cancellation state* each corresponding *cancellation set* is a minimal region. As we can see from the example above (Fig. 6) the case when *cancellation set* forms a minimal region might be rather common, especially when a process contains an exit transition labeled by a 'normal flow' event. Herein our example 'booking_hotel' is such a 'normal flow' event, which specifies the case when the booking hotel procedure has been terminated without failures.

Let us formalize the approach and prove its correctness under the assumption that after the deletion of the *cancellation state* each corresponding *cancellation set* is a minimal region.

Algorithm 1. *(Constructing a RWF-net from the TS with cancellations).*
Let $TS = (S, E, T, s_{in})$ be a transition system. Let $s_c \in S$ be a *cancellation state*, $e_{f_1}, ..., e_{f_n} \in E$ — *failure events*, $e_c \in E$ — a *catching event* and $CS(e_{f_1}), ..., CS(e_{f_n})$ — the corresponding *cancellation sets*.

1. Construct a TS $TS' = (S', E', T', s_{in})$ from TS by deleting the *cancellation state* and its incident arcs.

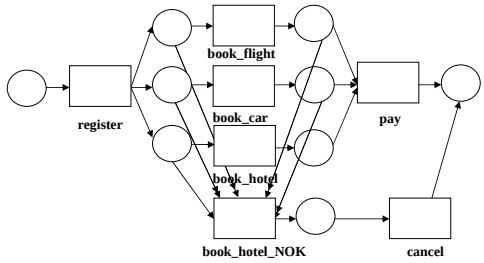


Fig. 7. Reset WF-net for booking process

2. Verify that $CS(e_{f_1}), ..., CS(e_{f_n})$ are minimal regions in TS', otherwise return a message that RWF-net cannot be constructed.
3. Construct WF' as a WF-net derived from TS' according to the state-based region algorithm. TS' is covered by its minimal regions and after that every minimal region is transformed to a place in WF' [7–9].
4. Perform the transformation of WF' by adding transitions corresponding to the *failure events* $e_{f_1}, ..., e_{f_n}$ and transition corresponding to the *catching event* $e_c \in E$ (Fig. 8).
5. Add outgoing control flow to the transition labeled by e_c, as if the state-based region algorithm was applied to the initial transition system TS.
6. For each *failure event* e_{f_i} a place corresponding to the minimal region $CS(e_{f_i})$ is connected by an outgoing arc with the transition denoting this *failure event*; all such transitions are connected by outgoing arcs with an additional place, which in turn is connected with the transition labeled by the *catching event* (Fig. 8). If there is only one *failure event* (see Fig. 7), it is connected directly with the transition labeled with the *catching event*.
7. All other places corresponding to the minimal regions, which contain states from $CS(e_{f_i})$, should be connected with the transition labeled by the *failure event* e_{f_i} by reset arcs.

The RWF-net which is manually constructed from the log (Fig. 2) according to Algorithm 1 is presented in Fig. 7. This RFW-net is structurally similar to the initial regular WF-net (Fig. 1) and the core process structure could be easily retrieved from the RWF-net. Let us prove the correctness of Algorithm 1.

Theorem 2. *Let $TS = (S, E, T, s_{in})$ be a transition system. Construct a RWF-net WF using an Algorithm 1. Then the labeled RG of WF is isomorphic (or split-isomorphic) to the minimized TS.*

Proof. The labeled RG of the intermediate WF-net WF' is safe and isomorphic (or split-isomorphic) to the minimized initial TS' according to the principles of the state-based region algorithm [6]. TS differs from TS' only in addition of the *cancellation state* $s_c \in S$ and its incident arcs (transitions). Let us consider an

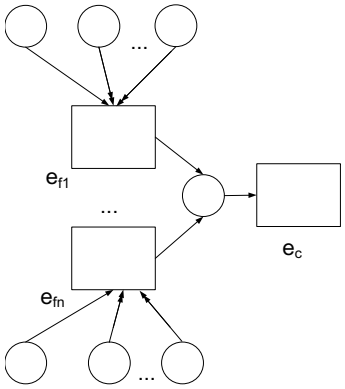


Fig. 8. Construction of the target RWF-net

arbitrary additional transition e_{f_i} which connects a state from the *cancellation set* $s \in S$ with the *cancellation state* $s_c \in S$. The presence of such a transition labeled by *failure event* e_{f_i} means that the target net can change its state having tokens in a place which corresponds to the *cancellation set* and other places which correspond to the minimal regions containing state s (these places have appropriate reset arcs in the target reset WF-net). Note that every minimal region in a TS corresponds to the place in the target WF-net [7–9]. And vice versa addition of new transitions, arcs and reset-arcs to the WF-net WF' will add necessary transitions to the TS. Also the outgoing flow of the transition labeled by *catching event* will be added according to the state-based region algorithm, taking into account that the *cancellation state* $s_c \in S$ forms a minimal region itself. All these arguments lead us to the conclusion that labeled RG of the target reset WF-net WF is isomorphic (or split-isomorphic) to the minimized initial TS.□

5 Conclusion

Construction of readable process models is an important requirement for process discovery techniques. Since cancellations occur in the majority of real-life event logs, it is necessary to construct an appropriate algorithm to deal with cancellations and synthesize simple and clear process models. In this work we have proved that occurrence of cancellations in a log frequently leads to process models with the overcomplicated control flow. We also described an algorithm, which discovers readable RWF-net models with clear regular structure from event logs. We have proved the correctness of this algorithm. In the future, we plan to implement this approach and apply it to real-life event logs.

References

1. W.M.P. van der Aalst *Discovery, Conformance and Enhancement of Business Processes.* Springer-Verlag, Berlin, 2011.
2. Object Management Group *Business Process Modeling Notation (BPMN) Version 2.0, OMG Final Adopted Specification.* Object Management Group, 2011.
3. W.M.P. van der Aalst and A.H.M. ter Hofstede YAWL: Yet another workflow language. *Information Systems.* Vol. 30, Nr. 4, pages 245-275. 2005.
4. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, MVoorhoeve, M.T. Wynn Soundness of Workflow Nets with Reset Arcs. *Transactions on Petri Nets and Other Models of Concurrency.* Vol. 3, pages 50-70. 2005.
5. W.M.P. van der Aalst Discovery, Verification and Conformance of Workflows with Cancellation. In 4th International Conference, ICGT 2008, Vol. 5214 of *Lecture Notes in Computer Science*, pages 18-37. Springer, 2008.
6. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. G?unther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.
7. M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical computer science.* Vol. 96, pages 3-33. 1992.
8. J. Cortadella,M.Kishinevsky,L. Lavagno, and A.Yakovlev. Synthesizing Petri nets from state-based models. Technical Report RR 95/09 UPC/DAC, Universitat Politecnica de Catalunya, April 1995.
9. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers.* Vol. 47, Nr. 8, pages 859-882. 1998.
10. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, Application and Theory of Petri Nets, Vol. 3536 of *Lecture Notes in Computer Science*, pages 444-454. Springer-Verlag, Berlin, 2005.

Genetic Algorithm with Path Relinking for the Orienteering Problem with Time Windows

Joanna Karbowska-Chilinska and Pawel Zabielski

Faculty of Computer Science
Bialystok University of Technology, Poland

Abstract. The Orienteering Problem with Time Windows (OPTW) is an optimisation NP-hard problem. This paper proposes a hybrid genetic algorithm (GAPR) for approximating a solution to the OPTW. Instead of the usual crossover we use a path relinking (PR) strategy as a form of intensification solution. This approach generates a new solution by exploring trajectories between two random solutions: genes not present in one solution are included in the other one. Experiments performed on popular benchmark instances show that the proposed GAPR outperforms our previously published version of GA and yields better results than the well-known iterated local search method (ILS) as well.

Keywords: orienteering problem with time windows, genetic algorithm, path relinking

1 Introduction

The Orienteering Problem with Time Windows (OPTW) is a type of optimisation routing problem first introduced by Kantor et al. [7]. The OPTW can be modelled as a weighted graph with a positive score/profit associated with each vertex. Let G be a graph with n vertices, in which each vertex i has a profit p_i, a service time T_i and a time window $[O_i, C_i]$, where O_i and C_i denote the opening and closing times of a vertex i. Each edge between vertices i and j has a fixed cost t_{ij} associated with it. The value t_{ij} is interpreted as the time or length needed to travel between vertices. The objective is to determine a single route, from a starting point s to a fixed ending point e, that visits some of the vertices within the fixed time windows and maximises the total profit. In addition, the total cost of the edges on the path must be less than the given constraint t_{max} and each vertex on the route is visited only once. It is possible to wait at a vertex for service before its time windows opens.

The OPTW is derived from the more general Orienteering Problem (OP) [23], [14]. In the OP each vertex inserted into the route could be visited in any time interval (there is no restriction in the form of time windows). The OP is seen as a combination of the Knapsack Problem and the Travelling Salesperson Problem, because in the OP the selected route limited in length contains the most profitable vertices. Both the OP and the OPTW are NP-hard [7].

Numerous applications can be found for the OPTW, e.g. in logistics for planning optimal routes, such as profitable delivery routes, as well as in optimisation of production scheduling [20]. The OPTW successfully models problems related to tourism [24]. Tourists visiting a city are usually unable to visit all points of interests (POI) because they are limited by time or money. The most effective heuristics for the OPTW are applied in electronic devices known as mobile tourist guides [24], [2] which make it possible to visit the most valuable POIs (taking into account their opening and closing times) within a fixed time limit. The Team Orienteering Problem with Time Windows (TOPTW) [12], which is an extension of the OPTW, is used to model multiple tours, with each tour satisfying the same fixed travel length or time constraint.

In this paper we present an improved version of the genetic algorithm for the OPTW described in [10]. We use hybridization of our genetic algorithm with a path relinking method (PR) instead of the crossover operator. In the PR approach two random solutions are chosen and routes combining these solutions are explored to provide better solutions: genes not present in one solution are included in the other one. Path relinking was originally described by Glover [5] and Laguna [6] for intensification and diversification of the tabu search method. Moreover, the PR significantly improves the results of the Greedy Randomised Adaptive Search Procedure (GRASP) for the general version of OP [21], [1]. This led us to use this method in combination with the previously developed genetic algorithm for solving the OPTW [10].

The remainder of the paper is organised as follows. The mathematical formulation of the problem is presented in Section 2. An overview of the main approaches in the literature is presented in Section 3. In section 4, we describe the concept of the hybrid genetic algorithm with path relinking. The results of computational experiments illustrating the effectiveness of our approach in comparison with other methods are discussed in Section 5. Concluding remarks and plans for further research are given in Section 6.

2 Mathematical formulation

Based on the notation introduced in the previous section the OPTW can be formulated as an mixed integer problem as follows [26]:

$$max \sum_{i=1}^{n-1} \sum_{j=2}^{n} p_i x_{ij} \tag{1}$$

$$\sum_{j=2}^{n} x_{1j} = \sum_{i=1}^{n-1} x_{in} = 1 \tag{2}$$

$$\sum_{i=1}^{n-1} x_{ik} = \sum_{j=2}^{n} x_{kj} \leqslant 1 \quad \forall k \in \{2, ..., n-1\} \tag{3}$$

$$\sum_{i=1}^{n-1}\sum_{j=2}^{n} t_{ij}x_{ij} \leqslant t_{max} \tag{4}$$

$$start_i + T_i + t_{ij} - start_j \leqslant M \cdot (1 - x_{ij}) \tag{5}$$

$$O_i \leqslant start_i \leqslant C_i \quad \forall i = 1, ..., n \tag{6}$$

where x_{ij} are binary variables, such that $x_{ij} = 1$ if the edge between i and j is included in a solution, and $x_{ij} = 0$ otherwise. Moreover, we assume that s=1 and $e = n$. Let $start_i$ denote the start of service time at vertex i and M be a large constant. The objective function (1) maximises the total collected profit of the route. The constraint in (2) guarantees that the path starts at vertex 1 and ends at vertex n. Constraint (3) requires that there may be at most one visit to any vertex. The constraint in (4) ensures that the time of the route is limited by t_{max}. Constraint (5) ensures the timeline of the route. The start of the service is restricted by a time window as in (6).

3 Literature review

It can be easily observed that the OPTW is a special case of the TOPTW: in the OPTW one route is constructed, while in TOPTW several routes are generated. Methods for the TOPTW could also be applied to the OPTW. Therefore, in this section we present solution approaches described in the literature for the OPTW as well as the TOPTW.

The Orienteering Problem with Time Windows has been studied since Kantor and Rosenweins article [7]. Their insertion heuristic constructs a route by iteratively inserting the vertex with the highest ratio $score/TimeInsertion$ without violating time windows and t_{max} constraints. In the second method proposed, they developed what is known as a tree heuristic, in which a depth-search algorithm constructs routes that begin in a given vertex. If a route is infeasible or unlikely to yield a better result, the route is abandoned. In this case the algorithm backtracks to the previous level of the tree and attempts to insert another vertex.

Righini et al. [18] developed an exact optimisation algorithm for the OPTW based on bi-directional dynamic programming. This technique requires extension of non-dominated states from both sides of the route: forward from the start vertex and backward from the end vertex. The decremental state relaxation method was also introduced for this algorithm with the idea of iteratively reducing the number of explored states [19].

Mansini et al. [17] introduced the Granular Variable Neighbour Search approach for TOPTW based on the idea of exploring a reduced neighbourhood instead of a complete one and not including arcs that are not promising. The

method improved algorithm efficiency with no loss of effectiveness. A more general concept of granularity was described in [11]. The cost of an arc was identified by the formula $(t_{ij} + w_{ij})/(p_i + p_j)$, where w_{ij} is the maximum possible waiting time at j provided that the service at i is assumed to start at the fixed time. Promising arcs were identified as follows: the lower the reduced cost associated with the arc, the higher the probability it will belong to a good solution.

Montemanni et al. [15] proposed a solution model using hierarchical generalization of TOPTW based on an Ant Colony System (ACS) algorithm. Two improvements to the ASC method were included in the ACS [16]: the constructive phase was sped up by considering the best solution computed so far, and the local search procedure was applied only to those solutions to which it had not been applied in the previous iteration (the same route was not optimised too often).

Tricoire et al. [22] adapted a solution to the Multi-Period Orienteering Problem with Multiple Time Windows for the TOPTW. They proposed an exact algorithm for the path feasibility sub-problems, and embedded it in a variable neighbourhood search (VNS) approach to solve the whole problem.

Vansteenwegen et al. proposed the Iterated Local Search approach (ILS) [25] to tackle the TOPTW. Because it is the fastest known heuristic, ILS is applied, for example, in electronic devices such as mobile tourist guides [4], [24]. The ILS method iteratively builds and improves one route by combining an insertion step and deletion of some consecutive locations (a shake step) to escape from a local maximum.

Lambadie et al. [12] developed a method for solving TOPTW which combines a greedy randomised adaptive search procedure (GRASP) with an evolutionary local search (ELS). In the ELS phase deletion and insertion mutations are performed for multiple child solutions. Child solutions are further improved by a variable neighbourhood descent procedure. The GRASP ELS method gives the best results on benchmark instances in comparison with the other methods mentioned [11]

4 Genetic Algorithm

The proposed method, called GAPR, is an extended version of the genetic algorithm GA proposed in [10]. The individuals (routes) are encoded as a sequence of vertices (genes). The GAPR starts by generating an initial population of P_{size} routes. Next, each individual is evaluated by means of the fitness function F. We use F as in [10], [8], [9], which is equal to $TotalProfit^3/TravelTime$. $TotalProfit$ and $TravelTime$ denote the sum of the profits assigned to the vertices on the route and the total travel time from the starting point to the ending point. In subsequent iterations of the GAPR the population is evolved by applying genetic operators selection, recombination and mutation in order to create new, better routes. The optimisation strategy, in contrast with the method proposed in [10], involves a recombination stage (crossover) performed on two random routes: instead of randomly choosing a crossing point between

vertices with similar time windows and starting and ending time of service [10], we used a path relinking process. In this process routes in the graph solution space connecting two random solutions are explored in order to find better solutions [5], [6]. To generate new solutions between selected random routes, genes not present in one route are included in the other. The solution generated by the path relinking process corresponds to the best individuals that could be obtained by applying the crossover operator to the same random parents.

The GAPR terminates after a fixed number of generations (denoted by N_g), or earlier if it converges. The GAPR result is the route in the final population with the highest profit value. The basic structure of the GAPR is as follows:

```
compute initial population;
algLoop=0;
while algLoop< Ng do
   algLoop++;
   tournament grouping selection;
   path relinking;
   mutation;
   if no improvements in last 100 iterations then break;
end;
return the route with the highest profit value;
```

Due to randomization, the GAPR is run several times during the tests. Each successive repetition of the GAPR is independent of the others, so this is a prime target for parallelisation. OpenMP [13], which is an API, is used in the algorithm for parallel computations, which substantially reduce its execution time.
The application of genetic operators for selection, recombination and creation of new individuals is described in more detail below.

4.1 Initialisation

In the approach presented a route is coded as a sequence of vertices. A population of P_{size} routes is generated as follows. First the chromosome is initialized by the s and e vertices. Then the following values are assigned sequentially to the initialized vertices: $arrive_i$ - arrival time at vertex i, $wait_i$ - waiting time, if the arrival at the vertex i is before opening time, $start_i$ and end_i - starting and ending service time at vertex i. Moreover, the maximum time the service of a visit i can be delayed without making other visits infeasible is calculated for each location in the route as follows [25]:

$$MaxShift_i = Min(C_i - start_i - T_i, wait_{i+1} + MaxShift_{i+1}) \tag{7}$$

Let l be the predecessor of vertex e in the route. In the subsequent steps a set of vertices is prepared. Each vertex v from this set is adjacent to vertex l and vertex e and will satisfy the following conditions after insertion: (a) $start_v$ and end_v are within the range $[O_v, C_v]$; (b) the locations after v could be visited

in the route; and (c) the current travel length does not exceed the given t_{max} (including consumption time to insert the vertex v between l and e). A random vertex v is chosen from this set. The values $arrive_v$, $wait_v$, $start_v$ and end_v are calculated and the vertex v is inserted. After the insertion, the values $arrive_e$, $wait_e$, $start_e$ and end_e are updated. Moreover, for each vertex in the tour (from vertex e to s) the $MaxShift$ value is updated as well. The tour generation is continued for as long as locations that have not been included are present and t_{max} is not exceeded.

4.2 Selection

We use tournament grouping selection, which yields better adapted individuals than standard tournament selection [9]. In this method a set of P_{size} individuals is divided into k groups and the tournaments are carried out sequentially in each of the groups. t_{size} random individuals are removed from the group, the chromosome with the highest value for the fitness function $TotalProfit^3/TravelTime$ is copied to the next population, and the t_{size} previously chosen individuals are returned to the old group. After selection from the group currently analysed has been repeated P_{size}/k times, P_{size}/k individuals are chosen for a new population. Finally, when this step has been repeated in each of the remaining groups, a new population is created, containing P_{size} routes.

4.3 Path relinking

First two random routes R_1 and R_2 are selected from the new population chosen in the selection step. Let $V_{R_1-R_2}$ be the set of vertices present in R_1 and not in R_2 , and let $V_{R_2-R_1}$ denote the set of vertices present in R_2 and not in R_1. During $\mathrm{PR}(R_1, R_2)$ we attempt to insert vertices from $V_{R_2-R_1}$ into R_1 in the best possible position. The total consumption time associated with inserting a vertex j between vertex i and k is calculated as follows [25]: $Shift_j = t_{ij} + wait_j + T_j + t_{jk} - t_{ik}$. In addition, we check whether the shift resulting from the new insertion exceeds the constraints associated with the previously calculated $wait$ and $MaxShift$ values for the vertices located directly after the newly inserted one. If the shift exceeds the constraints the vertices from $V_{R_1-R_2}$ are removed to restore the possibility of inserting new locations. For each vertex u from this set a ratio is calculated as follows: $RemovalRatio = (p_u)^2/(end_u - arrive_u)$, with the power 2 having been determined experimentally. After this computation the vertex with the smallest value for $RemovalRatio$ is removed. This removal is repeated until we can insert some vertices into the path. Finally the vertex u with the highest value for $(p_u)^2/Shift(u)$ and not exceeded the mentioned constrains is selected for insertion. After u is inserted the values of $arrive_u$, $wait_u$, $start_u$ and end_u are calculated. For each location after u the arrival time, waiting time, and start and end of service are updated. $MaxShift$ values are also updated for the vertices from the starting point to the ending point of the route. As we can see, the insertion of one vertex from $V_{R_2-R_1}$ into R_1 is a multi-stage process. The process is repeated for as long as t_{max} is not exceeded and

the set $V_{R_2-R_1}$ is not empty. In addition, we perform PR(R_2, R_1) by inserting vertices from $V_{R_1-R_2}$ into R_2. Two new routes are created as a result of PR(R_1, R_2) and PR(R_2, R_1). If the fitness values of the new routes are higher than the fitness value of R_1 and R_2, they replace them.

4.4 Mutation

In this phase a random route is selected from P_{size} individuals. Two types of mutation are possible a gene insertion or gene removal (the probability of each is 0.5). The mutation process is repeated on the selected route N_m times, where N_m is the parameter. During the *insertion mutation*, all possibilities for inclusion of each new vertex (not present in the route) are considered in the same way as in the path relinking process. The locations before and after the inserted vertex should be updated as in the case of the insertion process in the path relinking. In the *deletion mutation* we remove a randomly selected gene (excluding the first and last genes) in order to shorten the travel length. After the gene is removed, all locations after the removed gene are shifted towards the beginning of the route. Furthermore, the locations before and after the removed gene are updated as in the insertion mutation.

5 Experimental Results

The GAPR was coded in C++ and run on an Intel Core i7, 1.73 GHz CPU (turbo boost to 2.93 GHz). The algorithm was tested on Solomon [20] and Cordeau [3] test instances for the OPTW. The number of vertices in the Solomon instances is equal to 100 and different layouts for the vertices are considered: cluster (c), random (r) and random-clustered (rc) classes. The Solomon benchmarks c200, r200, rc200 and c100, r100, rc100 have the same coordinates of vertices, profits and visiting times, but the $c \setminus r \setminus rc200$ instances have approximately three times higher values of t_{max} and larger time windows than the $c \setminus r \setminus rc100$ instances. The Cordeau instances vary between 48 and 288 vertices.

The parameters of the GAPR were determined by performing several tests on a selected subset of Solomon and Cordeau instances. Preliminary tests identified the following as good performing parameters: 150 for the initial population size, 3 for the number of individuals chosen from the group in the tournament selection, and 15 for the number of groups in the tournament selection. Based on the tests described in [10], the N_m number of mutations repeated on the selected route was set to 15.

Detailed results obtained by the GAPR on benchmark instances in comparison with other methods are given in Tables 2 - 4. There are two columns for the GAPR, denoted GAPR(I) and GAPR(II). The first reports the results obtained by considering only PR(R_1, R_2) in the path relinking (in each iteration of the algorithm). The second shows the results of the use of both PR(R_1, R_2) and PR(R_2, R_1) during the path relinking. For comparison of the results, the best known solution value (BK) (solutions obtained by GRASP ELS and ACS

algorithm [12], [16]), the GA (with crossover) [10] and ILS [25] are also reported. In the BK columns the optimal values when are known are marked in italic. The GAPR was tested by performing sixteen runs concurrently two runs each on eight processor cores. The results of the GA were obtained with sixteen runs of the algorithm (without concurrency) on the same computer used to run the GAPR. The total time of the sixteen runs (expressed in seconds) and the minimum, average, and maximum solutions are given in the tables for the GA and the GAPR. The ILS (deterministic algorithm) results were obtained with one run [25]. Tables 2 - 4 also show the average percentage gap between the best known solution (BK) values and the average value of the GAPR, and for comparison, the gaps between BK and the other methods mentioned. An empty cell denotes a gap equal to 0.

The results presented in Tables 2 - 4 indicate that the GAPR outperforms the ILS results on $c \setminus r \setminus rc100$, $c \setminus rc200$ and the Cordeau instances. Only in the case of $r200$ does the ILS perform slightly better than the GAPR (the ILS has a smaller gap than the GAPR, by about 0.4%). The average gap between the BK and the ILS results for all these instances is 3.6%, while the gaps between BK and GAPR(I) and GAPR(II) are 2.4% and 2.5%, respectively. Because the use of PR(R_1, R_2) and PR(R_2, R_1) results in faster convergence of the algorithm, in some cases (e.g. rc200) the creation of two new routes in each iteration of GAPR(II) yields worse results than calculation of only one route as in GAPR(I). For comparison, the average gap between BK and the previous genetic algorithm GA is 5.6%. The application of the path relinking stage in place of the crossover significantly improves the GAPR results by about 6% in comparison to the GA. As a result of the parallel computing, the GAPR is on average 22 times faster than the GA and its execution time is comparable with the ILS. Moreover, the GAPR provides several new best solutions on Cordeau instances p11, p15, p17 and p19, whose improved values are given in bold in Table 4. There are not an known optimal values for these instances [11]. Examples of the best-generated routes by GAPR for pr17 and pr15 are presented in Figure 2. Comparison the GAPR with the GA and the ILS results for these benchmarks is presented in Table 1.

The number of generations in the GAPR was experimentally set to 500 as the stopping criterion. As seen in Figure 1 in the case of the pr11-20 the best routes were generated earlier than 500 generations (the exception is the pr20, where the result was only better about 2% after 620 generations). Therefore, for the optimisation of the execution time, the algorithm was stopped earlier if were not any improvements in the lengths of the routes by 100 generations.

6 Conclusions and Further Work

This paper presents the application of the genetic algorithm hybridised with the path relinking method to the Orienteering Problem with Time Windows. Using path relinking instead of crossover improves the results on benchmark instances by about 6%. Moreover, the proposed GAPR algorithm outperforms

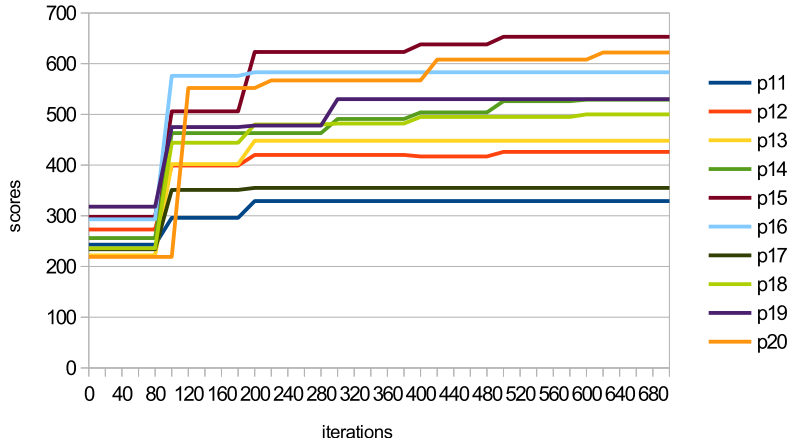


Fig. 1. Convergence of the GAPR for pr11-20 instances.

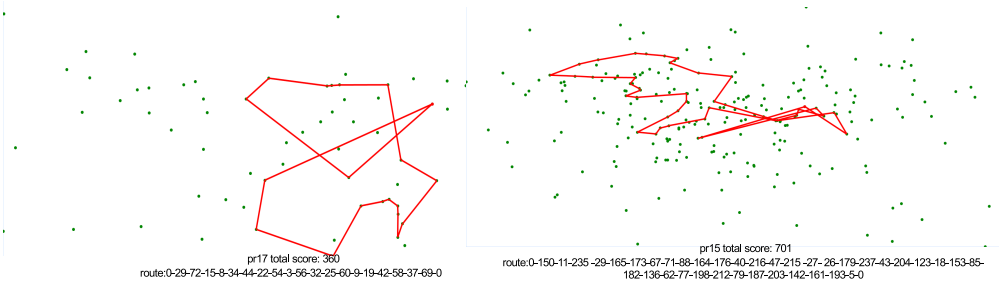


Fig. 2. Examples of the best-generated routes by GAPR for pr17 and pr15.

the results of the ILS heuristic, while the execution times of the two algorithms are comparable. The ILS is very fast and is applied, for example, in mobile tourist guide applications [25]. The proposed heuristic can also be adapted to solve problems related to planning tourist routes.

Further research directions include improving the GAPR results by applying path relinking operators between pairs of elite solutions (known as evolutionary path relinking) and conducting tests on a realistic database. Moreover, we intend to focus our research on developing effective heuristics for the Team Orienteering Problem with Time Windows, which is an extension of the OPTW.

Table 1. Comparison the GAPR with the GA and the ILS routes for pr15 and pr17.

	method	length of the route	profit	route
p15	ILS	69643	630	0-165-158-91-11-81-102-29-64- -62-136-77-198-182-85-153-18-238-123-204-43-237-179- -26-144-119-67-129-154-193-142-203-55-187-161-5-0
	GA	60801	653	0-150-11-29-235-165-173-67-88- -164-71-119-144-26-179-34-204-123-43-72-18-153-85- -182-136-62-77-198-212-79-203-142-187-161-193-0
	GAPR	68775	701	0-150-11-235-29-165-173-67-71-88-164-176- -40-216-47-215-27-26-179-237-43-204-123-18-153-85- -182-136-62-77-198-212-79-187-203-142-161-193-5-0
p17	ILS	70696	346	0-35-32-54-22-44-34-8-4- -21-48-66-30-60-9-19-42-58-61-37-69-0
	GA	70324	353	0-29-63-5-15-56-54-8-44- -34-22-32-6-9-19-42-58-37-69-0
	GAPR	70955	360	0-29-72-15-8-34-44-22-5-3- -56-32-25-60-9-19-42-58-37-69-0

Acknowledgements

The authors gratefully acknowledge support from the Polish Ministry of Science and Higher Education at the Bialystok University of Technology (grant S/WI/1/2011 and W/WI/2/2013).

References

[1] Campos, V., Marti, R.,Sanchez-Oro, J., Duarte, A.: Grasp with Path Relinking for the Orienteering Problem. Technical Raport, 116 (2012)
[2] http://www.citytripplanner.com/en/home . Last access: June 29, 2013
[3] Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30(2)**(1997) 105-119
[4] Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaz, M.: Integrating Public Transportation in Personalised Electronic Tourist Guides. Computers & Operations Research. **40** (2013) 758–774
[5] Glover, F.: A Template For Scatter Search And Path Relinking. Lecture Notes in Computer Science. **1363** (1997) 13-54
[6] Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers. Boston (1997)
[7] Kantor, M., Rosenwein, M .: The Orienteering Problem with Time Windows. Journal of the Operational Research Society. **43** (1992) 629–635
[8] Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K., Zabielski, P.: Genetic algorithm solving orienteering problem in large networks. Frontiers in Artificial Intelligence and Applications. **243** (2012) 28–38

[9] Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K., Zabielski, P.: A Genetic Algorithm with Grouping Selection and Searching Operators for the Orienteering Problem. (under review)
[10] Karbowska-Chilinska, J., Zabielski, P.:A Genetic Algorithm Solving Orienteering Problem with Time Windows. (accepted for publication in Springer series: Advances in Intelligent Systems and Computing)
[11] Labadie, N., Mansini, R., Melechovsky, J., Wolfler Calvo, R.: The Team Orienteering Problem with Time Windows: An LP-based Granular Variable Neighborhood Search. European Journal of Operational Research. **220(1)** (2012) 15-27
[12] Labadie, N., Melechovsk, J., Wolfler Calvo, R.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics. **17(6)** (2011) 729-753
[13] http://openmp.org/wp/ . Last access: June 29, 2013
[14] Ostrowski, K., Koszelew, J.: The comparison of genetic algorithm which solve Orienteering Problem using complete an incomplete graph. Zeszyty Naukowe, Politechnika Bialostocka. Informatyka **8** (2011), 61–77
[15] Montemanni, R., Gambardella, L.M.: Ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences. **34** (2009)
[16] Montemanni, R., Weyland, D., Gambardella L. M.: An Enhanced Ant Colony System for the Team Orienteering Problem with Time Windows. Proceedings of IEEE ISCCS 2011 The 2011 International Symposium on Computer Science and Society, Kota Kinabalu, Malaysia 381-384 (2011)
[17] Mansini, R., Pelizzari, M. Wolfler, R.: A Granular Variable Neighborhood Search for the Tour Orienteering Problem with Time Windows, Technical Report of the Department of Electronics for Automation, University of Brescia (2006)
[18] Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path. Networks. **51(3)** (2008) 155 –170
[19] Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the Orienteering Problem with Time Windows with dynamic programming. Computers & Operations Research. **36** (2009) 1191 – 1203
[20] Solomon, M.: Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research **35(2)** (1987) 254-265
[21] Souffriau, W.,Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.: A Path Relinking approach for the Team Orienteering Problem, Computers & Operations Research, **37(11)** (2010) 1853-1859
[22] Tricoire, F., Romauch, M., Doerner, K. F., Hartl, R. F. Heuristics for the multi-period orienteering problem with multiple time windows. Comput. Oper. Res. **34(2)** (2010) 351–367
[23] Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society. **35(9)** (1984) 797–809
[24] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: The City Trip Planner: An expert system for tourists. Expert Systems with Applications. **38(6)** (2011) 6540–6546
[25] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers O.R. **36** (2009) 3281–3290
[26] Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.. The Orienteering Problem: A survey. European Journal of Operational Research. **209(1)** (2011) 1–10.

Table 2. Results for Solomon's test problems (n=100).

	BK	ILS		GA				GAPR(I)				GAPR(II)					% gap BK with			
name	score	score	time	min	avg.	max	time	min	avg.	max	time	min	avg.	max	time		ILS	GA	GAPR(I)	GAPR(II)
c101	*320*	320	0.4	310	317	320	4.2	320	320	320	0.2	320	320	320	0.2		1.0			
c102	*360*	360	0.3	360	360	360	4.4	360	360	360	0.2	360	360	360	0.3					
c103	*400*	390	0.5	380	389	400	6.3	400	400	400	0.3	400	400	400	0.3		2.5	2.8		
c104	*420*	400	0.3	400	403	420	6.3	420	420	420	0.3	400	409	410	0.4		4.8	4.2		2.6
c105	*340*	340	0.3	340	340	340	4.3	340	340	340	0.2	340	340	340	0.2					
c106	*340*	340	0.3	340	340	340	4.0	340	340	340	0.2	340	340	340	0.2					
c107	*370*	360	0.3	360	362	370	4.5	360	360	360	0.2	360	360	360	0.2		2.7	2.2	2.7	2.7
c108	*370*	370	0.3	370	370	370	5.1	370	370	370	0.3	370	370	370	0.2					
c109	*380*	380	0.3	380	380	380	4.9	380	380	380	0.3	380	380	380	0.2					
sum:	**3300**	**3260**	**3.0**	**3240**	**3260**	**3300**	**44**	**3290**	**3290**	**3290**	**2.2**	**3270**	**3279**	**3280**	**2.2**	**avg.:**	**1.2**	**1.2**	**0.3**	**0.6**
r101	*198*	182	0.1	182	189	198	3.2	197	197	197	0.2	198	198	198	0.2		8.1	4.8	0.5	
r102	*286*	286	0.2	281	286	286	5.4	286	286	286	0.2	286	286	286	0.2			0.1		
r103	*293*	286	0.2	286	290	293	5.3	293	293	293	0.3	293	293	293	0.3		2.4	1.1		
r104	*303*	297	0.2	297	297	298	4.9	303	303	303	0.3	297	298	298	0.3		2.0	2.0		1.7
r105	*247*	247	0.1	240	244	247	3.9	247	247	247	0.2	247	247	247	0.2			1.2		
r106	*293*	293	0.2	281	292	293	4.7	293	293	293	0.2	293	293	293	0.2			0.3		
r107	*299*	288	0.2	286	292	297	6.1	299	299	299	0.3	299	299	299	0.3		3.7	2.3		
r108	*308*	297	0.2	297	300	308	5.5	308	308	308	0.4	303	308	308	0.4		3.6	2.5		
r109	*277*	276	0.2	258	270	274	4.6	270	272	274	0.2	270	273	274	0.3		0.4	2.5	1.7	1.4
r110	*284*	281	0.3	274	277	281	4.7	283	283	283	0.3	281	281	281	0.2		1.1	2.4	0.4	1.1
r111	*297*	295	0.2	275	293	295	5.8	297	297	297	0.3	297	297	297	0.3		0.7	1.3		
r112	*298*	295	0.2	290	293	295	6.2	292	292	292	0.2	295	295	295	0.3		1.0	1.6	2.0	1.0
sum.	**3383**	**3323**	**2.3**	**3247**	**3323**	**3365**	**60.4**	**3368**	**3370**	**3372**	**3.0**	**3359**	**3368**	**3369**	**3.2**	**avg.:**	**1.8**	**1.8**	**0.4**	**0.4**
rc101	*219*	219	0.2	210	216	219	3.5	216	216	216	0.2	216	216	216	0.2			1.4	1.4	1.4
rc102	*266*	259	0.2	255	261	266	4.8	266	266	266	0.3	266	266	266	0.2		2.6	1.9		
rc103	*266*	265	0.3	253	262	266	5.3	259	265	266	0.3	265	265	266	0.3		0.4	1.5	0.5	0.4
rc104	*301*	297	0.3	276	294	301	6.1	301	301	301	0.2	297	301	301	0.3		1.3	2.3		
rc105	*244*	221	0.2	230	236	241	4.5	241	241	241	0.2	241	241	241	0.2		9.4	3.3	1.2	1.2
rc106	*252*	239	0.2	233	245	250	4.8	128	242	250	0.2	249	250	250	0.3		5.2	2.8	3.8	0.8
rc107	*277*	274	0.2	264	269	274	5.0	122	258	277	0.2	271	273	277	0.3		1.1	2.9	6.9	1.4
rc108	*298*	288	0.2	278	289	298	4.9	298	298	298	0.2	288	297	298	0.3		3.4	3.0		0.3
sum.	**2123**	**2062**	**1.8**	**1809**	**2072**	**2115**	**39.1**	**1831**	**2087**	**2115**	**1.8**	**2093**	**2109**	**2115**	**2.1**	**avg.:**	**2.9**	**2.4**	**1.7**	**0.7**

Table 3. Results for Solomon's test problems, cont. (n=100).

	BK	ILS		GA				GAPR(I)				GAPR(II)					% gap BK with			
name	score	score	time	min	avg.	max	time	min	avg.	max	time	min	avg.	max	time		ILS	GA	GAPR(I)	GAPR(II)
c201	870	840	1.1	840	848	860	13.2	860	860	860	0.4	860	860	860	0.6		3.4	2.5	1.1	1.1
c202	930	910	2.8	890	901	920	15.9	920	920	920	0.5	920	924	930	0.7		2.2	3.1	1.1	0.6
c203	960	940	1.7	910	931	940	17.8	950	957	960	0.7	960	960	960	0.7		2.1	3.0	0.3	
c204	980	950	1.6	940	952	970	24.2	960	968	970	0.8	970	970	970	0.8		3.1	2.9	1.3	1.0
c205	910	900	1.2	880	893	900	16	900	900	900	0.4	900	900	900	0.5		1.1	1.9	1.1	1.1
c206	930	910	1.6	890	905	910	15.6	900	919	920	0.5	850	906	910	0.5		2.2	2.7	1.2	2.6
c207	930	910	2.1	890	910	930	16.1	930	930	930	0.7	920	920	920	0.5		2.2	2.2		1.1
c208	950	930	1.6	920	931	940	15.3	940	940	940	0.5	950	950	950	0.5		2.1	2.0	1.1	
sum:	**7460**	**7290**	**13.7**	**7160**	**7271**	**7370**	**134.2**	**7360**	**7394**	**7400**	**4.5**	**7330**	**7390**	**7400**	**4.8**	**avg.:**	**2.3**	**2.5**	**0.9**	**0.9**
r201	797	788	1.2	733	760	782	18.9	772	776	778	0.6	787	789	789	0.8		1.1	4.6	2.6	1.0
r202	929	880	1.4	834	867	892	28.3	883	885	891	1.1	888	890	892	0.8		5.3	6.7	4.8	4.2
r203	1021	980	1.6	925	954	980	35.1	965	969	974	1.2	961	968	979	1.2		4.0	6.6	5.1	5.2
r204	1086	1073	1.7	968	1018	1051	40.4	1031	1031	1031	1.0	1042	1042	1042	0.8		1.2	6.4	5.1	4.1
r205	953	931	1.4	851	876	925	25.7	900	925	933	1.3	894	895	897	0.9		2.3	8.1	3.0	6.1
r206	1029	996	1.5	930	954	987	32.3	996	1007	1011	1.1	999	1005	1009	1.2		3.2	7.4	2.1	2.3
r207	1072	1038	2.0	939	986	1022	33.1	1023	1041	1051	1.3	1023	1031	1035	1.5		3.2	8.0	2.9	3.9
r208	1112	1069	1.6	1002	1042	1086	38.7	1057	1074	1075	1.4	1080	1098	1099	1.4		3.9	6.3	3.4	1.3
r209	950	926	2.4	866	897	927	27.1	903	927	943	1.0	920	923	930	1.2		2.5	5.6	2.4	2.8
r210	987	958	1.9	870	915	944	28.2	933	944	946	1.0	960	960	960	0.8		2.9	7.3	4.4	2.7
r211	1046	1023	1.6	934	967	1007	37.6	1002	1003	1003	0.8	1019	1019	1019	1.1		2.2	7.5	4.2	2.6
sum:	**10982**	**10662**	**18.3**	**9852**	**10235**	**10603**	**345**.8	**10465**	**10581**	**10636**	**11.9**	**10573**	**10619**	**10651**	**11.7**	**avg.:**	**2.9**	**6.8**	**3.7**	**3.3**
rc201	795	780	1.0	670	768	794	21.3	787	789	790	0.6	768	770	771	0.6		1.9	3.4	0.7	3.1
rc202	936	882	1.3	822	866	932	20.8	919	919	919	0.7	887	900	905	0.8		5.8	7.5	1.8	3.9
rc203	1003	960	2.7	822	866	932	20.8	953	956	956	1.0	948	963	973	1.2		4.3	7.1	4.7	13.9
rc204	1136	1117	2.3	978	1044	1107	32.2	1123	1123	1123	0.9	1077	1079	1079	1.3		1.7	8.1	1.1	5.1
rc205	859	840	1.0	751	808	837	20.8	842	842	842	0.7	838	842	849	0.9		2.2	5.9	2.0	1.9
rc206	895	860	1.1	819	850	865	20.8	857	859	860	0.7	856	856	856	0.8		3.9	5.0	4.0	4.4
rc207	983	926	1.3	859	896	928	26.7	946	948	950	0.9	916	930	940	1.0		5.8	8.9	3.6	5.4
rc208	1053	1037	2.3	937	976	1032	23.1	1005	1005	1005	0.6	1034	1034	1034	0.8		1.5	7.3	4.6	1.8
sum:	**7660**	**7402**	**13**	**6658**	**7140**	**7427**	**191.5**	**7432**	**7442**	**7445**	**6.2**	**7324**	**7374**	**7407**	**7.4**	**avg.:**	**3.4**	**6.8**	**2.9**	**5.1**

Table 4. Results for Cordeau's test problems (n from 48 to 288).

	BK	ILS		GA				GAPR(I)				GAPR(II)					% gap BK with			
n name	score	score	time	min	avg.	max	time	min	avg.	max	time	min	avg.	max	time		ILS	GA	GAPR(I)	GAPR(II)
48 pr1	*308*	304	0.5	275	298	308	6.4	305	305	305	0.2	305	306	308	0.3		1.3	3.2	1.0	0.6
96 pr2	*404*	385	0.6	370	381	393	10.1	398	400	401	0.4	382	386	386	0.4		4.7	5.7	1.1	4.5
144 pr3	*394*	384	1.0	344	373	392	12.8	393	393	393	0.4	393	393	393	0.6		2.5	5.3	0.3	0.3
192 pr4	*489*	447	1.9	413	449	471	21.1	487	487	487	0.6	470	470	470	0.6		8.6	8.2	0.4	3.9
240 pr5	*595*	576	4.6	521	553	585	33.4	549	559	577	2.0	567	578	587	1.6		3.3	7.1	6.1	2.9
288 pr6	*590*	538	2.5	471	512	568	27.8	567	573	579	1.6	580	582	583	1.6		8.8	13.2	2.9	1.4
72 pr7	*298*	291	0.4	270	282	291	6.0	288	288	288	0.2	289	289	291	0.3		2.3	5.4	3.4	3.0
144 pr8	*463*	463	1.0	410	426	447	14.1	463	463	463	0.4	463	463	463	0.6			8.0		
216 pr9	*493*	461	1.4	422	448	470	24.9	450	451	459	0.9	455	456	458	1.0		6.5	9.1	8.5	7.5
288 pr10	594	539	3.6	502	536	568	33.5	536	556	570	1.8	539	540	541	1.7		9.5	9.8	6.3	9.1
sum:	**4628**	**4388**	**17.5**	**3998**	**4258**	**4491**	**190.2**	**4436**	**4475**	**4522**	**8.6**	**4443**	**4463**	**4483**	**8.7**	**avg.:**	**5.2**	**8.0**	**3.3**	**3.6**
48 pr11	330	330	0.3	310	332	342	7.0	342	344	**345**	0.3	**345**	**345**	**345**	0.2			-0.6	-4.3	-4.5
96 pr12	442	431	0.9	401	417	433	10.7	424	432	434	0.4	437	437	437	0.5		2.5	5.7	2.3	1.1
144 pr13	461	450	1.9	410	429	446	18.8	431	455	457	0.9	451	458	469	1.2		2.4	6.9	1.3	0.6
192 pr14	567	482	1.1	464	489	524	23.1	522	533	535	1.5	510	518	526	1.1		15.0	13.8	5.9	8.6
240 pr15	685	638	5.3	558	611	676	39.6	687	700	**701**	2.4	649	656	663	1.8		6.9	10.8	-2.1	4.2
288 pr16	674	559	4.1	525	564	611	35.6	600	607	609	1.5	605	626	635	2.5		17.1	16.3	10.0	7.1
72 pr17	359	346	0.2	332	349	356	8.4	350	359	**360**	0.3	353	353	353	0.3		3.6	2.8	-0.4	1.7
144 pr18	535	479	0.8	432	468	508	16.3	528	528	528	0.7	536	**536**	536	0.6		10.5	12.5	1.3	-0.2
216 pr19	562	499	2.7	450	501	530	29.1	532	533	535	1.3	531	534	546	1.3		11.2	10.9	5.1	4.9
288 pr20	667	570	2.5	569	590	613	29.1	601	611	612	2.1	623	627	629	2.2		14.5	11.5	8.3	6.0
sum:	**5282**	**4784**	**19.8**	**4451**	**4750**	**5039**	**232.9**	**5017**	**5102**	**5116**	**11.5**	**5040**	**5090**	**5139**	**12.0**	**avg.:**	**9.4**	**10.1**	**3.4**	**3.6**

Parameter Synthesis for Timed Kripke Structures

Extended Abstract

Michał Knapik[1] and Wojciech Penczek[1,2]

[1] Institute of Computer Science, PAS, Warsaw, Poland
[2] University of Natural Sciences and Humanities, II, Siedlce, Poland
{knapik,penczek}@ipipan.waw.pl

Abstract. We show how to synthesise parameter values under which a given property, expressed in a certain extension of CTL called $\mathrm{RTCTL_P}$, holds in a parametric timed Kripke structure. Similarly as in fixed-point symbolic model checking approach, we introduce special operators which stabilise on the solution. The process of stabilisation is essentially a translation from $\mathrm{RTCTL_P}$ parameter synthesis problem to a discrete optimization task. We argue that this leads to new opportunities in model checking, including the use of integer programming and related tools.

1 Introduction

Complex systems, both hardware and software, present in critical areas need to be verified. The best moment for the verification is the design phase, perhaps even before any prototype is developed. This helps to reduce errors and costs; the found flaws can also provide valuable pointers to a designer.

Model checking is one of the established methods for verification of complex, timed, and reactive systems. In this approach, a model for a verified system is built (e.g. a Kripke structure or a Petri net), and a property to be checked is specified in a version of a modal logic (e.g. CTL or TCTL). The pair consisting of a model and a formula is the input for a model checking tool. The output is simply the *property holds* or *property does not hold* answer.

However, such an approach has its drawbacks. In the beginning phases of a system design some of the features required in a model might be unknown (e.g. timing constraints), which forces the designer to substitute them with some guessed or standard values. Even if it is possible to present a full model of the system, there is no guarantee that this specification will not be subject to some changes. Often the minimal alteration of the original model may lead to violation of a checked property, therefore the process of verification has to be repeated.

A system designer using model checking methods would substantially benefit from a tool that is able to accept an underspecified model with some values abstracted as parameters. In this case the expected output consists of a set of parameter valuations under which a given property holds. This approach is called *parametric model checking* or *parameter synthesis*. Parametric model checking

eliminates the needs for guessing and for performing batches of tests for ranges of values.

In this paper we show how to perform parameter synthesis for timed Kripke structures, i.e., Kripke structures where transition is augmented with an additional label specifying how long it takes to traverse it. The input logic is a certain extension of Computation Tree Logic, which allows for expressing properties over the restricted fragments of paths.

1.1 Related Work and Paper Outline

The logic considered in this paper and its models are based on the Real Time Computation Tree Logic (RTCTL) and timed Kripke structures introduced in [1].

As we show, the problem of parameter synthesis is decidable for $\mathrm{RTCTL_P}$. It is however not decidable for even as simple properties as reachability for many other models, e.g. parametric timed automata (PTA) [2, 3] and bounded parametric time Petri nets [4]. Difference bound matrix - based semi-algorithms for reachability were extended to the PTA case in [5] and implemented in UPPAAL-PMC. In [6] we showed how to synthesise by means of bounded model checking a part of the set of valuations for PTA reachability. The problem of synthesis of bounded integer valuations for PTA is analysed in [7] and shown to be in PSPACE. In [8] the authors show how to synthesise the constraints on valuations under which a PTA is *time-abstract* equivalent to some initial one; the work is implemented in IMITATOR prototype tool. Parametric analysis is also possible with HyTech [9] by means of hybrid automata.

In the next section we introduce the $\mathrm{RTCTL_P}$ logic and its models. In Section 3 we show how to solve the synthesis problem via a translation to sets of linear inequalities over natural numbers. We conclude the work with a comment on the possible benefits and downsides of our approach and future plans.

2 Parameterized Temporal Logics

Let $\mathbb{N}$ denote the set of all natural numbers (including 0), and let $P(D)$ denote the power set of a set D. For any sequence $x = (x_1, \ldots, x_n)$ and $0 \leq i \leq n$, let $x|_i = x_i$ be the projection of x on the i–th variable.

2.1 The Syntax of $\mathrm{RTCTL_P}$

The Real Time CTL [1] allows to express branching-time temporal properties involving the integer time-step depth of considered paths.

Definition 1 (Syntax of $\mathrm{RTCTL_P}$). *Let $\mathcal{PV}$ be a set of propositional variables containing the symbol* true. *The formulae of* $\mathrm{RTCTL_P}$ *are defined as follows:*

1. *every member of $\mathcal{PV}$ is a formula,*
2. *if α and β are formulae, then so are $\neg\alpha$, $\alpha \wedge \beta$,*

3. if α and β are formulae, then so are $EX^{\leq k}\alpha$, $EG^{\leq k}\alpha$, $E\alpha U^{\leq k}\beta$ for $k \in \mathbb{N}$.

As to give an example of the meaning of an $\mathrm{RTCTL_P}$ formula, $EG^{\leq 5}p$ states that *"there exists a path such that p holds in each state reached from the beginning in time not greater than* 5*."*

2.2 The Semantics of $\mathbf{RTCTL_P}$

We evaluate the truth of the formulae in the parametric timed Kripke structures. These are standard Kripke structures with the transitions decorated by additional labels interpreted as time variables.

Definition 2. *A parametric timed Kripke structure (a* model*) is a 5-tuple* $\mathrm{M} = (S, s^0, T, \to, \mathcal{L})$ *where:*

- S *is a finite set of* states*,*
- $s^0 \in S$ *is the* initial state*,*
- T *is a set of* time step parameters *(variables),*
- $\to \subseteq S \times T \times S$ *is a transition relation such that for every* $s \in S$ *there exists* $s' \in S$ *and* $t \in T$ *with* $(s, t, s') \in \to$ *(i.e., the relation is total),*
- $\mathcal{L} : S \longrightarrow 2^{\mathcal{PV}}$ *is a valuation function satisfying true* $\in \mathcal{L}(s)$ *for each* $s \in S$*.*

Let s, s' be two states of a model, and let t be a time step parameter. By $s \xrightarrow{t} s'$ we denote that $(s, t, s') \in \to$. The intuitive meaning of $s \xrightarrow{t} s'$ is that it takes t time units to reach s' from s. We define $in(s)$, $out(s)$, $link(s, s')$ as the sets of the labels of the transitions entering s, leaving s, and connecting s with s', respectively. More formally, $in(s) = \{t \in T \mid s' \xrightarrow{t} s \text{ for } s' \in S\}$, $out(s) = \{t \in T \mid s \xrightarrow{t} s' \text{ for } s' \in S\}$, and $link(s, s') = \{t \in T \mid s \xrightarrow{t} s'\}$.

A function $\omega : T \to \mathbb{N}$ is called a *parameter valuation*. The set of all the parameter valuations is denoted by Ω. Consider an infinite sequence $\pi = (s_0, t_0, s_1, t_1, \ldots)$ such that $s_i \in S$ and $s_i \xrightarrow{t_i} s_{i+1}$ for $i \in \mathbb{N}$. By $\pi_i = s_i$ we denote the i–th state of π. We define the *time distance function* between the positions π_0 and π_j on a sequence π as $\delta^j_\pi = \sum_{i=0}^{j-1} t_i$, and we assume that $\delta^0_\pi = 0$. If ω is a parameter valuation, then let $\delta^j_\pi(\omega) = \sum_{i=0}^{j-1} \omega(t_i)$. A sequence π is called an *ω–path* if $lim_{j\to\infty}\delta^j_\pi(\omega) = \infty$, or simply a *path* if ω is evident from the context.

Definition 3 (Semantics of $\mathrm{RTCTL_P}$**).** *Let* $\mathrm{M} = (S, s^0, T, \to, \mathcal{L})$ *be a model and* $s \in S$*. Let* $\alpha, \beta \in \mathrm{RTCTL_P}$*, let* $\omega \in \Omega$ *be a parameter valuation, and* $k \in \mathbb{N}$*.* $M, s \models_\omega \alpha$ *denotes that* α *is true at the state* s *of* M *under the valuation* ω*. (In what follows we omit* M *where it is implicitly understood.) The relation* $\models_\omega$ *is defined inductively as follows:*

1. $s \models_\omega p$ *iff* $p \in \mathcal{L}(s)$*,*
2. $s \models_\omega \neg\alpha$ *iff* $s \not\models_\omega \alpha$*,*
3. $s \models_\omega \alpha \wedge \beta$ *iff* $s \models_\omega \alpha$ *and* $s \models_\omega \beta$*,*
4. $s \models_\omega EX^{\leq k}\alpha$ *iff there exists a path* π *s.t.* $\pi_0 = s, \delta^1_\pi(\omega) \leq k$*, and* $\pi_1 \models_\omega \alpha$*,*

5. $s \models_\omega EG^{\leq k}\alpha$ *iff there exists a path* π *such that* $\pi_0 = s$, *and for all* $i \geq 0$ *if* $\delta^i_\pi(\omega) \leq k$, *then* $\pi_i \models_\omega \alpha$,
6. $s \models_\omega E\alpha U^{\leq k}\beta$ *iff there exists a path* π *such that* $\pi_0 = s$ *and for some* $i \in \mathbb{N}$ *it holds that* $\delta^i_\pi(\omega) \leq k$ *and* $\pi_i \models_\omega \beta$, *and* $\pi_j \models_\omega \alpha$ *for all* $0 \leq j < i$.

The RTCTL$_\mathrm{P}$ logic slightly differs from RTCTL presented in [1]. Firstly, we have omitted the non-superscripted modalities. It is straightforward to extend the logic with these, and to see that the standard fixpoint algorithms for EG and EU verification can be applied with no changes. Secondly, we define the semantics on ω–paths, explicitly requiring the total traversal time to grow to the infinity with the depth of the path. This is consistent with the usual requirement of progressiveness of timed systems.

3 Translation to Linear Algebra

In what follows we fix a model $M = (S, s^0, T, \rightarrow, \mathcal{L})$.

We need several simple notions concerning the sets of statements (called *linear statements*) of the form $c_1t_1 + \ldots + c_nt_n$, where $t_i \in T$ are time step parameters, $c_i \in \mathbb{N}$, and $t_i \neq t_j$ for all $1 \leq i, j \leq n$, $i \neq j$. The set of all linear statements over T is denoted by $\mathcal{LS}_T$; we omit the T subscript if it is implicitly understood. In this paper we consider only finite subsets of $\mathcal{LS}_T$.

Let $\eta = c_1t_1 + \ldots + c_nt_n$, and let $\omega \in \Omega$. We define the application of ω to η as $\eta[\omega] = c_1\omega(t_1) + \ldots + c_n\omega(t_n)$. We also define the k-bounding operation for $k \in \mathbb{N}$ as follows:

$$[\eta]_k := min(c_1, k+1)t_1 + \ldots + min(c_n, k+1)t_n.$$

To show an example, consider the statement $\eta = 6t_1 + 9t_2$ and 5-bounding $[\eta]_5 = min(6,6)t_1 + min(9,6)t_2 = 6t_1 + 6t_2$.

The operation of *k-bounding* has a property such that if $\approx \in \{\leq, <, >, \geq\}$, then for any $k \in \mathbb{N}$ the inequalities $\eta \approx k$ and $[\eta]_k \approx k$ have the same sets of solutions. This can be easily verified on a case-by-case basis, by noticing that if a given coefficient c_i of η exceeds $k+1$, then any nonzero value of t_i makes $\eta \approx k$ true for $\approx \in \{>, \geq\}$, while $\approx \in \{\leq, <\}$ means that only zero can be substituted for t_i.

Previous observation is crucial to the theory, as it means that every set of linear statements over the finite parameter set T, obtained by means of k-bounding with respect to some fixed natural k, is finite. We extend the $[\ ]_k$ operation to subsets $A \subseteq \mathcal{LS}$ as follows:

$$[A]_k = \{[\eta]_k \mid \ \eta \in A\}.$$

Let $A, B \subseteq \mathcal{LS}$, then we define $A + B = \{\eta + \mu \mid \eta \in A \text{ and } \mu \in B\}$.

Now let us consider $A \subseteq \mathcal{LS}$, $k \in \mathbb{N}$, and $\approx \in \{\leq, <, >, \geq\}$. We define $[A]_{\approx k}$ as follows:

$$[A]_{\approx k} = \bigcup_{\eta \in A} \{\omega \mid \eta[\omega] \approx k\}.$$

As to give an example, let $A = \{t_1 + 2t_2, t_3\}$, then $[A]_{<4}$ consists of all the valuations ω such that $\omega(t_1) + 2\omega(t_2) < 4$, or $\omega(t_3) < 4$.

We call the set $S \times P(\Omega)$ the *parametric state space*, and its elements are called the *parametric states*. As to give an example, consider $A \subseteq \mathcal{LS}$ such that $A = \{2t_1 + 3t_2, 2t_1 + 3t_4\}$. The pair of form $(s_0, [A]_{\leq 10})$ is a parametric state.

The last preliminary notion needed in the rest of the paper is the auxiliary operator *Flatten*. Let $B \subseteq S \times P(\Omega)$, then we define:

$$(s, A) \in Flatten(B) \text{ iff } A = \bigcup \{C \mid (s, C) \in B\},\ A \neq \emptyset.$$

To make this definition clearer, consider an example where $B = \{(s_0, C_1), (s_0, C_2), (s_1, C_3), (s_1, C_4), (s_2, C_5)\}$. In this case $Flatten(B) = \{(s_0, C_1 \cup C_2), (s_1, C_3 \cup C_4), (s_2, C_5)\}$.

If $Flatten(B) = B$, then the set B is called *flat*. If B is flat, then by $B(s)$ we denote the *parameter selector*, that is $B(s) = C$ iff $(s, C) \in B$. The parameter selector is a well defined partial function on S.

Algorithm 1 $Synthesize(M, \phi)$

if $\phi = p$ **then**
  return A_p
end if
if $\phi = \neg\alpha$ **then**
  $A_\alpha = Synthesize(M, \alpha)$
  return $\imath A_\alpha$
end if
if $\phi = \alpha \wedge \beta$ **then**
  $A_\alpha = Synthesize(M, \alpha)$
  $A_\beta = Synthesize(M, \beta)$
  return $A_\alpha * A_\beta$
end if
if $\phi = EX^{\leq k}\alpha$ **then**
  $A_\alpha = Synthesize(M, \alpha)$
  return $\mathcal{EX}^{\leq k} A_\alpha$
end if
if $\phi = EG^{\leq k}\alpha$ **then**
  $A_\alpha = Synthesize(M, \alpha)$
  return $\mathcal{EG}^{\leq k} A_\alpha$
end if
if $\phi = E\alpha U^{\leq k}\beta$ **then**
  $A_\alpha = Synthesize(M, \alpha)$
  $A_\beta = Synthesize(M, \beta)$
  return $\mathcal{E} A_\alpha \mathcal{U}^{\leq k} A_\beta$
end if

3.1 The translation

Our aim is to find all the valuations under which a given formula $\phi \in \mathrm{RTCTL_P}$ holds in a model M. In our solution we augment each state s with the set $A_\phi(s)$ of parameter valuations such that $s \models_\omega \phi$ iff $\omega \in A_\phi(s)$. This is done recursively in Algorithm 1, with respect to the formula structure. For each s the set $A_\phi(s)$ can be represented as a finite union of solution sets of a finite number of linear (integer) inequalities. This means that $A_\phi(s)$ has a finite representation for each s, and for this reason we call the method a translation from $\mathrm{RTCTL_P}$ parametric model checking to linear algebraic problem.

Let $p \in \mathcal{PV}$, then $A_p = \{(s, \Omega) \mid p \in \mathcal{L}(s)\}$ is the set of such pairs (s, Ω) that $p \in \mathcal{L}(s)$. Intuitively, A_p contains the pairs consisting of a state in which p holds, together with the full set Ω; this expresses the lack of restrictions on the parameter values. Obviously, A_p is flat.

In the algorithm we use several new operators that are counterparts of propositional connectives and $\mathrm{RTCTL_P}$ modalities:

1. operator $*$ – a counterpart of $\wedge$,
2. operator $\imath$ – related to $\neg$,
3. operator $\mathcal{EX}^{\leq k}$ – a counterpart of $EX^{\leq k}$,
4. operator $\mathcal{EG}^{\leq k}$ – a counterpart of $EG^{\leq k}$.
5. operator $\mathcal{EU}^{\leq k}$ – related to $EU^{\leq k}$.

The detailed description of these notions is a subject of the rest of this section, starting with the $*$ operator.

Definition 4. *Let A, B be two flat subsets of $S \times P(\Omega)$. Define:*

$$A * B = \{(s, C \cap C') \mid (s, C) \in A, \text{ and } (s, C') \in B\}.$$

The next corollary follows immediately from the above definition.

Corollary 1. *Let ϕ, ψ be $\mathrm{RTCTL_P}$ formulae, and A_ϕ, A_ψ be such flat subsets of the parametric state space that $s \models_\omega \phi$ iff $\omega \in A_\phi(s)$ and $s \models_\omega \psi$ iff $\omega \in A_\psi(s)$ for all $s \in S$. Then $s \models_\omega \phi \wedge \psi$ iff $\omega \in (A_\phi * A_\psi)(s)$.*

It should be noted that in our applications, the $*$ operation is purely symbolic, as we deal with the sets of inequalities only.

Example 1. Consider the following sets:

$$A_\phi = \{(s_0, \Omega), (s_1, \{\omega \mid \omega(t_1) + 3\omega(t_2) < 5\})\},$$
$$A_\psi = \{(s_1, \{\omega \mid 2\omega(t_1) + 3\omega(t_3) < 4\})\},$$

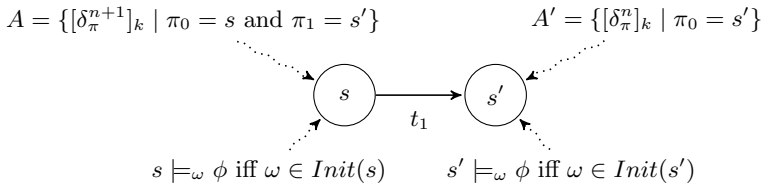
We have $A_\phi * A_\psi = \{(s_1, \{\omega \mid \omega(t_1) + 3\omega(t_2) < 5 \ \wedge \ 2\omega(t_1) + 3\omega(t_3) < 4\})\}$.

In the translation of $EG^{\leq k}$ and $EU^{\leq k}$ we make use of the *bounded backstep operation*. This operation is defined on sets of triples (s, A, C), where s is a state, A is a set of linear statements used to track possible constraints on parameters, and C is a set of parameter valuations used to track the allowed values of time step parameters.

Definition 5. *Let $D \subseteq S \times P(\mathcal{LS}) \times P(\Omega)$, $k \in \mathbb{N}$, and $Init$ be a flat subset of $S \times P(\Omega)$ such that for each $e \in D$ there is $f \in Init$ satisfying $e|_1 = f|_1$. Now, $(s, A, C) \in BackStep_k(D, Init)$ iff:*

1. *there exists $e \in D$ such that $e|_1 = s$,*
2. *for some $A' \subseteq \mathcal{LS}$, $C' \subseteq \Omega$, and $s' \in S$ there exists $(s', A', C') \in D$, such that:*
   (a) *the set $link(s, s')$ of time step parameters (treated as linear statements) is nonempty (i.e. there is a transition from s to s'),*
   (b) *$A = [link(s, s') + A']_k$,*
   (c) *$C = C' \cap Init(s)$.*

While the bounded backstep operation may seem involved, it originates from a natural idea. Let ϕ be some property and let $Init$ be such a set that $s \models_\omega \phi$ iff $\omega \in Init(s)$ for each state s. Let $D \subseteq S \times P(\mathcal{LS}) \times P(\Omega)$ and $(s', A', C') \in D$.



Assume that $C' = Init(s')$, let $n \in \mathbb{N}$, and A' be the set of k–bounded time distance functions for all paths leaving s' and measuring the distance up to the n–th position. It is easy to see, that $BackStep_k(D, Init)$ contains a tuple $(s, A, Init(s) \cap Init(s'))$, where $A = [link(s, s') + A']_k$. The set A consists of k–bounded time distance functions for all paths leaving s, entering s' in the next step, and measuring the distance up to the $(n+1)$–th position. The set $Init(s) \cap Init(s')$ contains such parameter valuations ω that $s \models_\omega \phi$ and $s' \models_\omega \phi$.

Example 2. Consider the sets:

$$C_1 = \{\omega \mid \omega(t_1) > 2\},\ C_2 = \{\omega \mid \omega(t_2) + \omega(t_3) \leq 4\},$$
$$D = \{(s_1, \{6t_1 + 8t_2\}, C_1), (s_2, \{4t_2 + 7t_3, t_4\}, C_2)\},$$

and assume that the only transitions involving s_1 and s_2 are $(s_1, t_1, s_2), (s_1, t_2, s_2)$, and let $Init = \{(s_1, C_1), (s_2, C_2)\}$. Let us compute $BackStep_5(D, Init)$. We can see that $link(s_1, s_2) = \{t_1, t_2\}$, $link(s_1, s_1) = link(s_2, s_2) = link(s_2, s_1) = \emptyset$. Let $A = [\{t_1, t_2\} + \{4t_2 + 7t_3, t_4\}]_5 = \{t_1 + 4t_2 + 6t_3, 5t_2 + 6t_3, t_1 + t_4, t_2 + t_4\}$, and $C = C_2 \cap Init(s_1) = C_2 \cap C_1 = \{\omega \mid \omega(t_1) > 2 \text{ and } \omega(t_2) + \omega(t_3) \leq 4\}$. In this case $BackStep_5(D, Init) = \{(s_1, A, C)\}$.

We say that a sequence of sets $H_0, H_1, \ldots$ *stabilizes* if there exists $i \geq 0$ such that $H_j = H_i$ for all $j > i$, and denote this as $lim_{j \to \infty} H_j = H_i$.

Let D be a finite subset of $S \times P(\mathcal{LS}) \times P(\Omega)$. Notice that if we fix some $k \in \mathbb{N}$ and $Init$, then the sequence defined by $H_0 = D$, and $H_{i+1} = H_i \cup BackStep_k(H_i, Init)$ stabilizes. This is due to the fact that there is a finite

number of time parameters in a model (therefore a finite number of k-bounded expressions built with respect to $[\,]_k$), and a finite number of parameter valuation sets in D.

Let $(s, A, C) \in S \times P(\mathcal{LS}) \times P(\Omega)$, $\approx \in \{\leq, <, >, \geq\}$, and $k \in \mathbb{N}$. Denote $[(s, A, C)]_{\approx k} = (s, [A]_{\approx k} \cap C)$. Intuitively, this encodes a state together with those parameter valuations which satisfy constraints present in $[A]_{\approx k}$ (the path length constraints), and in C (the initial constraints). We extend this notion to the space on which $BackStep$ operates, by putting $[D]_{\approx k} = \{[(s, A, C)]_{\approx k} \mid (s, A, C) \in D\}$ for any $D \subseteq S \times P(\mathcal{LS}) \times P(\Omega)$.

Let us move to the first application of $BackStep_k$ operation, i.e., the translation of $EG^{\leq k}$. The following example provides some intuitions behind the parametric counterpart of this modality.

Example 3. Consider model shown in Fig. 1, where $\mathcal{L}(s_0) = \mathcal{L}(s_1) = \{p\}$, and formula $EG^{\leq 2}p$. For the simplicity, the loops on states s_2, s_3 are unlabeled.

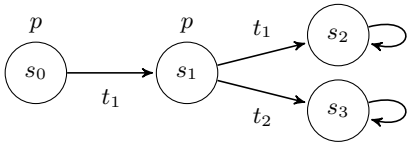


Fig. 1: A simple model

It is easy to see that $s_1 \models_\omega EG^{\leq 2}p$ iff $\omega(t_1) > 2$ or $\omega(t_2) > 2$, i.e., using the newly introduced notation, $\omega \in [out(s_1)]_{>2}$. It also holds that $s_0 \models_\omega EG^{\leq 2}p$ if $\omega \in [out(s_0)]_{>2}$, but this is not an exhaustive description of all such parameter valuations. Indeed, $s_0 \models_\omega EG^{\leq 2}p$ also if $2\omega(t_1) > 2$ or $\omega(t_1) + \omega(t_2) > 2$, i.e., $\omega \in [t_1 + out(s_1)]_{>2}$. By a straightforward case-by-case analysis we can check that $s_0 \models_\omega EG^{\leq 2}p$ iff $\omega \in [out(s_0)]_{>2} \cup [t_1 + out(s_1)]_{>2}$.

Definition 6. *Let A be a flat subset of $S \times P(\Omega)$ and $k \in \mathbb{N}$. Define:*

$$G_0(A) = \{(s, out(s), A(s)) \mid \text{there exists } e \in A \text{ such that } e|_1 = s\},$$
$$G_{j+1}(A) = BackStep_k(G_j(A), A).$$

We define $\mathcal{EG}^{\leq k}A = Flatten(\bigcup_{j=0}^{\infty}[G_j(A)]_{>k})$.

The $Flatten$ operator is used only in order to obtain the result in a less complex form, where for each state s there exists at most one $e \in \mathcal{EG}^{\leq k}A$ such that $e|_1 = s$.

Theorem 1. *Let ϕ be a formula of* $\mathrm{RTCTL_P}$*, and A_ϕ be such a flat subset of $S \times P(\Omega)$ that $s \models_\omega \phi$ iff $\omega \in A_\phi(s)$. For any state $s \in S$, $k \in \mathbb{N}$, and a parameter valuation ω we have $s \models_\omega EG^{\leq k}\phi$ iff $\omega \in (\mathcal{EG}^{\leq k}A_\phi)(s)$.*

Proof. If $s \models_\omega EG^{\leq k}\phi$, then there exists a path $\pi = (s_0, t_0, s_1, t_1, \ldots)$, such that for some $n \in \mathbb{N}$ it holds that $\pi_0 = s$, $\delta_\pi^{n+1}(\omega) > k$ and $\delta_\pi^i(\omega) \leq k$ for all

$0 \leq i \leq n$, and $\pi_i \models_\omega \phi$ for all $0 \leq i \leq n$.

$$\pi = \overbrace{\underbrace{s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} s_n}_{\delta^n_\pi(\omega) \leq k} \xrightarrow{t_n} s_{n+1}}^{\delta^{n+1}_\pi(\omega) > k} \xrightarrow{t_{n+1}} \ldots$$

For each $0 \leq i \leq n$ we have that $\pi_i \models_\omega \phi$, therefore $A_\phi(s_i)$ is well defined for each $0 \leq i \leq n$, and $\omega \in \bigcap_{i=0}^n A_\phi(s_i)$. It is easy to see that $(s_n, out(s_n), A_\phi(s_n)) \in G_0(A_\phi)$, and $t_n \in out(s_n)$. Notice that $s_{n-1} \xrightarrow{t_{n-1}} s_n$, thus $(s_{n-1}, [link(s_{n-1}, s_n) + out(s_n)]_k, A_\phi(s_{n-1}) \cap A_\phi(s_n)) \in BackStep_k(G_0(A_\phi), A_\phi) = G_1(A_\phi)$. Again, we have that $[t_{n-1} + t_n]_k \in [link(s_{n-1}, s_n) + out(s_n)]_k$. After $n+1$ such inductive steps we obtain that there is a tuple $(s_0, A, \bigcap_{i=0}^n A_\phi(s_i)) \in G_n(A_\phi)$ such that $[t_0 + t_1 + \ldots + t_n]_k \in A$, and $\omega \in \bigcap_{i=0}^n A_\phi(s_i)$. Recall that $\delta^n_\pi = t_0 + t_1 + \ldots + t_n$, and as $\delta^n_\pi(\omega) > k$, we have that $[t_0 + t_1 + \ldots + t_n]_k(\omega) > k$, therefore $\omega \in [A]_{>k}$. This means that $\omega \in [A]_{>k} \cap \bigcap_{i=0}^n A_\phi(s_i)$, which in view of the fact that $[(s, A, \bigcap_{i=0}^n A_\phi(s_i))]_{>k} \in [G_n(A_\phi)]_{>k}$ concludes this part of the proof.

Now let $\omega \in (\mathcal{EG}^{\leq k} A_\phi)(s)$. This means that for some $m \in \mathbb{N}$, and $e_m = (s_m, B_m)$, where $s_m = s$ we have that $e_m \in [G_m(A_\phi)]_{>k}$, and $\omega \in B_m$. This in turn means that there is a sequence $(s_0, A_0, C_0), (s_1, A_1, C_1), \ldots, (s_m, A_m, C_m)$ such that:

1. $A_i = [link(s_i, s_{i-1}) + A_{i-1}]_k$ for all $0 < i \leq m$, and $A_0 = out(s_0)$,
2. $C_i = \bigcap_{j=0}^i A_\phi(s_j)$ and $\omega \in C_i$ for all $0 \leq i \leq m$,
3. $(s_i, A_i, C_i) \in G_i(A_\phi)$ for all $0 \leq i \leq m$,
4. $[A_n]_{>k} \cap C_m = B_m$.

From the above points it follows that there exists such a finite sequence $\pi' = (s_m, t_m, s_{m-1}, t_{m-1} \ldots, s_0, t_0)$ that $[\delta^m_{\pi'}]_k = [t_m + t_{m-1} + \ldots + t_0]_k \in A_m$, and $[\delta^m_{\pi'}]_k(\omega) > k$. Notice that the latter is equivalent to $\delta^m_{\pi'}(\omega) > k$, and that the second point implies that $s_i \models_\omega \phi$ for all $0 \leq i \leq m$. The sequence π' is a prefix of some infinite path π (due to the totality of the transition relation), such that $\pi_i \models_\omega \phi$ for all $0 \leq i \leq m$, and $\delta^m_\pi(\omega) > k$. This means that $s \models_\omega EG^{\leq k}\phi$, which concludes the proof. □

Definition 7. *Let A, B be two flat subsets of $S \times P(\Omega)$ and $k \in \mathbb{N}$. Denote:*

$$\begin{aligned} H_0(A,B) &= \{(s, link(s,s'), A(s) \cap B(s')) \mid \text{there exists } e \in B,\ e|_1 = s', \\ &\qquad \text{and } link(s,s') \neq \emptyset\}, \\ H_{i+1}(A,B) &= BackStep_k(H_i(A,B), A). \end{aligned}$$

We define $\mathcal{EAU}^{\leq k} B = Flatten((\bigcup_{i=0}^\infty [H_i(A,B)]_{\leq k}) \cup B)$.

Again, the *Flatten* operator is used only for the convenience, and the sequence $(\bigcup_{i=0}^j H_i)_{j \geq 0}$ is guaranteed to stabilize.

Theorem 2. *Let ϕ, ψ be* RTCTL$_{\mathrm{P}}$ *formulae, and A_ϕ, A_ψ be such flat subsets of parametric state space that $s \models_\omega \phi$ iff $\omega \in A_\phi(s)$ and $s \models_\omega \psi$ iff $\omega \in A_\psi(s)$, for each state s. For any state s, any $k \in \mathbb{N}$, and parameter valuation ω it holds that $s \models_\omega E\phi U^{\leq k}\psi$ iff $\omega \in (\mathcal{E}A_\phi \mathcal{U}^{\leq k} A_\psi)(s)$.*

Proof. Assume that $s \models_\omega E\phi U^{\leq k}\psi$. This means that there exists a sequence $\pi = (s_0, t_0, s_1, t_1, \ldots, s_n, t_n, \ldots)$ such that $\pi_0 = s$, for some $n \geq 0$ we have $\delta^n_\pi(\omega) \leq k$, $\pi_n \models_\omega \psi$, and $\pi_i \models_\omega \phi$ for all $0 \leq i < n$. If $n = 0$, then $s \models_\omega \psi$, therefore $\omega \in A_\psi(s)$; now it suffices to notice that A_ψ is a (flattened) subset of $\mathcal{E}A_\phi \mathcal{U}^{\leq k} A_\psi$. We can therefore assume that $n > 0$, which means that $s_{n-1} \models_\omega \phi$, and $s_n \models_\omega \psi$, thus $\omega \in A_\phi(s_{n-1}) \cap A_\psi(s_n)$. As $t_{n-1} \in link(s_{n-1}, s_n)$, we obtain that $\big(s_{n-1}, link(s_{n-1}, s_n), (A_\phi(s_{n-1}) \cap A_\psi(s_n))\big) \in H_0(A_\phi, A_\psi)$. Similarly as in a first part of the proof of Theorem 1 we can now create a sequence $(s_0, A_0, C_0), (s_1, A_1, C_1), \ldots, (s_{n-1}, A_{n-1}, C_{n-1})$ such that for all $0 \leq i \leq n-1$:

1. $A_i = [link(s_i, s_{i+1}) + link(s_{i+1}, s_{i+2}) + \ldots + link(s_{n-1}, s_n)]_k$,
2. $C_i = \bigcap_{j=i}^{n-1} A_\phi(s_j) \cap A_\psi(s_n)$ and $\omega \in C_i$,
3. $(s_i, A_i, C_i) \in H_{n-i-1}(A_\phi, A_\psi)$.

Now let us notice that $[t_0 + t_1 + \ldots + t_{n-1}]_k \in A_0$, and as $\delta^n_\pi(\omega) \leq k$, also $[t_0 + t_1 + \ldots + t_{n-1}]_k(\omega) \leq k$. This means that $\omega \in [A_0]_{\leq k} \cap C_0$, therefore there is $e \in [H_0(A_\phi, A_\psi)]_{\leq k}$ such that $e|_1 = s_0 = s$, and $\omega \in e|_2$, which concludes the case.

Now let us assume that $\omega \in (\mathcal{E}A_\phi \mathcal{U}^{\leq k} A_\psi)(s)$. If $\omega \in A_\psi(s)$, then obviously $s \models_\omega \psi$ and $s \models_\omega E\phi U^{\leq k}\psi$, therefore let us assume that for some $m \in \mathbb{N}$ we have that $e = (s_m, B_m) \in [H_m(A_\phi), A_\psi]_{\leq k}$ where $s_m = s$, and $\omega \in B_n$. Again, this means that there exist a state s' such that $\omega \in A_\psi(s')$, and a sequence $(s_0, A_0, C_0), (s_1, A_1, C_1), \ldots, (s_m, A_m, C_m)$ such that:

1. $link(s_{i+1}, s_i) \neq \emptyset$ for all $0 \leq i < m$, and $link(s_0, s') \neq \emptyset$,
2. $A_i = [link(s_i, s_{i-1}) + link(s_{i-1}, s_{i-2}) + \ldots + link(s_0, s')]_k$ for all $0 \leq i \leq m$,
3. $C_i = \bigcap_{j=0}^{i} A_\phi(s_j) \cap A_\psi(s')$ and $\omega \in C_i$ for all $0 \leq i \leq m$,
4. $(s_i, A_i, C_i) \in H_i(A_\phi, A_\psi)$ for all $0 \leq i \leq m$,
5. $[A_m]_{\leq k} \cap C_m = B_m$.

From the above points we can infer the existence of such a finite sequence $\pi' = (s_m, t_m, s_{m-1}, t_{m-1}, \ldots, s_0, t_0, s', t')$ (the t' is an arbitrary time step parameter from $out(s')$) that:

1. $t_i \in link(s_i, s_{i-1})$ for all $0 < i \leq m$, and $t' \in link(s_0, s')$,
2. $\pi'(i) \models_\omega \phi$ for all $0 \leq i \leq m$, and $\pi'(m+1) \models_\omega \psi$,
3. $\delta^m_{\pi'}(\omega) \leq k$, as $[\delta^m_{\pi'}]_k(\omega) = [t_0 + t_1 + \ldots + t_m]_k(\omega) \leq k$.

By the virtue of the totality of the transition relation this means that $s \models_\omega E\phi U^{\leq k}\psi$, which concludes the proof. □

Definition 8. *Let A be a flat subset of $S \times P(\Omega)$, and $k \in \mathbb{N}$. Denote:*

$$I_k(A) = \{(s, link(s, s'), A(s')) \mid \text{exists } e \in A \text{ s. t. } e|_1 = s' \text{ and } link(s, s') \neq \emptyset\}.$$

We define $\mathcal{EX}^{\leq k} A = Flatten([I_k(A)]_{\leq k})$.

Intuitively, in $I_k(A)$ for each state s we gather its connections with other states s' and constraints $A(s')$ imposed in s'. It suffices to ensure that these constraints are consistent with conditions of transition from s to s' in under k time units.

Corollary 2. *Let ϕ be a formula of* RTCTL$_\mathrm{P}$*, let $k \in \mathbb{N}$, and let A_ϕ be such a flat subset of $S \times P(\Omega)$ that $s \models_\omega \phi$* iff *$\omega \in A_\phi(s)$. For any state s and parameter valuation ω we have $s \models_\omega EX^{\leq k}\phi$ iff $\omega \in (\mathcal{EX}^{\leq k} A_\phi)(s)$.*

We have proved that the proposed translation is valid for all nonnegated expression. To complete the theory we show how to deal with negations.

Definition 9. *Let A be a flat subset of $S \times P(\Omega)$. We define:*

$$\begin{aligned} \imath A = Flatten(\{(s, \Omega \setminus A(s)) \mid \text{exists } e \in A \text{ such that } e|_1 = s\} \\ \cup \{(s, \Omega) \mid \text{there is no } e \in A \text{ such that } e|_1 = s\}). \end{aligned}$$

Let us present some intuitions concerning the translation of the negation. Let A_ϕ characterize the states augmented with parameter valuations under which the ϕ property holds. The $\imath A_\phi$ set is built by:

1. augmenting any state s represented in A_ϕ, by those valuations under which ϕ does not hold (the complement of $A_\phi(s)$),
2. including all the states which are not represented in A_ϕ together with the full set of parameter valuations.

This gives rise to the following corollary.

Corollary 3. *Let A_ϕ be such a flat subset of $S \times P(\Omega)$ that $s \models_\omega \phi$* iff *$\omega \in A_\phi(s)$. For any state s and $\omega \in \Omega$ it holds that $s \models_\omega \neg\phi$ iff $\omega \in (\imath A_\phi)(s)$.*

4 Conclusions

The method presented in this paper allows for the synthesis of parameter values in timed Kripke structures for properties expressed in RTCTL$_\mathrm{P}$ logic. To be more precise, for a given property ϕ the result of synthesis is the set A_ϕ of constraints on time step parameters. These constraints are expressed as linear inequalities over natural numbers, therefore our method is in fact a translation from the problem of RTCTL$_\mathrm{P}$ parameter synthesis to a problem stated in the language of linear algebra. If properly implemented, this enables to take advantage of the vast work and available tools from the discrete optimization field.

It is rather straightforward to show that for a given RTCTL$_\mathrm{P}$ formula ϕ it suffices to consider only the parameter step values which do not exceed the greatest superscript in ϕ plus 1. While Ω can be limited to a finite set, an enumerative verification of all possible valuations from this set would soon prove to be intractable. A symbolic model checking approach gives a chance of alleviating these limitations via an efficient representation of statespace and operations on its subsets. We plan to research the possibilities of implementing the presented work using various versions of decision diagrams and SMT-theories.

Acknowledgements Michał Knapik is supported by the Foundation for Polish Science under International PhD Projects in Intelligent Computing. Project financed from the European Union within the Innovative Economy Operational Programme 2007-2013 and European Regional Development Fund.

References

1. Emerson, E.A., Trefler, R.: Parametric quantitative temporal reasoning. In: Proc. of the 14th Symp. on Logic in Computer Science (LICS'99), IEEE Computer Society (July 1999) 336–343
2. Alur, R., Henzinger, T., Vardi, M.: Parametric real-time reasoning. In: Proc. of the 25th Ann. Symp. on Theory of Computing (STOC'93), ACM (1993) 592–601
3. Doyen, L.: Robust parametric reachability for timed automata. Inf. Process. Lett. **102** (May 2007) 208–213
4. Tranouez, L.M., Lime, D., Roux, O.H.: Parametric model checking of time Petri nets with stopwatches using the state-class graph. In: Proc. of the 6th Int. Workshop on Formal Analysis and Modeling of Timed Systems (FORMATS'08). Volume 5215 of LNCS., Springer-Verlag (2008) 280–294
5. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.: Linear parametric model checking of timed automata. In: Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01). Volume 2031 of LNCS., Springer-Verlag (2001) 189–203
6. Knapik, M., Penczek, W.: Bounded model checking for parametric timed automata. T. Petri Nets and Other Models of Concurrency **5** (2012) 141–159
7. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for timed automata. In: Proceedings of the 19th international conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'13, Berlin, Heidelberg, Springer-Verlag (2013) 401–415
8. André, E., Chatain, T., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. International Journal of Foundations of Computer Science **20**(5) (Oct 2009) 819–836
9. Henzinger, T., Ho, P., Wong-Toi, H.: HyTech: A model checker for hybrid systems. In: Proc. of the 9th Int. Conf. on Computer Aided Verification (CAV'97). Volume 1254 of LNCS., Springer-Verlag (1997) 460–463

Voronoi Based Strategic Positioning for Robot Soccer

S. Kaden, H. Mellmann, M. Scheunemann, and H.-D. Burkhard

Department of Computer Science, Cognitive Robotics Group
Humboldt-Universität zu Berlin, Germany
{kaden,mellmann,scheunem,hdb}@informatik.hu-berlin.de

Abstract. Strategic positioning is a decisive part of the team play within a soccer game. In most solutions the positioning techniques are treated as a constituent of a complete team play strategy. In a comprehensive overview we discuss the team play and positioning methods used within RoboCup and extract the essential requirements for player positioning. In this work, we propose an approach for strategic positioning allowing for flexible formulation of arbitrary strategies. Based on the conditions of a specific strategy, the field is subdivided in regions by a Voronoi tessellation and each region is assigned a weight. Those weights influence the calculation of the optimal robot position as well as the path. A team play strategy can be expressed by the choice of the tessellation as well as the choice of the weights. This provides a powerful abstraction layer simplifying the design of the actual play strategy. We also present an implementation of an example strategy based on this approach and analyze the performance of our approach in simulation.

1 Introduction

With the advancement of RoboCup, team play turns into a key feature of success. While in simulation leagues a strong team play is a decisive aspect since the very beginning, in hardware leagues more basic aspects like motion, perception and modeling usually yield more overall improvement. A central part of team play is the *strategic positioning* of players on the field, i.e., the placement of robots not in possession of the ball. Hereby, two core issues have to be addressed: where does the player have to go to and how does he get there. The answer, depends among others on numerous factors, involving player positions on the field, ball position, distribution of team-mates and opponent players and so on. In this paper we present an approach which allows to formulate all the aspects necessary for the positioning and mechanisms to derive the target position and the path to it.

To achieve that, the whole field is separated into weighted regions. To obtain a practical separation, Voronoi tessellation is employed due to its convex structure and embedded neighboring relation. Each region is evaluated according to some criteria. The weights encode a variety of aspects, e.g. obstacles, free space, the probability that the ball is contained or the dominant regions of players. Based

on those weights, an optimal region can be determined, as well as a feasible path to this region.

For illustration and testing purposes we utilize a supporter example strategy to prove the functionality of our approach. Beside a simple heuristic strategy for positioning, a scalar field is exploited for strategy description and weight calculation for each region.

The residual paper is divided into four main parts. The next section offers a comprehensive overview of strategic positioning approaches proposed within the RoboCup. The description of the VBSM algorithm is given in section 3. In section 4 we discuss the experiments and their results. At the end we present a conclusion and provide an outlook for future work.

2 Related Work

The problem of the strategic positioning within the RoboCup context has been widely investigated in the past years. The ideas vary a lot and there seems to be no *standard approach*. Some major differences are surely reflections of the characteristics exposed by the different leagues. In general, the solutions within the simulation leagues are more elaborated, assume a much more rich and reliable world model and are more computationally complex in comparison to the solutions of the hardware leagues. In most cases the positioning issue is not solved separately, it is mostly part of an entire team play solution. In the following overview our main focus is rather on the positioning ideas than on the high-level parts necessary to organize a team like role assignment.

One of the most direct ways to position a robot is to calculate its target position as a geometric relation to its world model, which results in a reactive behavior. Although very simple, those techniques proved to be quite effective and robust to noisy data. An example for such positioning can be found in [1] by Carlos E. Agüero et al. which targeted the issue of role switching and role positions within the *Four Legged League* (4LL). Here the defender should occupy a point on the line between the goal and the ball to prevent the opponent attacker most effectively from scoring a goal. A supporter chooses the center of the rectangle stretched by the ball and the most distant opponent corner. Another example from 4LL is given by Phillips and Veloso, in [15]. They present reactive supporter strategies. Here the field is subdivided in rough fixed regions, such like offensive, defensive etc., which allows to choose different strategies depending on which region the ball is in. For instance, when the ball is in the offense region, the supporter covers a point left or right in a fixed distance beside the ball. Is the ball situated in the defense region, it is followed by the supporter in one axis which stays in the offense region. In addition, the supporter chooses a corner of the opponent penalty area if the ball is close to the opponent goal to catch the ball if it rebounds. In the *Humanoid League* (HL), K. Petersen, G. Stoll and O. von Stryk [14] developed another reactive behavior for a supporter. Similar to the previous approach, the supporter chooses a position relative to the ball. Thereby the relation may be adjusted depending on the game situation.

Potential fields are another very popular tool for positioning. A potential field is a function which assigns a direction to each point on the field and is usually formulated as the negative gradient of a scalar field. To navigate the robot can simply follow the direction of the field at its current position. This approach may represent the world state, e.g., obstacles are represented as maxima (*repeller*), as well as strategy, e.g., the target position is formulated as a minimum (*attractor*). Potential fields are prone to local minima, which is a minor issue within dynamic situations. They can adapt very smoothly in dynamic scenarios and can be resistant to noise. These properties make them a very tempting tool for dynamic planning scenarios with noisy data. A good example is presented by the team B-Human in [17](SPL). Here, the team play is organized in three fixed *formations* which are activated based on the state of the game, e.g., goal score. A formation defines a role for each player. A specific role defines the actual player position on the field, e.g., the target supporter position is calculated based on the position of the ball and the striker. This target position as well as the current game situation is formulated by a potential field, which is used for navigation. Thereby, the target position is formulated as an attractor whereas the other players, the own penalty area as well as the line between the ball and the opponent goal are formulated as repellers. This way the robot avoids obstacles and forbidden areas on its way to the target position. A very similar approach is presented by Work, Chown, Hermans, Butterfield and McGranaghan in [18](4LL). The formations are additionally organized in strategies and each role splits in a number of sub roles. A sub role is activated depending on the ball position on the field and defines the target position of the player. The positioning itself is also based on the potential fields and is formulated in a similar way to the Team B-Human approach.

In their work [13] (SPL) Nieuwenhuisen, Steffens, and Behnke consider the positioning of a robot as a path planning problem. Hereby, the task of approaching the ball while avoiding obstacles is directly addressed. A multi resolution occupancy grid in egocentric Cartesian and Log-Polar coordinates is used to model the obstacles. The resolution of the grid is higher in the proximity of the robot and the path is determined by the A* search. B-Human use in [17](SPL) an *extended bidirectional Rapidly-Exploring Random Tree* to estimate the path to the ball, which doesn't require discretization of the space.

Another way is to formulate the positioning task as an optimization problem where the world state and the strategy are encoded as conditions to be satisfied. In their work Kyrylov, Razykov and Hou [7,8](S2D) subdivide the field in a grid. Each cell is then rated according to certain criteria. For instance, the player should be open for a direct pass, the distance to opponent players should be maximized and the distance to the reference position defined by the strategy should be minimal. These criteria, e.g., the reference position, are defined by the formation and the roles of the players. The target position is then determined as a Pareto-Optimum based on those criteria. Both publications deal with different scenarios (offensive, defensive) and discuss different criteria for the optimization.

Voronoi diagram and its dual *Delaunay triangulation* state another popular tool used for the player positioning. Hidehisa Akiyama and Itsuki Noda [2](S2D) use in their work a Delaunay triangulation of the field to encode the positioning of the robots depending on the ball position. To construct such triangulation, a representative set of possible ball positions and the corresponding positions of the players are predefined. Those ball positions define the nodes of the triangulation. During the game the actual positions of the players are determined as an interpolation between the positions provided by the nodes of the triangle in which the ball is located. Another way to use Voronoi diagrams for positioning is presented by Hesam Addin Dashti et al. in [4](S2D). Thereby, the actual positions of the robots define the Voronoi cells, i.e., are the Voronoi sites. The repulsing and attracting properties of the objects on the playing field, e.g., ball, opponents, goal, etc., are modeled as forces which affect the agent and are represented by vectors. The vector to the center of gravity of the agent's Voronoi cell provides an additional alignment vector. Thus the player try to relax the Voronoi diagram and to keep as much distance between each other as possible, which results in a good field coverage. In [3](S2D) the technique is extended by including the opponent players as additional cells into the diagram.

Considering the time which is needed to reach a point instead of the Euclidean distance to it provides another kind of regions which are called *dominant regions* (DR). Figuratively speaking, a DR is defined as the area on the field which can be reached by a particular player before the others. In general, calculation of dominant regions requires a good motion model of the players which may be especially a problem for opponent robots. In [12](SSL) Nakanishi, Murakami and Naruse introduce the notion of a *dominant polygon* which essentially is an approximation of the area which can be reached by the robot within a fixed given time limit. Thereby a quadratic motion model for the players is used. Those polygons are used on one hand to estimate the dominant regions for all the player on the field and on the other hand to determine a good position for a pass. To achieve a good passing position the robot essentially tries to leave the dominant polygons of the opponents which may interfere. Another example is presented in [11](SSL) by Nakanishi, Bruce, Murakami, Naruse and Veloso where the dominant regions are used to plan pass combination. E.g., if a pass would lead through an opponent region, a third player is moved in between to close the gap and the pass is performed indirectly (1-2-3 shoot). The border between the dominant regions of a pair of robots is calculated analytically assuming a simple quadratic motion model . Calculating those borders pairwise for all the players on the field leads to a full DR-diagram.

Colin McMillen and Manuela Veloso present in [10] (4LL) a different approach based on plans. A *Play* is a plan which assigns roles for each of the players. A role consists of a predefined *responsibility region* and a behavior strategy for each of the three cases: the ball is outside or inside of the region, or the ball is not seen. A particular play is applied when the game situation satisfies some predefined requirements. When several plays are applicable, the one with the higher weight is selected. The corresponding requirements are based on the game state like

remaining game time, count of players and the actual score. Another scenario based approach is *Scenario-Based Team working* (SBT) [16] (S2D) by Ali Ajdari Rad, Navid Qaragozlou and Maryam Zaheri. A scenario contains a sequence of sub-plans consisting of actions for each robot. Furthermore, it is equipped with a *goal*, e.g., shooting a goal or conquer the ball, triggering conditions, expected costs etc.. During the execution each agent tries to satisfy its iterative sub-plan, which defines the agent's actions. To position the players the field is subdivided in regions. The scenarios are structured in a directed graph, where the nodes are particular scenarios and edges are weighted with a probability for the next scenario to be executed. This graph is created before the game by connecting the scenarios depending on their goals and triggers. The weights of the edges are adjusted during the game depending on the success of a scenario.

The *Situation Based Strategic Positioning* (SBSP) described by Luís Paulo Reis, Nuno Lau and Eugénio Costa Oliveira adapts concepts of human team strategies implemented in the simulated domain by *FC Portugal* [6] and partly used by the team *CAMBADA* within the Middle Size League (MSL) [9]. The team strategy in SBSP consists mainly of a set of tactics, tactic activation rules and roles. A tactic itself consists of predefined plans and team formations. The formations describe the positioning of a player inside a formation. The positioning consists of a reference position, a predefined fixed region, as well as behavior for the cases when the ball is inside or outside of this region. To position the robot the reference position is adjusted with respect to the ball.

3 Voronoi Based Situation Map

Based on the analysis of the related work, the most common approach to define a team strategy is defined by three layers: formations assigning each player a role; roles defining the robots positioning on the field and its behavior; and at last the positioning method which determines the actual movements of the robot. In most cases, the positioning itself (the lowest layer) is implemented directly as a kind of a geometric relation to dynamic and static objects on the field, e.g., relative to the ball, a free position based on DR etc; or is formulated as forces applied to alter the position, e.g., potential fields, alignment towards the Voronoi centroids. Few approaches discretize the field in cells and try to determine the position in a more elaborated way, e.g., path planning, Pareto-optimum. In general, a simple positioning layer leads to more complexity in the higher layers to generate sophisticated behavior. The following approach strives to provide a generalized and flexible tool for the formulation of the robot positioning and simplify the upper layers defining the overall strategy.

To make our idea more clear we introduce a simple example. Imagine a situation with two robots and the ball placed in the opponent part of the field like depicted in the Fig. 1. We define the one closer to the ball to be the *striker* and the other one the *supporter*. Note, the robots will never change their roles in our examples as we are focusing only on the positioning problem itself. In this example neither the striker nor the ball moves. The only acting part is the

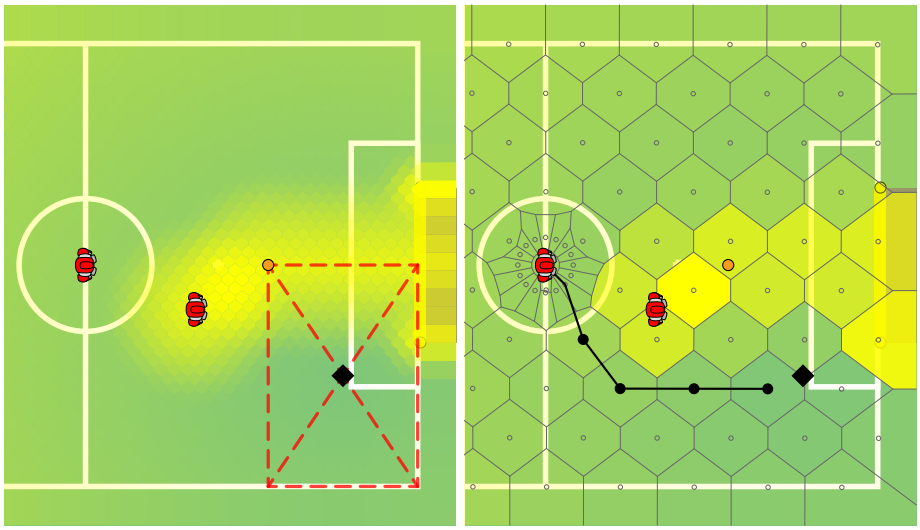


Fig. 1. An example situation: (left) initial positions of the supporter (center) and the attacker (closer to the ball); the center (black diamond) of the red dashed rectangle illustrates the target position for the supporter; the scalar field encoding the strategy is depicted by the intensity of the yellow glow (the global minimum is at the diamond); (right) the Voronoi tessellation with the weights of the regions depicted by the intensity of the yellow color; path calculated by the A*.

supporter. We consider the situation from the point of view of the supporter. Its only task is to move to a position where it could receive a pass or take over the ball in case the striker loses it. For that we assume a simple heuristic strategy for the supporter which is inspired by Carlos E. Agüero et al. in [1]: thereby the supporter's desired position p_0 is defined as the center of the rectangle spanned by the ball and the opposite opponent corner of the field as depicted in the Fig. 1. Now the task is to formulate the problem of the supporter getting to the point p_0 while avoiding collisions with other objects like the striker or the goal posts. Of course one could immediately imagine a simple positioning solution for this case. But the aim of the presented approach is to provide a general solution for a wide range of positioning problems. This example is just fine to illustrate the concept and to explore its basic principles.

3.1 General Idea

The general idea is to separate the field in weighted regions, which are then used to determine the target region as well as the path to this region. The conditions defining the desired position and the path can be formulated in terms of the separation in regions and the choice of weights. With this approach our example strategy could be formulated in a way where the weight of the region

containing the desired position p_0 is minimal and the regions containing obstacles are assigned high weights causing the path finder to avoid them.

There are numerous possibilities to separate the field in regions. At this point we decided to use *Voronoi tessellation*, which is a very powerful and flexible tool. The tessellation is defined by a set of points, called *Voronoi sites*, distributed over the field. Based on those points we use the *Fortune's algorithm* [5] to calculate the tessellation. Apart from a set of regions we also get a graph, called *Delaunay graph*, which is defined by the cells as nodes and the neighborhood as edges. This graph gives us a possibility for efficient search within the tessellation. With this we can easily construct very complex tessellations based on the conditions given by our strategy.

The whole *situation map* is defined by a Voronoi tessellation and positive weights assigned to each cell. Thus, the map consist of the spatial separation of the field in regions and a graph structure over the defining nodes. Basically, we can consider this map as a *weighted undirected graph* where the weights of the nodes are given directly by the definition and the weights for the edges are determined as a combination of the metric distance between the defining points and the weights of the nodes. From another point of view it can be seen as a discretized scalar field. To solve the positioning task we employ the A* algorithm to find the shortest path. Thereby the *start node* is the region containing the position of the robot and the target node defined by the minimal weight.

More precisely, the *Voronoi Based Situation Map* (VBSM) is defined by (G, W) where $G := (V, E)$ is the Delaunay graph of the Voronoi sites $V \subset \mathbb{R}^2$. The function $W : V \to \mathbb{R}$ assigns a weight $w \in \mathbb{R}_+$ to each node of the graph G. Note, each node represents a Voronoi cell and therefore the weights are assigned to each of the Voronoi cells.

3.2 Tessellation of the Field

To achieve a desired tessellation we have to choose the points (the Voronoi sites) appropriately. The resolution of the tessellation, i.e., number of points, has a major impact on the computational cost of the tessellation, the weights as well as the search. To resolve the trade-off between the accuracy and the speed, the field is discretized in two steps. At first points are chosen in a way that their Voronoi cells result in hexagons. These points are used to get a rough base resolution. In the second step the tessellation is refined around the position of the player, e.g., supporter. For that the position itself is added as a Voronoi site as well as 16 Voronoi sites around it, which are equidistant distributed on a circle. As the result, the scalar field will have a higher resolution around the player's position. Thus, the determined path will be more precise in the proximity of the player. Note that the geometry of the tessellation changes over time depending on the position of the player. The path calculated in one frame gives only a rough direction for the movement. The resulting path which emerges through the robot following the given directions will be much smoother as the higher resolution around the robot moves with it. The Fig. 1 (right) illustrates the resulting tessellation.

3.3 Positioning Strategy

As already described in the introductory example, the target position for the supporter is determined as the center of the rectangle, which is defined by the ball's position and the outer opponent corner (from ball's point of view). If the ball is close to the longitudinal axis of symmetry the determined position of the supporter might change the field sides due to noisy ball perception. To avoid this problem a hysteresis is used. We utilize scalar fields to formulate this strategy and to express it in terms of weights of the VBSM. Thereby, the target position is modeled as global minima of a scalar field. The striker, goal posts as well as the line between ball and opponent goal should be avoided and therefore are modeled as maxima of the scalar field. For each of the objects we introduce an id $\mathcal{I} := \{target, striker, line, goalpost, \cdots\}$. We assume there is a distance function $d_\iota : \mathbb{R}^2 \to \mathbb{R}_+$ defined for each of the objects $\iota \in \mathcal{I}$. The distance function d_ι assigns to each point $x \in \mathbb{R}^2$ the distance between x and the object ι. Except for the target position, the objects should have a limited range of influence. To formulate this we define the function $Q : \mathbb{R}_+ \to [0, 1]$ by

$$Q_{\alpha,\beta}(t) := \begin{cases} e^{\frac{\alpha}{\beta} - \frac{\alpha}{\beta - t}} & \text{, if } \beta > t \\ 0 & \text{, else} \end{cases} . \tag{1}$$

The function $Q_{\alpha,\beta}$ has a compact support which is bounded by the parameter β and has its maximum equal 1 in $t = 0$. The parameter β describes the radius of the influence and α describes the steepness of the slope of $Q_{\alpha,\beta}$. With this function we now can define scalar fields U_ι for each of the objects in $\mathcal{I}$:

$$U_{target}(p) := \frac{1}{l} \cdot d_{target}(p) \tag{2}$$

$$U_\iota(p) := Q_{\alpha_\iota,\beta_\iota}(d_\iota(p)) \tag{3}$$

where l is the length of the field diagonal. To model the striker and the line between the ball and the opponent goal we have used in our experiments the values $\alpha_{striker} := \alpha_{line} := 800$, $\beta_{line} := \beta_{striker} := 1000$ and for the goalposts we have used $\alpha_{goalpost} := 800$, $\beta_{goalpost} := 500$. For each Voronoi cell we define the weight as a sum of the scalar fields at the Voronoi site p defining the cell:

$$W(p) := \sum_{\iota \in \mathcal{I}} U_\iota(p) \tag{4}$$

3.4 Path Finding with A*

The graph structure of VBSM makes an efficient application of A* search possible. To define the cost function and the heuristic for the search, the weights of the nodes can be seen as height information. Figuratively, they shape a kind of mountains over the field, where the robot tries to get to the lowest point. With this idea the cost function c can be defined as the Euclidean distance in three dimensions:

$$c(p, q) := \left\| \begin{pmatrix} p_x \\ p_y \\ \alpha \cdot W(p) \end{pmatrix} - \begin{pmatrix} q_x \\ q_y \\ \alpha \cdot W(q) \end{pmatrix} \right\|_2 \tag{5}$$

where p, q are two nodes of the Delaunay graph and the weight W defined in the equation 4. The heuristic for a node p can be defined as the direct cost to the target $h(p) := c(p, p_0)$. The factor $\alpha \in \mathbb{R}^+$ is used to scale the influence of the weight on the cost function and the heuristic. The heuristic function defined this way is consistent which makes the A* search optimal.

4 Experimental Analysis

In this section we investigate some of the basic properties of the presented approach in isolated experimental setups. The experiments are performed in the simulator SimSpark. To analyze the properties of the VBSM we utilize the knowledge of the precise positions of the objects within the simulation. Thus, we can assume there is no sensory noise. In particular this allows to observe the actual path or the robot movement. The trajectory of the center of mass is projected onto the playing field as walked path.

The following experiments illustrate the VBSM in an example scenario of supporter positioning. It means, the situation is visualized from its point of view. In particular we consider two different situations. At first we consider the situation described in our introductory example from the section 3. To briefly recall: the ball is placed in the opponent half; a robot (*striker*) is placed next to the ball, while the *supporter* is placed in the center of the field; the only active party is the supporter: its task is to get to its strategic target position as depicted in Fig. 2. In the second situation the striker and the ball are located in the upper opponent field part as shown in Fig. 2 (right). Here, the supporter has to change the side to reach its position. In both experiments the influence of the cell weights on the search costs is varied by changing the parameter α in the cost- and heuristic function 5.

Due to the rough overall resolution of the tessellation, the *estimated path* calculated in one frame approximates the ideal path only roughly. However, as already mentioned, the geometry of the tessellation changes over time depending on the position of the player, which leads to a smooth *final path* which actually emerges through the robot following the estimated path as illustrated in the Fig. 2. The defining conditions are reflected much better as well.

The influence of the parameter α on the final path can clearly be seen in both situations illustrated in the Fig. 2. Figuratively spoken, the height of the mountains defined by the cell weights is scaled by α, which changes the relation between the cost of the actual distance and the costs produced by the weights. Thus, with a small α the way around an obstacle seems to be longer than the way through it and vice versa in the case of a large α. In the Fig. 2 (left) the path is closer to the obstacle for a smaller and farther for a larger α. In the Fig. 2 (right) a too small α causes the robot to ignore the virtual obstacle defined by the line between the ball and the goal. While the path is slightly adjusted in the fist scenario, it is changed qualitatively in the second.

The estimated path can also be seen as a *plan* which roughly reflects the situation on the field. The resolution of this plan is defined by the resolution of

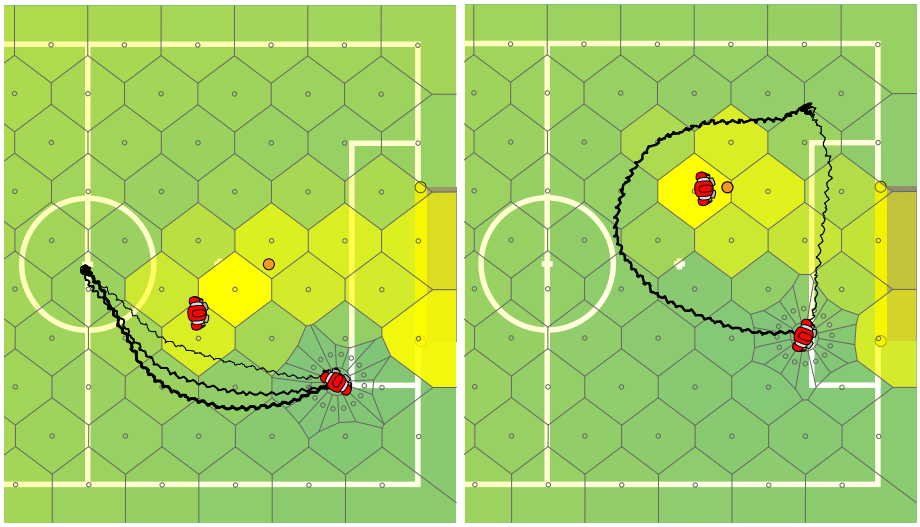


Fig. 2. Walk path of the robot in two different scenarios with different influence α of the cell weights in the search cost: from thin to thick the paths correspond to $\alpha = 600, 1000, 3000$; (left) the supporter walks to its position; (right) the supporter changes the side with parameters $\alpha = 600$ and $\alpha = 3100$;

the VBSM. Thus, the plan is refined as the robot moves, which corresponds to the *least commitment* principle. However, the resolution has to be high enough to reflect crucial qualitative aspects of the situation. For instance, in the case of too rough resolution some maxima might disappear between the cells in analogy to the *Nyquist–Shannon sampling theorem*.

5 Conclusion and Future Work

We presented a new method for strategic positioning in RoboCup based on a VBSM. Thereby the field is subdivided in regions by a Voronoi tessellation and each region is assigned a weight. The region with a minimal weight is chosen as the target region. The path to the target is estimated with A* on the Delaunay graph which is dual to the tessellation. Whereby, the distance between the nodes as well as the assigned weights are represented by a cost function and heuristics. A positioning strategy can be expressed in VBSM by the choice of the tessellation, i.e., the Voronoi points, and the weights of the regions.

An example implementation was illustrated in a scenario of supporter positioning. Thereby, scalar fields have been used to calculate the weights. The tessellation consists of two parts: a rough static tessellation of the whole field and higher resolution around the robots position. This implementation has been tested in simulation. It has been shown to produce a smooth resulting path in static situations despite a rough discretization, which is mainly due to the dynamic tessellation refinement for the direct vicinity of the robot.

This approach utilizes some well studied techniques and yields a flexible and powerful method for a description of positioning strategies. From our experiments the VBSM reveals a lot of capabilities and seems to be a promising approach for further development.

So far only static scenarios have been considered. The main focus of the ongoing research, is on investigating how more complex strategies can be formulated with VBSM as well as its behavior in dynamic situations. Further studies also need to investigate the choice of the tessellation as well as the way to assign the weights. In particular, the refinement of the tessellation seems to be a crucial aspect. Local refinement depending on the weights or along the estimated path could lead to better representation of conditions encoded in the weights. Also relaxing to a centroidal Voronoi tessellation could lead to a higher expressive power of the weights as the defining points would mark the center of a region representing it better. In the current implementation the tessellation is recalculated in every step, adaptive algorithms could reduce the computational costs by adjusting the old tessellation rather than calculating a new one. Similar to the tessellation, the weights could be propagated between the frames. This would provide a possibility for modeling some time dependent properties directly in the VBSM. For instance, the dominant regions could be estimated by propagation of the weights representing an opponent between the cells based on its motion model. Another idea is to lower the weights along the estimated path, which would result in a kind of memory for the path and prevent oscillations.

References

1. Agüero, C.E., Matellán, V., Caˆnas, J.M., Gómez, V.M., Carlos, J.: Switch! dynamic roles exchange among cooperative robots. In: in Proceedings of the 2nd International Workshop on Multi-Agent Robotic Systems - MARS 2006. INSTICC. pp. 99–105. Press (2006)
2. Akiyama, H., Noda, I.: Multi-agent positioning mechanism in the dynamic environment. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI, Lecture Notes in Computer Science, vol. 5001, pp. 377–384. Springer Berlin / Heidelberg (2008)
3. Dashti, H.T., Kamali, S., Aghaeepour, N.: Positioning in robots soccer. In: Lima, P. (ed.) Robotic Soccer, pp. 29–44. I-Tech Education and Publishing (2007)
4. Dashti, H., Aghaeepour, N., Asadi, S., Bastani, M., Delafkar, Z., Disfani, F., Ghaderi, S., Kamali, S., Pashami, S., Siahpirani, A.: Dynamic positioning based on voronoi cells (dpvc). In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Computer Science, vol. 4020, pp. 219–229. Springer Berlin / Heidelberg (2006)
5. Fortune, S.: A sweepline algorithm for voronoi diagrams. Algorithmica 2, 153–174 (1987)
6. Hannebauer, M., Wendler, J., Pagello, E., Reis, L., Lau, N., Oliveira, E.: Situation based strategic positioning for coordinating a team of homogeneous agents. In: Balancing Reactivity and Social Deliberation in Multi-Agent Systems, Lecture Notes in Computer Science, vol. 2103, pp. 175–197. Springer Berlin / Heidelberg (2001)

7. Kyrylov, V., Hou, E.: Pareto-optimal collaborative defensive player positioning in simulated soccer. In: Baltes, J., Lagoudakis, M., Naruse, T., Ghidary, S. (eds.) RoboCup 2009: Robot Soccer World Cup XIII, Lecture Notes in Computer Science, vol. 5949, pp. 179–191. Springer Berlin / Heidelberg (2010)
8. Kyrylov, V., Razykov, S.: Pareto-optimal offensive player positioning in simulated soccer. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI, Lecture Notes in Computer Science, vol. 5001, pp. 228–237. Springer Berlin / Heidelberg (2008)
9. Lau, N., Seabra Lopes, L., Filipe, N., Corrente, G.: Roles, positionings and set plays to coordinate a robocup msl team. In: Lopes, L., Lau, N., Mariano, P., Rocha, L. (eds.) Progress in Artificial Intelligence, Lecture Notes in Computer Science, vol. 5816, pp. 323–337. Springer Berlin / Heidelberg (2009)
10. McMillen, C., Veloso, M.: Distributed, play-based coordination for robot teams in dynamic environments. In: Lakemeyer, G., Sklar, E., Sorrenti, D., Takahashi, T. (eds.) RoboCup 2006: Robot Soccer World Cup X, Lecture Notes in Computer Science, vol. 4434, pp. 483–490. Springer Berlin / Heidelberg (2007)
11. Nakanishi, R., Bruce, J., Murakami, K., Naruse, T., Veloso, M.: Cooperative 3-robot passing and shooting in the robocup small size league. In: Lakemeyer, G., Sklar, E., Sorrenti, D., Takahashi, T. (eds.) RoboCup 2006: Robot Soccer World Cup X, Lecture Notes in Computer Science, vol. 4434, pp. 418–425. Springer Berlin / Heidelberg (2007)
12. Nakanishi, R., Murakami, K., Naruse, T.: Dynamic positioning method based on dominant region diagram to realize successful cooperative play. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI, Lecture Notes in Computer Science, vol. 5001, pp. 488–495. Springer Berlin / Heidelberg (2008)
13. Nieuwenhuisen, M., Steffens, R., Behnke, S.: Local multiresolution path planning in soccer games based on projected intentions. In: Röfer, T., Mayer, N., Savage, J., Saranlı, U. (eds.) RoboCup 2011: Robot Soccer World Cup XV, Lecture Notes in Computer Science, vol. 7416, pp. 495–506. Springer Berlin Heidelberg (2012)
14. Petersen, K., Stoll, G., von Stryk, O.: A supporter behavior for soccer playing humanoid robots. In: Ruiz-del Solar, J., Chown, E., Plöger, P. (eds.) RoboCup 2010: Robot Soccer World Cup XIV, Lecture Notes in Computer Science, vol. 6556, pp. 386–396. Springer Berlin / Heidelberg (2011)
15. Phillips, M., Veloso, M.: Robust supporting role in coordinated two-robot soccer attack. In: Iocchi, L., Matsubara, H., Weitzenfeld, A., Zhou, C. (eds.) RoboCup 2008: Robot Soccer World Cup XII, Lecture Notes in Computer Science, vol. 5399, pp. 235–246. Springer Berlin / Heidelberg (2009)
16. Rad, A., Qaragozlou, N., Zaheri, M.: Scenario-based teamworking, how to learn, create, and teach complex plans? In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003: Robot Soccer World Cup VII, Lecture Notes in Computer Science, vol. 3020, pp. 137–144. Springer Berlin / Heidelberg (2004)
17. Röfer, T., Laue, T., Müller, J., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Humann, A., Honsel, D., Kastner, P., Kastner, T., Könemann, C., Markowsky, B., Riemann, O.J.L., Wenk, F.: B-human team report and code release 2011. Tech. rep. (2011)
18. Work, H., Chown, E., Hermans, T., Butterfield, J., McGranaghan, M.: Player positioning in the four-legged league. In: Iocchi, L., Matsubara, H., Weitzenfeld, A., Zhou, C. (eds.) RoboCup 2008: Robot Soccer World Cup XII, Lecture Notes in Computer Science, vol. 5399, pp. 391–402. Springer Berlin / Heidelberg (2009)

Adaptive Grasping for a Small Humanoid Robot Utilizing Force- and Electric Current Sensors

Heinrich Mellmann, Marcus Scheunemann, and Oliver Stadie

Department of Computer Science, Cognitive Robotics Group
Humboldt-Universitt zu Berlin, Germany
{mellmann,scheunem,stadie}@informatik.hu-berlin.de

Abstract. The ability to grasp objects of different size and shape is one of the most important skills of a humanoid robot. Human grasping integrates a lot of different senses. In particular, the tactile sensing is very important for a stable grasping motion. When we lift a box without knowing what is inside, we do it carefully using our tactile and proprioceptive senses to estimate the weight and thus, the force necessary to hold and to lift this box. In this paper we present an adaptive controlling mechanism which enables a robot to grasp objects of different weights. Thereby, we only use the proprioceptive sensors like positions and electric current at the joints and force sensors at the end-effectors providing the robot with tactile feedback. We implemented and tested our approach on a humanoid robot.

1 Introduction

One of the features that made humans a very successful living being is their ability to grasp and manipulate objects. Such feature granted humans the possibility to modify and adapt the surrounding environment making it more suitable to their own needs. For such reason, grasping and manipulating objects can be seen as a strategic goal for robotics. Restraining objects is a not trivial task due to each object's geometrical and physical peculiarities.

In particular, to grasp objects of different weights requires different force to be applied for holding as well as for lifting. It has been shown in case of humans that the force is adjusted anticipatory for both, the grasping as well as the lifting of the object. At this junction anticipatory means a pre-evaluation based on certain assumptions, e.g., on experience or visual analysis of the object. However, a wrongly estimated force can quickly be adjusted while grasping before the fingers are slipping on the surface (cf. [6]). These adjustments are very reactive and not pre-planned by the higher cognition. As discussed in [9] this reflex is also called the *grasping force control reflex*. Some research has been done on implementing grasping reflexes on anthropomorphic robotic hands [4]. In particular, in [7] it has been tried to imitate the primitive grasping reflex to grasp unknown objects.

In this paper we present an implementation of an adaptive bimanual grasping motion on a humanoid robot *Nao* [5], which is based on the concept of humans' grasping force control reflex. According to [3] this work with limited hardware

can be considered to the *minimalistic approach to design*. Our algorithm is especially able to adapt to objects of different weights by only using proprioceptive sensors and tactile feedback. Thereby, only a few assumptions regarding the properties of the objects are made. The basic idea is as simple as human strategy: the robot tries to lift an object with as less force as possible and increases its efforts in case it does not succeed. In order to recognize whether the object is grasped or not we use proprioceptive sensors of the robot. The whole algorithm is realized by local sensory loops. Thus, it is highly adaptive and requires only little computational resources.

These discussed methods adapt in a reflex-like manner to the respective situation without planning more than one step in advance as well as without extensive models or knowledge about the environment and about object to be grasped. These local cognition methods may be embedded in existing grasping methods and as a result, make them more robust to noise and environmental changes. By the way we implicitly explore properties of the object to be grasped, like the weight. This task belongs to the field of haptics more than to robotics according to [3].

The robot's grasping capabilities highly depend on the hands' mechanical structure, its sensors and, of course, the available computational power. Moreover, a robot has to be able to perform stable and flexible motions in order to act in a dynamic environment, moving the whole body whenever necessary. This is especially important if an object has to be grasped with both hands. The presented dynamic control is integrated in a complete grasping behavior as described in [8].

A detailed survey about the modeling of the grasping movement is demonstrated in [3]. The general approach is to calculate the contact points first. Extensive models and knowledge about the environment as well as the object to be grasped are the basis for such calculations. The trajectory of the hands in order to reach those points and the force to ideally hold the object are calculated afterwards. Third, after adequately fixing the object further calculated trajectories ensure that the object can be moved while staying fixed.

1.1 Outline

The remainder of the paper is structured as follows. At first we briefly outline the hardware of the robot used. Thereby, we make a particular accent on its sensing capabilities and its kinematic constraints. In the third section we present the general design of the grasping algorithm and the dynamic control. In the fourth part we show some experimental results benchmarking the control effectiveness and we suggest some ideas where to address the further developments in the last part.

2 Platform analysis

Nao robot is a humanoid robot produced by the French company *Aldebaran Robotics* and is currently used in *RoboCup* competitions within the *Standard*

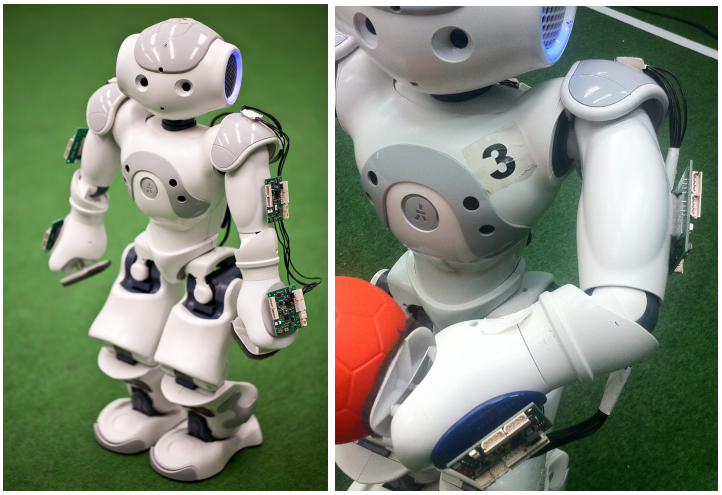


Fig. 1. Humanoid robot *Nao* by *Aldebaran Robotics* equipped with additional force sensors at its hands.

Platform League [1]. In this section we systematically analyze the available grasping abilities of the robot. At first we present the hardware, then we explore the arm's workspace and its constraints and finally we discuss its sensing capabilities.

2.1 Nao Robot

Nao robot has a very articulate body, it is 58cm of size and weighs about 4.8kg including the battery. Each arm has four degrees of freedom describing a workspace quite similar to the human arm's one. The joints are actuated by DC motors and the platform is equipped with a low power and low consumption *Geode LX 800* processor with just 500 MHz. Since the CPU processing power is quite limited compared to the robot's physical structure, it is challenging to implement very complex algorithms for motion controlling. Therefore simplicity in design is the preferred approach. Figure 1 shows the robot *Nao*, while standing and while grasping a ball.

Each of the joints is controlled by a PID controller. The API provides two values for controlling each joint: target angle which should be reached and the maximal electric current which is used to drive the joint. The last one is also called *stiffness* of the joint, since it defines how hard the joint will try to reach the requested position. Further details can be found in [5].

2.2 Sensors

The robot *Nao* is equipped with four force sensors on each foot, a gyroscope, an accelerometer, two ultrasounds in the chest and two VGA cameras (operating on a single bus) in the head. Each joint is equipped with sensors measuring the

actual angular position and the electric current consumed by the motor. The camera images can be received up to 30 times per second, while all other sensor data, like joint's positions and electric current, can be read every 10 ms. The motion system requires a control signal at the same frame rate, i.e., every 10 ms, to ensure the correct execution of the planned movements. The gyroscope and the accelerometer can be used together with the feet's force sensors for inferring robot body's posture, while ultrasounds can be used for inferring the position of obstacles in the front of the robot. The camera provides information about the surrounding environment that is very effective for navigation and object recognition. In particular, visual sensing can be used to control the high level parts of the grasping motion, like aligning the hands around the ball, which do not require very high reactivity. In [8] the joint's motor internal sensors were chosen to estimate the force applied by the end effector. In order to be able to grasp and lift objects with different shape, weight and sturdiness, the robot has to adapt the applied force, receiving a sensor feedback if necessary. In our research we apply additional force sensors instead of hands to be more precisely and simulate a one dimensional haptic perception.

2.3 Physical Preconditions

A robot's grasping ability is highly dependent on its kinematic constraints. These are, among other things, determined by the *reachable space* of the robot's hands. The *reachable space* is usually defined as the set of points that can be reached by its end effector, e.g., the hand, with respect to a reference frame of the robot. In general, this space is defined by some basic physical constraints that a humanoid robot has to satisfy during the motion, including the kinematic constraint (e.g., the limits of joint angles; and the collision constraint) and the balance constraint. We represent the *reachable space* by a three dimensional grid, thereby we consider basically the positions of the end effector but not its rotation. Figure 2 illustrates the reachability grid for a hand of the *Nao* robot.

Nao's arms are equipped with four joints, two for the shoulder and two for the elbow as shown in the Figure 1 (right). Both links can be controlled along the roll, while the second joint controls the pitch and the yaw for the shoulder and the elbow respectively. The end effectors can operate in relatively large workspaces that are partially overlapping each other. However, such freedom of movement is a further node of complexity for determining a successful grasping pose, because the same point in the space can be reached in many ways that differs only on the end effector orientation. To reduce complexity, we fixed one joint, thereby the degrees of freedom of each arm reduced by one. We decided to fix the yaw-joint of the elbow, because its fixation limits the robot at least. The right grid in the Figure 2 was generated with a fixed yaw-joint in the elbow. Considering the difference between the full and the restricted grid, shown in the center of the Figure 2, it can be seen that only some positions behind the robot are lost by this restriction.

An interesting challenge to overcome on the *Nao* robot is dealing with the hand's structure itself. In the used version the *Nao* robot typically owns passive

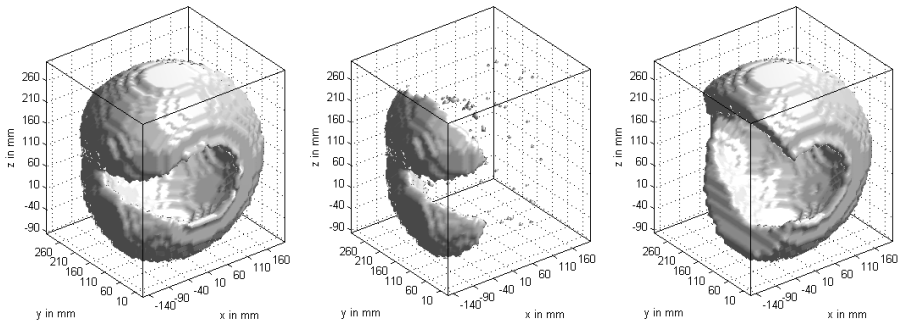


Fig. 2. The *reachable space* of the *Nao's* hand is approximated by a three dimensional grid; (left) the theoretical reachable grid generated in simulation; (right) reachable positions with the fixed yaw-joint in the elbow; (center) difference between both grids;

hand effectors, which are equipped with three fingers. The passive fingers of the robot result in very irregular surfaces making the control of the object more difficult. Therefore we replace the hands by flat gripping adapters, as described in 2.2, with expanded material to have friction characteristics similar to human skin. Additionally, force sensors were built into the adapters, which provide better control of the exerted force.

3 Grasping Algorithm

In this section we illustrate some simple controllers used for implementing a grasping motion. At first we show the system's infrastructure and an outline of the control mechanism, on the second part we discuss three possible regulators and finally we provide some experimental data to better analyze the motion control.

3.1 General Design

The whole grasping motion can be divided in two parts: approaching the object and restraining it. The first part of the motion is needed to bring the item in the hands' reachability space and can be further decomposed in the tasks of recognizing the object, reaching it and crouching. Once the target is reachable the core of the grasp motion is executed. At first the hands are aligned to the object and then they are moved in order to squeeze the target. This part of the motion assumes as reference system the robot's chest as visualized in the Figure 3 (right).

The end effectors are driven by inverse kinematic, so the same point can be reached with different arm configurations. This feature makes the motion more general but introduces an extra degree of complexity due to the freedom

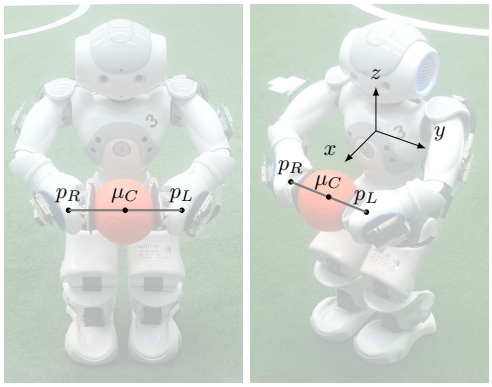


Fig. 3. Geometry of the grasping motion: μ_C denote the center of the grasping motion (e.g., estimated center of gravity of the ball), p_L and p_R are the desired points for the hands.

of rotation of the arm. In fact, hand's orientation is coupled with elbow's orientation making impossible having a direct influence on hand's rotation without modifying the end effector position, as already pointed out in Section 2.3.

As a consequence, an item may be approached using different arm configurations but not all of them offer a convenient grasping surface. For this reason, the elbow's rotation is fixed for the duration of the entire grabbing motion, forcing the hands to touch an object always on the side with the flat gripping adapters. In this way the *Nao's* arms can be seen as a big two fingered gripper.

Thus, we formulate the geometry of the grasping task as follows: in the first step align the hands around the point $\mu_C \in \mathbb{R}^3$, representing the center of the object, with a certain distance $\rho \in \mathbb{R}_+$. In the second step close the hands reducing the distance ρ between the hands and the point μ_C. Thus, the actual target positions for the hands can be calculated as points with the distance ρ left and right from μ_C, i.e., $p_L = \mu_C + \rho \cdot e_2$ and $p_R = \mu_C - \rho \cdot e_2$ for the right and left hand respectively, whereas $e_2 = (0, 1, 0)^T$. The point μ_C is controlled by vision and is used for choosing a useful spot where to grasp the object, since the hands will be placed according to it. The distance ρ is driven by joint sensor feedback and a mapping of the end effector force applied to the target: the smaller ρ the bigger the force intensity. Figure 3 visualizes the geometrical configuration of the grasping.

3.2 Dynamic Control

In this section we discuss an integrated controlling mechanism adapting dynamically to the weight of the grasped object. This allows for grasping and lifting objects having different weights. The main task here is to estimate the right force which is needed to grasp the object. The trivial solution is, of course, to take the maximum force available. However, this strategy is obviously very inefficient

and may destroy fragile objects, e.g., a paper cup. Thus, our general strategy for the grasping is: as soft as possible, but as strong as necessary.

The problem to determine the force necessary for grasping is stated in [2] as one of the most important sub problems of the grasping task. As we already discussed in [8] the calculation efforts for the estimation of the necessary grasping force may become very high. The presented method costs only few calculations.

The whole controller consists mainly of two parts: controlling of the stiffness and controlling of the distance between the hands. Both parts are designed in a way allowing for them to be considered independently.

Stiffness The stiffness is controlled for each joint separately by a P-controller. Thereby, the stiffness is locally determined to be proportional to the difference between the requested and the measured angle of the joint. Formally, the stiffness σ at a joint is determined by

$$\sigma = |\hat{\alpha} - \alpha| \cdot C$$

where $\hat{\alpha}$ is the measured angle and α the requested one. The constant C can be determined experimentally.

I.e., the stiffness of a joint is reduced to a minimum in the case if the requested angle position can be reached. However, if the joint is prevented from reaching the requested position by some external force, the stiffness is increased and the joint is working with more force against the external obstacle. Thus, each of the joints is reacting locally according to an external force.

Distance between Hands In order to control the distance ρ between the robots hands we use a threshold controller based on the force F measured at the hands of the robot, i.e., the distance between the hands is successively reduced by a δ until the force F exceeds a certain threshold M. By this we ensure that a certain minimal force M is applied to the object during the grasping.

$$\rho(t) = \begin{cases} \rho(t-1) - \delta & \text{for } F(t-1) < M \\ \rho(t-1) & \text{otherwise} \end{cases}$$

Thereby, for each object with a different weight we need another appropriate force in order to lift it, i.e., in particular we need for each object a different threshold for the controller. This threshold can be estimated by the means of the controlling electric current of the joints (cf. Section 2.2). The more load is on a joint, the higher is the corresponding electric current. In particular, the pitch joints of the shoulders appeared to be the best suitable for this estimation. This is because they are the only arm joints in our grasping geometry which apply vertical force to the object. Our experiments have shown that for the used test objects a cubic dependency between the threshold M and the electric current is sufficient, i.e., for the measured current I_L and I_R at the left and right shoulder respectively we can write

$$M = \max(I_L, I_R)^3 \cdot C$$

Fig. 4. Top row: robot is grasping a plastic bottle of coke with a weight of ca. 400g which requires the maximal force the robot can apply; bottom row: grasping a paper cup requires a gentle touch, a forceful grasp would deform the cup;

with an experimentally determined constant C. Intuitively, this rule means that for a bigger weight of the object a larger grasping force is applied, i.e., the heavier an object, the stronger the robot is grasping.

It should be remarked that the relation between the vertical lifting force which is produced by the shoulder joints and the tangential force which is necessary to hold the object between the hands strongly depends of the friction between the hand palms and the object.

4 Experiments

To study the behavior of the integrated controller III-B an isolated scenario was set up. The general robot's task is to grasp and lift an object placed in front of itself. For this challenge we use objects of different weights and consistencies. We chose an empty coffee cup and a full cola plastic bottle as representative objects. The cup weighs about 30g by the robot and can be crushed easily, whereas the bottle is very sturdy and weighs approximately 500g.

The robot behavior in the experiment consists of three phases:

1. sit down and stretch the arms (positioned left and right of the object);

2. clasp the hands around the object;
3. stand up with the grasped object and lifting it;

During the second and third phase the intrinsic grasp control mechanism is active. The series of photographs shown in the Figure 4 illustrates the progress of the experiment in which the robot lifts a bottle. Figure 5 visualizes some sensor values, which were recorded during the experiment with the cup respectively with the bottle. The upper graph shows the development of the force measured on the hands of the robot, the center graph illustrates the development of the controlling electric current measured at the shoulder pitch joints. For symmetric reasons, in both cases the maximum is calculated between the left and the right sensor. The vertical dashed lines separate the different phases of the grasping motion.

At the end of the second phase you are able to spot the first contact with the object in the image above, exactly in this moment the force increases for the first time. The contact with the cup occurs earlier, because it has a slightly larger radius than the bottle. In this example it can be clearly noticed how the movement is adapted to different forms. In the third phase, the robot tries to stand up while grasping the object. Concurrently the robot tries to lift the object slightly with its arms. Therefore the shoulder pitch joints are actuated (stretched) and we are able to measure the increasing controlling electric current. If the object cannot be lifted the control current would increase steadily, which can be observed in the middle graph of Figure 5.

The increase of the current leads to the reduction of the distance between the hands and increases the force on the object. This strengthens the connection between the object and hands. If the object is lifted the shoulder current will not continue to increase and the force on the object remains on the current state of the power in the shoulders.

This behavior can be observed very well in the top two graphs. In the case of the cup (thin line) the current in the shoulders increases slightly till the cup is lifted, whereas in the case of the bottle (thick line) the current increases more significantly, with the result that the force rises to about $8N$ and the friction between the hand and the bottle is large enough to lift it.

In this way just as much force is exerted as needed to generate the necessary friction for lifting, thereby a behavior is originated that allows a lifting of light and heavy objects with a minimum effort. That means the robot does not spend more force than necessary. Another aspect is that in this way light and fragile items, such as a cup, are not deformed or even broken.

5 Conclusions and Future Work

Grasping is still a hard task for a robot. As the main result of this paper an algorithm was presented which enables a robot to grasp and control objects with different weights. In the experimental setup the robot was able to lift a fragile cup and a comparably heavy bottle. The most remarkable aspect is the actuation of the arms while grasping the objects. By measuring the controlling

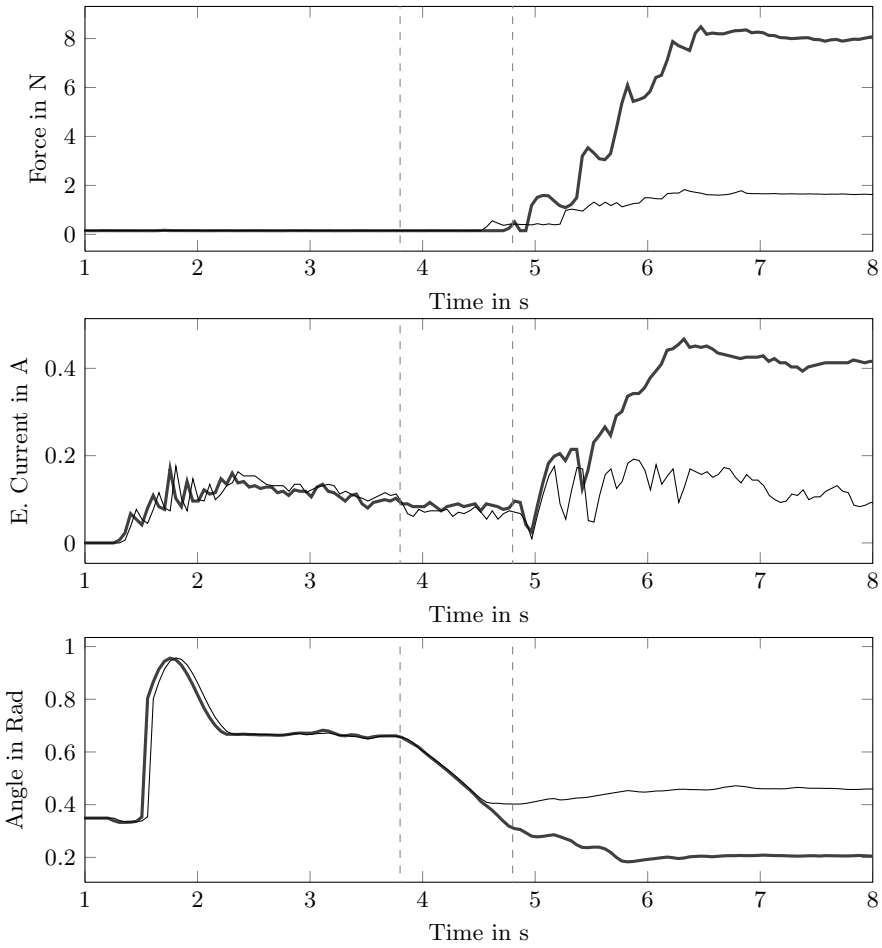


Fig. 5. Sensory data of the robot *Nao* recorded while lifting two different objects: a light paper cup (thin line) and a full plastic bottle of coke 0,5L (thick line). At the top the progress of the force measured at the hands of the robot can be seen, thereby the maximum of both hands is plotted. The middle plot illustrates the electric current measured at the pitch joints of the robots shoulders, again the maximum of both shoulders is visualized. At least, the bottom graph shows the measured roll-angle of the left shoulder (both shoulders are moved symmetrically, so it is enough to plot only one angle). The vertical dashed lines separate the different phases of the grasping motion.

electric current at the shoulder joints, an estimation of the tangential force which is applied to the arms is allowed. If the arms are not actuated the electric current does not behave proportional regarding the object's weight due to the friction in the gears.

Our future research will focus on more general rules for the adaptation and for a more precise estimation of the object's weight. In particular, measurements of other body joints, e.g., the electric current of the knee, could also be incorporated in order to exploit redundancy and to archive a better estimation of the force.

Additionally, we are working on a compensation of the weight of the object by balancing the body. In order to do this the force resistive sensors in the feet of the robot could be used. Thinking ahead, we hope to enable the robot to estimate the actual weight of an object with respect to its own by exactly that kind of balancing. Last not least, it is intended to incorporate this knowledge in its own kinematic model so that as a consequence, a robot is able to walk with an object, e.g., a bottle, still compensating its weight and inertia.

References

1. RoboCup website (1997), http://www.robocup.org
2. Bicchi, A., Kumar, V.: Robotic grasping and contact: A review. In: Proc. IEEE Int. Conf. on Robotics and Automation. pp. 348–353. San Francisco, CA (2000)
3. Bicchi, A., Kumar, V.: Robotic grasping and manipulation. In: Nicosia, S., Siciliano, B., Bicchi, A., (eds.), P.V. (eds.) Ramsete: Articulated and mobile robots for services and Technology, vol. 270, chap. 4, pp. 55–74. Springer-Verlag, Berlin Heidelberg, Germany (2001)
4. Folgheraiter, M., Gini, G.: Human-like reflex control for an artificial hand. BioSystems 76(1-3), 65–74 (2004)
5. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: Mechatronic design of nao humanoid. In: Robotics and Automation, 2009. ICRA '09. IEEE International Conference on. pp. 769 –774 (May 2009)
6. Jeannerod, M.: Intersegmental coordination during reaching at natural visual objects. In: Attention and Performance. vol. IX (1981)
7. Kawasaki, H., Mouri, T., Takai, J., Ito, S.: Grasping of unknown object imitating human grasping reflex. In: 15th Triennial World Congress (2002)
8. Mellmann, H., Cotugno, G.: Dynamic motion control: Adaptive bimanual grasping for a humanoid robot. Fundamenta Informaticae (2011), to appear
9. Tatsuma Sakurai, Masashi Konyo, S.O., Tadokoro, S.: Research of conditions of stimulus for inducing grasping force control reflex. In: Proceedings of the 2010 IEEE/SICE International Symposium on System Integration (SII'10). pp. 408–413 (Dec 2010)

Towards a Jason Infrastructure for Soccer Playing Agents*

Extended Abstract

Dejan Mitrović[1], Mirjana Ivanović[1], and Hans-Dieter Burkhard[2]

[1] Department of Mathematics and Informatics, Faculty of Sciences,
University of Novi Sad, Serbia
{dejan, mira}@dmi.uns.ac.rs
[2] Humboldt University, Institute of Informatics,
Rudower Chaussee 25, D-12489 Berlin, Germany
hdb@informatik.hu-berlin.de

Abstract. *AgentSpeak* and its practical interpreter *Jason* represent an excellent framework for implementing complex, reasoning agents. This paper discusses an ongoing research dedicated to extending *Jason* with the support for soccer playing agents. The end goal is to design an efficient infrastructure, capable of deploying and running *BDI* agents in the *RoboCup* soccer simulation league.

1 Intelligent agents playing soccer

RoboCup is an annual, internationally-recognized competition of football/soccer playing robots [5]. By providing a formidable challenge in a fun environment, its main goal is to support and further motivate the development of various artificial intelligence techniques.

Many concepts of the multi-agent technology, including autonomy, pro-active behaviour, coordination and cooperation, fit naturally into requirements of the *RoboCup* competition. These concepts are directly supported by the complex, *Belief-Desire-Intention* (*BDI*) agent architecture [6]. The *BDI* architecture has a strong mathematical basis and is widely supported by a number of agent development frameworks [2]. Our previous work on deploying *BDI* agents in *RoboCup* simulations [4] was based on the agent-oriented programming language *AgentSpeak* and its accompanying interpreter *Jason* [1]. The main reasons *Jason* was selected as for this task include its direct support for *BDI*, and a high level of customizability.

By analyzing the inner workings of *Jason* and the simulator, it was concluded that both systems support agents that operate in *sense-think-act* cycles. This fact simplifies the integration process significantly. To deploy *Jason* agents, it

* This work was partially supported by Ministry of Education, Science and Technological Development of the Republic of Serbia, through project no. OI174023: "Intelligent techniques and their integration into wide-spectrum decision support"

is sufficient to extend and modify the following set of the interpreter's subcomponents:

- *Simulated environment*: a model of the game that enables the agents to *sense* their surroundings, and *act* accordingly. The environment was extended with custom *parser* and *generator* components which, respectively, extract agent's belief literals from the simulator's set of percepts, and transform agent actions into concrete effectors;
- *Execution control*: handles *Jason* reasoning cycles. Development of a custom execution control was necessary for several reasons, including the support for *key-framed motions*. Key-framed motions often span across multiple *Jason* reasoning cycles. The execution control assures that the appropriate sets of commands are sent to the simulator as the motion progresses;
- *Agent architecture*: a link between the simulated environment and the remaining components.

Our custom implementation of these components was evaluated using a concrete implementation of a soccer playing agent [4]. The results have shown that *Jason* is perfectly capable of satisfying strict time constraints imposed by the official *RoboCup* simulator. However, further improvements and extensions are needed in order to implement and deploy agents that exhibit more complex behaviour. Our ongoing work is dedicated to designing and re-implementing the remaining parts of the *Jason* infrastructure [1]. This step is necessary in order to fully integrate *Jason* into the *RoboCup* simulator, allowing *Jason* agents to actually compete against other teams, and to do so by relying on extensively researched and well-understood concepts and methodologies of the multi-agent technology. In the long run, the plan is to further extend the infrastructure with *MOISE+*, an advanced *Jason*-compatible framework for organizational modelling that has already been tested in virtual soccer simulations [3].

References

1. Bordini, R.H., Hubner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Wiley Series in Agent Technology, John Wiley & Sons Ltd (2007)
2. Bădică, C., Budimac, Z., Burkhard, H.D., Ivanović, M.: Software agents: Languages, tools, platforms. Computer Science and Information Systems 8(2), 255–298 (2011)
3. Hubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering 1(3/4), 370–395 (2007)
4. Mitrović, D., Ivanović, M., Burkhard, H.D.: Intelligent Jason agents in virtual soccer simulations. In: Klusch, M., Thimm, M., Paprzycki, M. (eds.) Multiagent System Technologies - 11th German Conference. Lecture Notes in Computer Science, vol. 8076, pp. 334–345. Springer (2013)
5. RoboCup homepage. `http://www.robocup.org/`, retrieved on June 25, 2013
6. Wooldridge, M.J.: Reasoning about rational agents. Intelligent Robotics and Autonomous Agents, The MIT Press (2000)

An ExpTime Tableau Method for Dealing with Nominals and Quantified Number Restrictions in Deciding the Description Logic SHOQ

Linh Anh Nguyen[1,2] and Joanna Golińska-Pilarek[3]

[1] Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl
[2] Faculty of Information Technology, VNU University of Engineering and Technology
144 Xuan Thuy, Hanoi, Vietnam
[3] Institute of Philosophy, University of Warsaw
Krakowskie Przedmieście 3, 00-927 Warsaw, Poland
j.golinska@uw.edu.pl

Abstract. We present the first tableau method with an ExpTime (optimal) complexity for checking satisfiability of a knowledge base in the description logic $\mathcal{SHOQ}$, which extends $\mathcal{ALC}$ with transitive roles, hierarchies of roles, nominals and quantified number restrictions. The complexity is measured using binary representation for numbers. Our procedure is based on global caching and integer linear feasibility checking.

1 Introduction

Description logics (DLs) are formal languages suitable for representing terminological knowledge. They are of particular importance in providing a logical formalism for ontologies and the Semantic Web. Automated reasoning in DLs is useful, for example, in engineering and querying ontologies. One of basic reasoning problems in DLs is to check satisfiability of a knowledge base in a considered DL. Most of other reasoning problems in DLs are reducible to this one.

In this paper we study the problem of checking satisfiability of a knowledge base in the DL $\mathcal{SHOQ}$, which extends the basic DL $\mathcal{ALC}$ with transitive roles (S), hierarchies of roles (H), nominals (O) and quantified number restrictions (Q). It is known that this problem in $\mathcal{SHOQ}$ is ExpTime-complete [16] (even when numbers are coded in binary).

Nominals, interpreted as singleton sets, are a useful notion to express identity and uniqueness. However, when interacting with inverse roles (I) and quantified number restrictions in the DL $\mathcal{SHOIQ}$, they cause the complexity of the above mentioned problem to jump up to NExpTime-complete [15] (while that problem in any of the DLs $\mathcal{SHOQ}$, $\mathcal{SHIO}$, $\mathcal{SHIQ}$ is ExpTime-complete [16, 7, 15]).

In [8] Horrocks and Sattler gave a tableau algorithm for deciding the DL $\mathcal{SHOQ}$(D), which is the extension of $\mathcal{SHOQ}$ with concrete datatypes. Later, Pan and Horrocks [13] extended the method of [8] to give a tableau algorithm

for deciding the DL $\mathcal{SHOQ}(\mathrm{D}_n)$, which is the extension of $\mathcal{SHOQ}$ with n-ary datatype predicates. These algorithms use backtracking to deal with disjunction ($\sqcup$) and "or"-branching (e.g., the "choose"-rule) and use a straightforward way for dealing with with quantified number restrictions. They have a non-optimal complexity (NExpTime) when unary representation is used for numbers, and have a higher complexity (N2ExpTime) when binary representation is used. In [1] Faddoul and Haarslev gave an algebraic tableau reasoning algorithm for $\mathcal{SHOQ}$, which combines the tableau method with linear integer programming. The aim was to increase efficiency of handling quantified number restrictions. However, their algorithm still uses backtracking to deal with disjunction and "or"-branching and has a non-optimal complexity ("double exponential" [1]).

In this paper we present the first tableau method with an ExpTime (optimal) complexity for checking satisfiability of a knowledge base in the DL $\mathcal{SHOQ}$. The complexity is measured using binary representation for numbers. Our procedure is based on global caching and integer linear feasibility checking.

The idea of global caching comes from Pratt's work [14] on PDL. It was formally formulated for tableaux in some DLs in [3, 4] and has been applied to several modal and description logics (see [12] for references) to obtain tableau decision procedures with an optimal (ExpTime) complexity. A variant of global caching, called global state caching, was used to obtain cut-free optimal (ExpTime) tableau decision procedures for several modal logics with converse and DLs with inverse roles [5, 6, 9, 11].

Integer linear programming was exploited for tableaux in [2, 1] to increase efficiency of reasoning with quantified number restrictions. However, the first work that applied integer linear feasibility checking to tableaux was [10, 11]. In [10], Nguyen gave the first ExpTime (optimal) tableau decision procedure for checking satisfiability of a knowledge base in the DL $\mathcal{SHIQ}$, where the complexity is measured using binary representation for numbers. His procedure is based on global state caching and integer linear feasibility checking. In the current paper, we apply his method of integer linear feasibility checking to $\mathcal{SHOQ}$. It substantially differs from Farsiniamarj's method of exploiting integer programming for tableaux [2]. Our method of dealing with both nominals and quantified number restrictions is essentially different from the one by Faddoul and Haarslev [1].

Due to the lack of space, we restrict ourselves to introducing the problem of checking satisfiability of a knowledge base in the DL $\mathcal{SHOQ}$ and presenting some examples to illustrate our tableau method. For a full description of a tableau decision procedure with an ExpTime complexity we refer the reader to [12].

2 Notation and Semantics of $\mathcal{SHOQ}$

Our language uses a finite set $\mathbf{C}$ of *concept names*, a finite set $\mathbf{R}$ of *role names*, and a finite set $\mathbf{I}$ of *individual names*. A concept name stands for a unary predicate, a role name stands for a binary predicate, and an individual name stands for a constant. We use letters like A and B for concept names, r and s for role

names, and a and b for individual names. We also refer to A and B as *atomic concepts*, to r and s as *roles*, and to a and b as *individuals*.

An ($\mathcal{SHOQ}$) *RBox* $\mathcal{R}$ is a finite set of role axioms of the form $r \sqsubseteq s$ or $r \circ r \sqsubseteq r$. For example, $link \sqsubseteq path$ and $path \circ path \sqsubseteq path$ are such role axioms.

By $ext(\mathcal{R})$ we denote the least extension of $\mathcal{R}$ such that:

- $r \sqsubseteq r \in ext(\mathcal{R})$ for any role r
- if $r \sqsubseteq r' \in ext(\mathcal{R})$ and $r' \sqsubseteq r'' \in ext(\mathcal{R})$ then $r \sqsubseteq r'' \in ext(\mathcal{R})$.

Let $r \sqsubseteq_{\mathcal{R}} s$ denote $r \sqsubseteq s \in ext(\mathcal{R})$, and $trans_{\mathcal{R}}(r)$ denote $(r \circ r \sqsubseteq r) \in ext(\mathcal{R})$. If $r \sqsubseteq_{\mathcal{R}} s$ then r is a *subrole* of s (w.r.t. $\mathcal{R}$). If $trans_{\mathcal{R}}(s)$ then s is a *transitive role* (w.r.t. $\mathcal{R}$). A role is *simple* (w.r.t. $\mathcal{R}$) if it is neither transitive nor has any transitive subrole (w.r.t. $\mathcal{R}$).

Concepts in $\mathcal{SHOQ}$ are formed using the following BNF grammar, where n is a nonnegative integer and s is a simple role:

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \forall r.C \mid \{a\} \mid \geq n\, s.C \mid \leq n\, s.C$$

A concept stands for a set of individuals. The concept $\top$ stands for the set of all individuals (in the considered domain). The concept $\bot$ stands for the empty set. The constructors $\neg$, $\sqcap$ and $\sqcup$ stand for the set operators: complement, intersection and union. For the remaining forms, we give some examples: $\exists hasChild.Male$, $\forall hasChild.Female$, $\geq 2\, hasChild.Teacher$, $\leq 5\, hasChild.\top$.

We use letters like C and D to denote arbitrary concepts.

A *TBox* is a finite set of axioms of the form $C \sqsubseteq D$ or $C \doteq D$.

An *ABox* is a finite set of *assertions* of the form $a\!:\!C$, $r(a, b)$ or $a \neq b$.

An axiom $C \sqsubseteq D$ means C is a subconcept of D, while $C \doteq D$ means C and D are equivalent concepts. An assertion $a\!:\!C$ means a is an instance of concept C, and $a \neq b$ means a and b are distinct individuals.

A *knowledge base* in $\mathcal{SHOQ}$ is a tuple $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{R}$ is an RBox, $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

We say that a role s is *numeric* w.r.t. a knowledge base $KB = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ if:

- it is simple w.r.t. $\mathcal{R}$ and occurs in a concept $\geq n\, s.C$ or $\leq n\, s.C$ in KB, or
- $s \sqsubseteq_{\mathcal{R}} r$ and r is numeric w.r.t. KB.

We will simply call such an s a numeric role when KB is clear from the context.

A *formula* is defined to be either a concept or an ABox assertion. We use letters like φ, ψ and ξ to denote formulas. Let $\mathsf{null}\!:\!C$ stand for C. We use α to denote either an individual or null. Thus, $\alpha\!:\!C$ is a formula of the form $a\!:\!C$ or $\mathsf{null}\!:\!C$ (which means C).

An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, called the *interpretation function* of $\mathcal{I}$, that maps each concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name r to a binary relation $r^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, and each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation function is extended to complex concepts as follows, where $\sharp Z$

denotes the cardinality of a set Z:

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad \bot^{\mathcal{I}} = \emptyset \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} - C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \qquad \{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$$
$$(\exists r.C)^{\mathcal{I}} = \big\{x \in \Delta^{\mathcal{I}} \mid \exists y\big[\langle x, y\rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\big]\big\}$$
$$(\forall r.C)^{\mathcal{I}} = \big\{x \in \Delta^{\mathcal{I}} \mid \forall y\big[\langle x, y\rangle \in r^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\big]\big\}$$
$$(\geq n\, s.C)^{\mathcal{I}} = \big\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in s^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\big\}$$
$$(\leq n\, s.C)^{\mathcal{I}} = \big\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in s^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\big\}.$$

For a set Γ of concepts, define $\Gamma^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid x \in C^{\mathcal{I}} \text{ for all } C \in \Gamma\}$.

The relational composition of binary relations R_1, R_2 is denoted by $\mathrm{R}_1 \circ \mathrm{R}_2$.

An interpretation $\mathcal{I}$ is a *model of an RBox* $\mathcal{R}$ if for every axiom $r \sqsubseteq s$ (resp. $r \circ r \sqsubseteq r$) of $\mathcal{R}$, we have that $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ (resp. $r^{\mathcal{I}} \circ r^{\mathcal{I}} \subseteq r^{\mathcal{I}}$). Note that if $\mathcal{I}$ is a model of $\mathcal{R}$ then it is also a model of $ext(\mathcal{R})$.

An interpretation $\mathcal{I}$ is a *model of a TBox* $\mathcal{T}$ if for every axiom $C \sqsubseteq D$ (resp. $C \doteq D$) of $\mathcal{T}$, we have that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $C^{\mathcal{I}} = D^{\mathcal{I}}$).

An interpretation $\mathcal{I}$ is a *model of an ABox* $\mathcal{A}$ if for every assertion $a\!:\!C$ (resp. $r(a, b)$ or $a \neq b$) of $\mathcal{A}$, we have that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp. $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in r^{\mathcal{I}}$ or $a^{\mathcal{I}} \neq b^{\mathcal{I}}$).

An interpretation $\mathcal{I}$ is a *model of a knowledge base* $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$ if $\mathcal{I}$ is a model of $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$. A knowledge base $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$ is *satisfiable* if it has a model.

An interpretation $\mathcal{I}$ *satisfies* a concept C (resp. a set X of concepts) if $C^{\mathcal{I}} \neq \emptyset$ (resp. $X^{\mathcal{I}} \neq \emptyset$). It *validates* C if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. A set X of concepts is *satisfiable w.r.t. an RBox* $\mathcal{R}$ *and a TBox* $\mathcal{T}$ if there exists a model of $\mathcal{R}$ and $\mathcal{T}$ that satisfies X. For $X = \mathcal{A} \cup \mathcal{A}'$, where $\mathcal{A}$ is an ABox and $\mathcal{A}'$ is a set of assertions of the form $\neg r(a, b)$ or $a \doteq b$, we say that X is *satisfiable w.r.t. an RBox* $\mathcal{R}$ *and a TBox* $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$ such that: $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \notin r^{\mathcal{I}}$ for all $(\neg r(a, b)) \in \mathcal{A}'$, and $a^{\mathcal{I}} = b^{\mathcal{I}}$ for all $(a \doteq b) \in \mathcal{A}'$. In that case, we also say that $\mathcal{I}$ is a model of $\langle \mathcal{R}, \mathcal{T}, X\rangle$.

3 A Tableau Method for $\mathcal{SHOQ}$

We assume that concepts and ABox assertions are represented in negation normal form (NNF), where $\neg$ occurs only directly before atomic concepts. We use $\overline{C}$ to denote the NNF of $\neg C$, and for $\varphi = (a\!:\!C)$, we use $\overline{\varphi}$ to denote $a\!:\!\overline{C}$. For simplicity, we treat axioms of $\mathcal{T}$ as concepts representing global assumptions: an axiom $C \sqsubseteq D$ is treated as $\overline{C} \sqcup D$, while an axiom $C \doteq D$ is treated as $(\overline{C} \sqcup D) \sqcap (\overline{D} \sqcup C)$. That is, we assume that $\mathcal{T}$ consists of concepts in NNF. Thus, an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ iff $\mathcal{I}$ validates every concept $C \in \mathcal{T}$.

Let $\mathit{EdgeLabels} = \{\mathsf{testingCloseness}, \mathsf{checkingFeasibility}\} \times \mathcal{P}(\mathbf{R}) \times (\mathbf{I} \cup \{\mathsf{null}\})$. For $e \in \mathit{EdgeLabels}$, let $e = \langle \pi_t(e), \pi_r(e), \pi_i(e)\rangle$. Thus, $\pi_t(e)$ is the type of e, $\pi_r(e)$ is a set of roles, and $\pi_i(e)$ is either an individual or null.

We define a tableaux as a rooted graph. Such a graph is a tuple $G = \langle V, E, \nu\rangle$, where V is a set of nodes, $E \subseteq V \times V$ is a set of edges, $\nu \in V$ is the root, each

node $v \in V$ has a number of attributes, and each edge $\langle v, w\rangle$ may have a number of labels from *EdgeLabels*. Attributes of a tableau node v are:

- $\mathit{Type}(v) \in \{\mathsf{state}, \mathsf{non\text{-}state}\}$.
- $\mathit{SType}(v) \in \{\mathsf{complex}, \mathsf{simple}\}$ is called the subtype of v.
- $\mathit{Status}(v) \in \{\mathsf{unexpanded}, \mathsf{p\text{-}expanded}, \mathsf{f\text{-}expanded}, \mathsf{closed}, \mathsf{open}, \mathsf{blocked}\} \cup \{\mathsf{closed\text{-}wrt}(U) \mid U \subseteq V$ and $\mathit{Type}(u) = \mathsf{state} \wedge \mathit{SType}(u) = \mathsf{complex}$ for all $u \in U\}$, where p-expanded and f-expanded mean "partially expanded" and "fully expanded", respectively. $\mathit{Status}(v)$ may be p-expanded only when $\mathit{Type}(v) = \mathsf{state}$. If $\mathit{Status}(v) = \mathsf{closed\text{-}wrt}(U)$ then we say that the node v is closed w.r.t. the nodes from U.
- $\mathit{Label}(v)$ is a finite set of formulas, called the label of v.
- $\mathit{RFmls}(v)$ is a finite set of formulas, called the set of reduced formulas of v.
- $\mathit{IndRepl}(v) : \mathbf{I} \to \mathbf{I}$ is a partial mapping specifying replacements of individuals. It is available only when v is a complex node. If $\mathit{IndRepl}(v)(a) = b$ then, at the node v, we have $a \doteq b$ and b is the representative of its abstract class.
- $\mathit{ILConstraints}(v)$ is a set of integer linear constraints. It is available only when $\mathit{Type}(v) = \mathsf{state}$. The constraints use variables x_e indexed by labels e of edges outgoing from v such that $\pi_t(e) = \mathsf{checkingFeasibility}$. Such a variable specifies how many copies of the successor via e will be created for v.

If $\langle v, w\rangle \in E$ then we call v a *predecessor* of w, and w a *successor* of v. An edge outgoing from a node v has labels iff $\mathit{Type}(v) = \mathsf{state}$. When defined, the set of labels of an edge $\langle v, w\rangle$ is denoted by $\mathit{ELabels}(v, w)$. If $e \in \mathit{ELabels}(v, w)$ then $\pi_i(e) = \mathsf{null}$ iff $\mathit{SType}(v) = \mathsf{simple}$.

A node v is called a *state* if $\mathit{Type}(v) = \mathsf{state}$, and *non-state* otherwise. It is called a *complex node* if $\mathit{SType}(v) = \mathsf{complex}$, and a *simple node* otherwise. The label of a complex node consists of ABox assertions, while the label of a simple node consists of concepts. The root ν is a complex non-state.

A node may have status blocked only when it is a simple node with the label containing a nominal $\{a\}$. The status blocked can be updated only to closed or closed-wrt(...). We write closed-wrt(...) to mean closed-wrt(U) for some U.

The graph G consists of two layers: the layer of complex nodes and the layer of simple nodes. There are no edges from simple nodes to complex nodes. The edges from complex nodes to simple nodes are exactly the edges outgoing from complex states. That is: if $\langle v, w\rangle$ is an edge from a complex node v to a simple node w then $\mathit{Type}(v) = \mathsf{state}$; if $\mathit{Type}(v) = \mathsf{state}$ and $\langle v, w\rangle \in E$ then $\mathit{SType}(w) = \mathsf{simple}$. Each complex node of G is like an ABox (more formally: its label is an ABox), which can be treated as a graph whose vertices are named individuals. On the other hand, a simple node of G stands for an unnamed individual. If e is a label of an edge from a complex state v to a simple node w then the triple $\langle v, e, w\rangle$ can be treated as an edge from the named individual $\pi_i(e)$ (an inner node of the graph representing v) to the unnamed individual corresponding to w, and that edge is via the roles from $\pi_r(e)$.

We will use also assertions of the form $a : (\preceq n\, s.C)$ and $a : (\succeq n\, s.C)$, where s is a numeric role. The difference between $a : (\preceq n\, s.C)$ and $a : (\leq n\, s.C)$ is that, for checking $a : (\preceq n\, s.C)$, we do not have to pay attention to assertions of the

form $s(a,b)$ or $r(a,b)$ with r being a subrole of s. The aim for $a:(\succeq n\, s.C)$ is similar. We use $a:(\preceq n\, s.C)$ and $a:(\succeq n\, s.C)$ only as syntactic representations of some expressions, and do not provide semantics for them. We define

$$\texttt{FullLabel}(v) = \mathit{Label}(v) \cup \mathit{RFmls}(v) - \{\text{formulas of the form } a\colon(\preceq n\, s.C) \text{ or } a\colon(\succeq n\, s.C)\}.$$

We apply global caching: if $v_1, v_2 \in V$, $\mathit{Label}(v_1) = \mathit{Label}(v_2)$ and ($\mathit{SType}(v_1) = \mathit{SType}(v_2) =$ simple or ($\mathit{SType}(v_1) = \mathit{SType}(v_2) =$ complex and $\mathit{Type}(v_1) = \mathit{Type}(v_2)$)) then $v_1 = v_2$. Due to global caching, an edge outgoing from a state may have a number of labels as the result of merging edges.

We say that a node v may affect the status of the root ν if there exists a path consisting of nodes $v_0 = \nu, v_1, \ldots, v_{n-1}, v_n = v$ such that, for every $0 \leq i < n$, $\mathit{Status}(v_i)$ differs from open and closed, and if it is closed-wrt(U) then U is disjoint from $\{v_0, \ldots, v_i\}$. In that case, if $u \in \{v_1, \ldots, v_n\}$ then we say that v may affect the status of the root ν via a path through u.

From now on, let $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$ be a knowledge base in NNF of the logic $\mathcal{SHOQ}$, with $\mathcal{A} \neq \emptyset$.[4] In this section we present a tableau calculus $C_{\mathcal{SHOQ}}$ for checking satisfiability of $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$. A $C_{\mathcal{SHOQ}}$-tableau for $\langle \mathcal{R}, \mathcal{T}, \mathcal{A}\rangle$ is a rooted graph $G = \langle V, E, \nu\rangle$ constructed as follows:

Initialization: $V := \{\nu\}$, $E := \emptyset$, $\mathit{Type}(\nu) :=$ non-state, $\mathit{SType}(\nu) :=$ complex, $\mathit{Status}(\nu) :=$ unexpanded, $\mathit{RFmls}(\nu) := \emptyset$, $\mathit{Label}(\nu) := \mathcal{A} \cup \{(a:C) \mid C \in \mathcal{T}$ and a is an individual occurring in $\mathcal{A}$ or $\mathcal{T}\}$, for each individual a occurring in $\mathit{Label}(\nu)$ set $\mathit{IndRepl}(\nu)(a) := a$.

Rules' Priorities and Expansion Strategies: The graph is then expanded by the following rules, which are specified in detail in [12]:

(UPS) rules for updating statuses of nodes,
(US) unary static expansion rules,
(DN) a rule for dealing with nominals,
(NUS) a non-unary static expansion rule,
(FS) the forming-state rule,
(TP) a transitional partial-expansion rule,
(TF) a transitional full-expansion rule.

Each of the rules is parametrized by a node v. We say that a rule is *applicable* to v if it can be applied to v to make changes to the graph. The rule (UPS) has a higher priority than (US), which has a higher priority than the remaining rules in the list. If neither (UPS) nor (US) is applicable to any node, then choose a node v with status unexpanded or p-expanded, choose the first rule applicable to v among the rules in the last five items of the above list, and apply it to v. Any strategy can be used for choosing v, but it is worth to choose v for expansion only when v may affect the status of the root ν of the graph. Note that the priorities of the rules are specified by the order in the above list, but the rules (UPS) and (US) are checked globally (technically, they are triggered immediately when possible), while the remaining rules are checked for a chosen node.

[4] If $\mathcal{A}$ is empty, we can add $a:\top$ to it, where a is a special individual.

Termination: The construction of the graph ends when the root ν receives status closed or open or when no more changes that may affect the status of ν can be made[5].

To check satisfiability of $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ one can construct a $C_{\mathcal{SHOQ}}$-tableau for it, then return "no" when the root of the tableau has status closed, or "yes" in the other case. It can be proved that (see [12]):

- A $C_{\mathcal{SHOQ}}$-tableau for $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ can be constructed in ExpTime.
- If $G = \langle V, E, \nu \rangle$ is an arbitrary $C_{\mathcal{SHOQ}}$-tableau for $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ then $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ is satisfiable iff $\mathit{Status}(\nu) \neq \mathsf{closed}$.

Remark 3.1. Our technique for dealing with quantified number restrictions is similar to Nguyen's technique used for $\mathcal{SHIQ}$ [10, 11]. There are some technical differences, which are caused by that we use global caching for $\mathcal{SHOQ}$, while Nguyen's work [10] uses global state caching for $\mathcal{SHIQ}$ (due to inverse roles) and inverse roles can interact with quantified number restrictions.

We briefly explain our technique for dealing with nominals. Suppose v is a simple node with $\mathit{Status}(v) \notin \{\mathsf{closed}, \mathsf{open}\}$ and $\{a\} \in \mathit{Label}(v)$, a complex state u is an ancestor of v, and v may affect the status of the root ν via a path through u. Let u_0 be a predecessor of u. The node u_0 has only u as a successor and it was expanded by the forming-state rule. There are three cases:

- If, for every $C \in \mathit{Label}(v)$, the formula obtained from $a{:}C$ by replacing every individual b by $\mathit{IndRepl}(u)(b)$ belongs to $\mathtt{FullLabel}(u)$, then v is "consistent" with u.
- If there exists $C \in \mathit{Label}(v)$ such that the formula obtained from $a : \overline{C}$ by replacing every individual b by $\mathit{IndRepl}(u)(b)$ belongs to $\mathtt{FullLabel}(u)$, then v is "inconsistent" with u. In this case, if $\mathit{Status}(v)$ is of the form $\mathsf{closed\text{-}wrt}(U)$ then we update it to $\mathsf{closed\text{-}wrt}(U \cup \{u\})$, else we update it to $\mathsf{closed\text{-}wrt}(\{u\})$.
- In the remaining case, the node u is "incomplete" w.r.t. v, which means that the expansion of u_0 was not appropriate. Thus, we delete the edge $\langle u_0, u \rangle$ and re-expand u_0 by an appropriate "or"-branching (see [12]).

There are also treatments for dealing with assertions of the form $a{:}\{b\}$ and for updating statuses of nodes in the presence of $\mathsf{closed\text{-}wrt}(\ldots)$. □

4 Illustrative Examples

Example 4.1. Let us construct a $C_{\mathcal{SHOQ}}$-tableau for $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where

$$\mathcal{A} = \{a{:}A,\ a{:}\exists r.\exists r.(A \sqcup \{a\}),\ a{:}\geq 3\, r.\forall r.\neg A,\ a{:}\forall r.B,\ a{:}\leq 3\, r.B,\\ r(a,b),\ b{:}\forall r.\neg A,\ b{:}(\forall r.(\neg A \sqcap \neg\{a\}) \sqcup \neg B)\},$$

$\mathcal{R} = \emptyset$ and $\mathcal{T} = \emptyset$. An illustration is presented in Figure 1.

[5] That is, ignoring nodes that are unreachable from ν via a path without nodes with status closed or open, no more changes can be made to the graph.

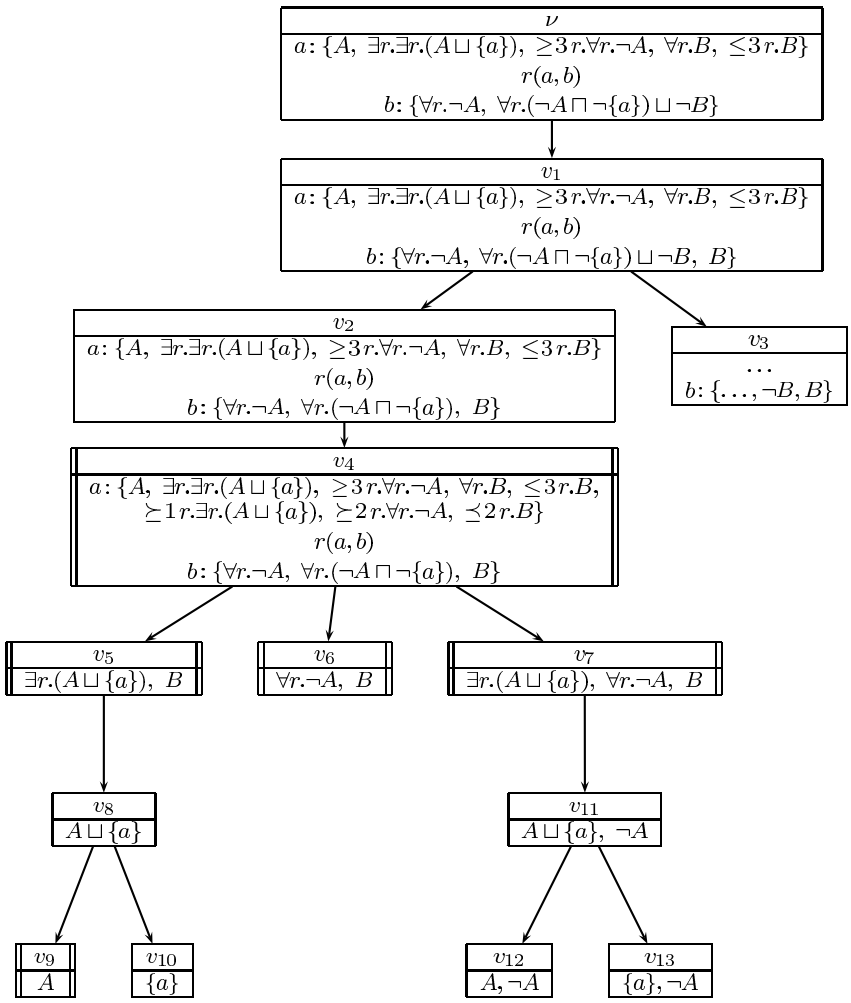


Fig. 1. An illustration of the tableau described in Example 4.1. The marked nodes v_4 – v_7 and v_9 are states. The nodes ν, v_1 – v_4 are complex nodes, the remaining are simple nodes. In each node, we display the formulas of its label.

At the beginning, the graph has only the root ν which is a complex non-state with $\mathit{Label}(\nu) = \mathcal{A}$. Since $\{a : \forall r.B, r(a,b)\} \subset \mathit{Label}(\nu)$, applying a unary static expansion rule to ν, we connect it to a new complex non-state v_1 with $\mathit{Label}(v_1) = \mathit{Label}(\nu) \cup \{b : B\}$.

Since $b : (\forall r.(\neg A \sqcap \neg\{a\}) \sqcup \neg B) \in \mathit{Label}(v_1)$, applying the non-unary static expansion rule to v_1, we connect it to new complex non-states v_2 and v_3 with

$$\mathit{Label}(v_2) = \mathit{Label}(v_1) - \{b : (\forall r.(\neg A \sqcap \neg\{a\}) \sqcup \neg B)\} \cup \{b : \forall r.(\neg A \sqcap \neg\{a\})\}$$
$$\mathit{Label}(v_3) = \mathit{Label}(v_1) - \{b : (\forall r.(\neg A \sqcap \neg\{a\}) \sqcup \neg B)\} \cup \{b : \neg B\}.$$

Since both $b : B$ and $b : \neg B$ belong to $Label(v_3)$, the node v_3 receives status closed. Applying the forming-state rule to v_2, we connect it to a new complex state v_4 with

$$Label(v_4) = Label(v_2) \cup \{a : \succeq 1\, r.\exists r.(A \sqcup \{a\}),\ a : \succeq 2\, r.\forall r.\neg A,\ a : \preceq 2\, r.B\}.$$

The assertion $a : \succeq 1\, r.\exists r.(A \sqcup \{a\}) \in Label(v_4)$ is due to $a : \exists r.\exists r.(A \sqcup \{a\}) \in Label(v_2)$ and the fact that the negation of $b : \exists r.(A \sqcup \{a\})$ in NNF belongs to $Label(v_2)$ (notice that $r(a, b) \in Label(v_2)$). The assertion $a : \succeq 2\, r.\forall r.\neg A \in Label(v_4)$ is due to $a : \geq 3\, r.\forall r.\neg A \in Label(v_2)$ and the fact that $\{r(a, b), b : \forall r.\neg A\} \subset Label(v_2)$. Similarly, the assertion $a : \preceq 2\, r.B \in Label(v_4)$ is due to $a : \leq 3\, r.B \in Label(v_2)$ and the fact $\{r(a, b), b : B\} \subset Label(v_2)$.

As r is a numeric role, applying the transitional partial-expansion rule[6] to v_4, we just change the status of v_4 to p-expanded. After that, applying the transitional full-expansion rule to v_4, we connect it to new simple non-states v_5, v_6, v_7 using edges labeled by $e_{4,5}$, $e_{4,6}$, $e_{4,7}$, respectively, such that $e_{4,i} = \langle \mathsf{checkingFeasibility}, \{r\}, a\rangle$ for $5 \leq i \leq 7$, and $Label(v_5) = \{\exists r.(A \sqcup \{a\}),\ B\}$, $Label(v_6) = \{\forall r.\neg A,\ B\}$, $Label(v_7) = \{\exists r.(A \sqcup \{a\}),\ \forall r.\neg A,\ B\}$. The creation of v_5 is caused by $a : \succeq 1\, r.\exists r.(A \sqcup \{a\}) \in Label(v_4)$, while the creation of v_6 is caused by $a : \succeq 1\, r.\forall r.\neg A$. The node v_7 results from merging v_5 and v_6. Furthermore, $ILConstraints(v_4)$ consists of $x_{e_{4,i}} \geq 0$, for $5 \leq i \leq 7$, and

$$\begin{aligned} x_{e_{4,5}} + x_{e_{4,7}} &\geq 1 \\ x_{e_{4,6}} + x_{e_{4,7}} &\geq 2 \\ x_{e_{4,5}} + x_{e_{4,6}} + x_{e_{4,7}} &\leq 2. \end{aligned}$$

Applying the forming-state rule to v_5, the type of this node is changed from non-state to state. Next, applying the transitional partial-expansion rule to v_5, its status is changed to p-expanded. Then, applying the transitional full-expansion rule to v_5, we connect v_5 to a new simple non-state v_8 with $Label(v_8) = \{A \sqcup \{a\}\}$ using an edge labeled by $e_{5,8}$ and set $ILConstraints(v_5) = \{x_{e_{5,8}} \geq 0, x_{e_{5,8}} \geq 1\}$.

Applying the non-unary static expansion rule to v_8, we connect it to new simple non-states v_9 and v_{10} with $Label(v_9) = \{A\}$ and $Label(v_{10}) = \{\{a\}\}$. The status of v_9 is then changed to open, which causes the statuses of v_8 and v_5 to be updated to open. The node v_{10} is not expanded as it does not affect the status of the root node ν.

Applying the forming-state rule to v_6, the type of this node is changed from non-state to state. Next, applying the transitional partial-expansion rule and then the transitional full-expansion rule to v_6, its status is changed to f-expanded. The status of v_6 is then updated to open.

Applying the forming-state rule to v_7, the type of this node is changed from non-state to state. Next, applying the transitional partial-expansion rule to v_7, its status is changed to p-expanded. Then, applying the transitional full-expansion rule to v_7, we connect v_7 to a new simple non-state v_{11} with $Label(v_{11}) = \{A \sqcup$

[6] which is used for making transitions via non-numeric roles

$\{a\}$, $\neg A\}$ using an edge labeled by $e_{7,11}$ and set $\mathit{ILConstraints}(v_7) = \{x_{e_{7,11}} \geq 0$, $x_{e_{7,11}} \geq 1\}$.

Applying the non-unary static expansion rule to v_{11}, we connect it to new simple non-states v_{12} and v_{13} with $\mathit{Label}(v_{12}) = \{A, \neg A\}$ and $\mathit{Label}(v_{13}) = \{\{a\}, \neg A\}$. The status of v_{12} is then changed to closed. Since $a : A \in \mathit{Label}(v_4)$, the status of v_{13} is updated to closed-wrt($\{v_4\}$), which causes the status of v_{11} to be updated also to closed-wrt($\{v_4\}$). As the set $\mathit{ILConstraints}(v_7) \cup \{x_{e_{7,11}} = 0\}$ is infeasible, the status of v_7 is updated to closed-wrt($\{v_4\}$). Next, as the set $\mathit{ILConstraints}(v_4) \cup \{x_{e_{4,7}} = 0\}$ is infeasible, the status of v_4 is first updated to closed-wrt($\{v_4\}$) and then updated to closed. After that, the statuses of v_2, v_1, ν are sequentially updated to closed. Thus, we conclude that the knowledge base $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable. □

Example 4.2. Let us modify Example 4.1 by deleting the assertion $a : A$ from the ABox. That is, we are now constructing a $C_{\mathcal{SHOQ}}$-tableau for $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where

$$\mathcal{A} = \{a : \exists r.\exists r.(A \sqcup \{a\}),\ a : \geq 3\, r.\forall r.\neg A,\ a : \forall r.B,\ a : \leq 3\, r.B,$$
$$r(a, b),\ b : \forall r.\neg A,\ b : (\forall r.(\neg A \sqcap \neg\{a\}) \sqcup \neg B)\},$$

$\mathcal{R} = \emptyset$ and $\mathcal{T} = \emptyset$. The first stage of the construction is similar to the one of Example 4.1, up to the step of updating the status of v_{12} to closed. This stage is illustrated in Figure 2, which is similar to Figure 1 except that the labels of the nodes ν and v_1 – v_4 do not contain $a : A$. The continuation is described below and illustrated by Figure 3.

Since $\mathit{Label}(v_{13}) = \{\{a\}, \neg A\}$, applying the rule for dealing with nominals to v_{13}, we delete the edge $\langle v_2, v_4 \rangle$ (from E) and re-expand v_2 by connecting it to new complex non-states v_{14} and v_{15} with $\mathit{Label}(v_{14}) = \mathit{Label}(v_2) \cup \{a : \neg A\}$ and $\mathit{Label}(v_{15}) = \mathit{Label}(v_2) \cup \{a : A\}$ as shown in Figure 3. The status of v_{13} is updated to blocked. The node v_4 is not deleted, but we do not display it in Figure 3.

Applying the forming-state rule to v_{14} we connect it to a new complex state v_{16}. The label of v_{16} is computed using $\mathit{Label}(v_{14})$ in a similar way as in Example 4.1 when computing $\mathit{Label}(v_4)$.

Applying the transitional partial-expansion rule to v_{16} we change its status to p-expanded. After that, applying the transitional full-expansion rule to v_{16} we connect it to the existing nodes v_5, v_6, v_7 by using edges labeled by $e_{16,5}$, $e_{16,6}$, $e_{16,7}$, respectively, which are the same tuple $\langle \mathsf{checkingFeasibility}, \{r\}, a \rangle$. The set $\mathit{ILConstraints}(v_{16})$ consists of $x_{e_{16,i}} \geq 0$, for $5 \leq i \leq 7$, and

$$x_{e_{16,5}} + x_{e_{16,7}} \geq 1$$
$$x_{e_{16,6}} + x_{e_{16,7}} \geq 2$$
$$x_{e_{16,5}} + x_{e_{16,6}} + x_{e_{16,7}} \leq 2.$$

Applying the forming-state rule to v_{15} we connect it to a new complex state v_{17}. The label of v_{17} is computed using $\mathit{Label}(v_{15})$ in a similar way as in Example 4.1 when computing $\mathit{Label}(v_4)$.

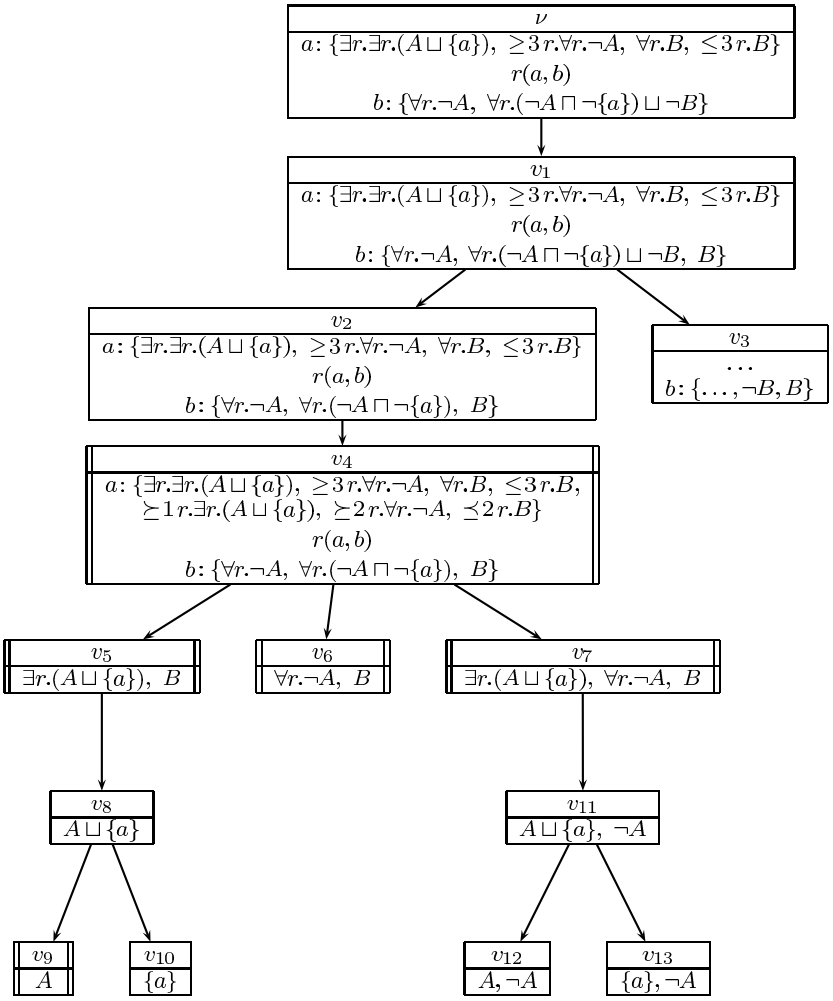


Fig. 2. An illustration for Example 4.2 – Part I.

The expansion of v_{17} is similar to the expansion of v_{16}. The set $\mathit{ILConstraints}(v_{17})$ is like $\mathit{ILConstraints}(v_{16})$, with the subscripts 16 replaced by 17. Analogously to updating the statuses of the nodes v_{13}, v_{11}, v_7 in Example 4.1 to $\mathsf{closed\text{-}wrt}(\{v_4\})$, the statuses of v_{13}, v_{11}, v_7 are updated to $\mathsf{closed\text{-}wrt}(\{v_{17}\})$. Next, as $\mathit{ILConstraints}(v_{17}) \cup \{x_{e_{17,7}} = 0\}$ is infeasible, the status of v_{17} is first updated to $\mathsf{closed\text{-}wrt}(\{v_{17}\})$ and then updated to closed. After that, the status of v_{15} is also updated to closed. As no more changes that may affect the status of ν can be made and $\mathit{Status}(\nu) \neq \mathsf{closed}$, we conclude that the knowledge base $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ is satisfiable. □

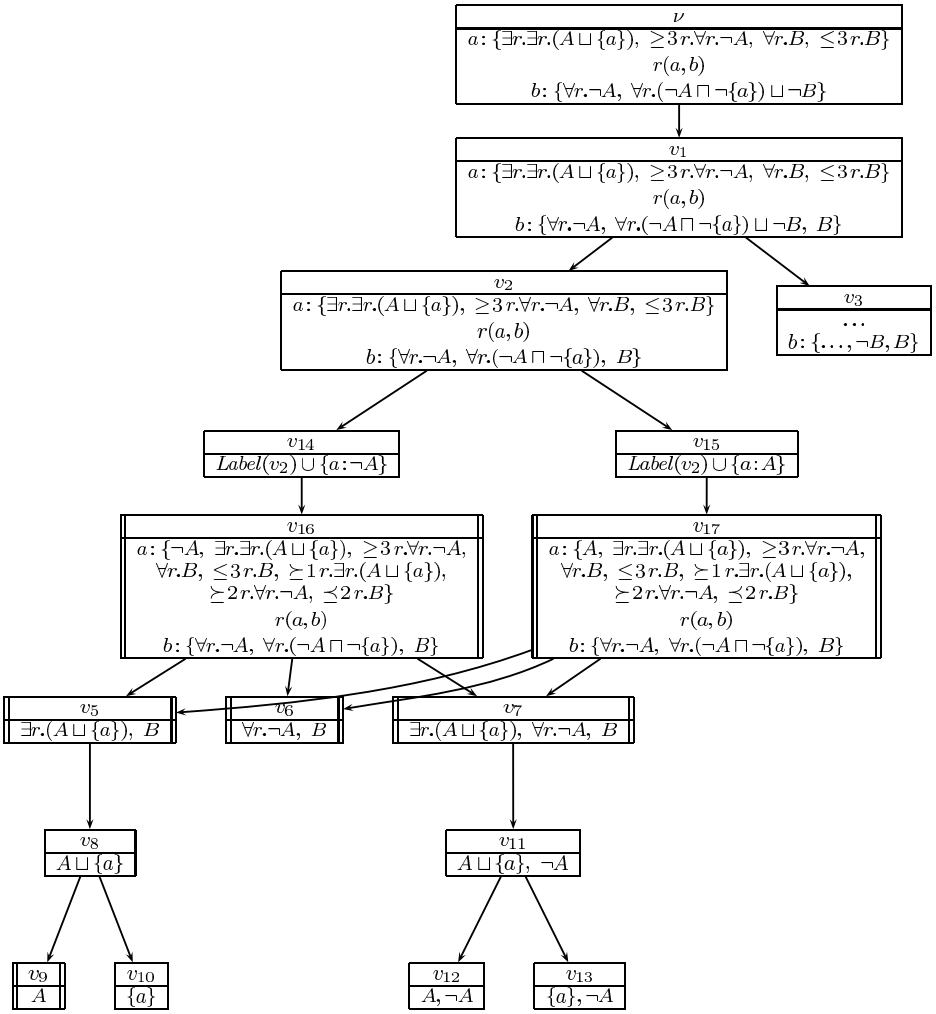


Fig. 3. An illustration for Example 4.2 – Part II.

5 Conclusions

We have presented the first tableau method with an ExpTime (optimal) complexity for checking satisfiability of a knowledge base in the DL $\mathcal{SHOQ}$. The complexity is measured using binary representation for numbers. Our detailed tableau decision procedure for $\mathcal{SHOQ}$ is given in [12].

This work differs from Nguyen's work [10] on $\mathcal{SHIQ}$ in that nominals are allowed instead of inverse roles. Without inverse roles, global caching is used instead of global state caching to allow more cache hits. To deal with nominals, we use additional statuses closed-wrt(...) for nodes of the graph to be constructed.

Acknowledgments. This work was supported by Polish National Science Centre (NCN) under Grants No. 2011/01/B/ST6/02759 (for the first author) and 2011/02/A/HS1/00395 (for the second author).

References

1. J. Faddoul and V. Haarslev. Algebraic tableau reasoning for the description logic SHOQ. *J. Applied Logic*, 8(4):334–355, 2010.
2. N. Farsiniamarj. Combining integer programming and tableau-based reasoning: a hybrid calculus for the description logic SHQ. Master's thesis, Concordia University, 2008.
3. R. Goré and L.A. Nguyen. ExpTime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *Proceedings of TABLEAUX 2007*, volume 4548 of *LNAI*, pages 133–148. Springer, 2007.
4. R. Goré and L.A. Nguyen. Exptime tableaux for ALC using sound global caching. *J. Autom. Reasoning*, 50(4):355–381, 2013.
5. R. Goré and F. Widmann. Sound global state caching for $\mathcal{ALC}$ with inverse roles. In M. Giese and A. Waaler, editors, *Proceedings of TABLEAUX 2009*, volume 5607 of *LNCS*, pages 205–219. Springer, 2009.
6. R. Goré and F. Widmann. Optimal and cut-free tableaux for propositional dynamic logic with converse. In J. Giesl and R. Hähnle, editors, *Proceedings of IJCAR 2010*, volume 6173 of *LNCS*, pages 225–239. Springer, 2010.
7. J. Hladik and J. Model. Tableau systems for SHIO and SHIQ. In *Proceedings of DL'2004*, volume 104 of *CEUR Workshop Proceedings*, pages 168–177, 2004.
8. I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of IJCAI'2001*, pages 199–204. Morgan Kaufmann, 2001.
9. L.A. Nguyen. A cut-free ExpTime tableau decision procedure for the description logic SHI. In *Proceedings of ICCCI'2011 (1)*, volume 6922 of *LNCS*, pages 572–581. Springer, 2011 (see also the long version `http://arxiv.org/abs/1106.2305v1`).
10. L.A. Nguyen. ExpTime tableaux for the description logic SHIQ based on global state caching and integer linear feasibility checking. arXiv:1205.5838, 2012.
11. L.A. Nguyen. A tableau method with optimal complexity for deciding the description logic SHIQ. In *Proceedings of ICCSAMA'2013*, volume 479 of *Studies in Computational Intelligence*, pages 331–342. Springer, 2013.
12. L.A. Nguyen and J. Golińska-Pilarek. A long version of the current paper. `http://www.mimuw.edu.pl/~nguyen/shoq-long.pdf`, 2013.
13. J.Z. Pan and I. Horrocks. Reasoning in the SHOQ(D_n) description logic. In *Proc. of DL'2002*, volume 53 of *CEUR Workshop Proceedings*, pages 53–62, 2002.
14. V.R. Pratt. A near-optimal method for reasoning about action. *J. Comp. Syst. Sci.*, 20(2):231–254, 1980.
15. S. Tobies. *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, RWTH-Aachen, 2001.
16. `http://owl.cs.manchester.ac.uk/navigator/`.

SMT versus Genetic Algorithms: Concrete Planning in the Planics Framework*

Extended Abstract

Artur Niewiadomski[1], Wojciech Penczek[1,2], and Jarosław Skaruz[1]

[1] ICS, Siedlce University, 3-Maja 54, 08-110 Siedlce, Poland
artur@ii.uph.edu.pl, jaroslaw.skaruz@uph.edu.pl
[2] ICS, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland
penczek@ipipan.waw.pl

Abstract. The paper deals with the concrete planning problem (CPP) – a stage of the Web Service Composition (WSC) in the PlanICS framework. A novel SMT-based approach to CPP is defined and its performance is compared to the standard Genetic Algorithm (GA) in the framework of the PlanICS system. The discussion of both the approaches is supported by extensive experimental results.

Keywords: Web Service Composition, SMT, GA, Concrete Planning

1 Introduction

The main concept of Service-Oriented Architecture (SOA) [2] consists in using independent components available via well-defined interfaces. Typically, a composition of web services need to be executed to realize the user objective. The problem of finding such a composition is hard and known as the WSC problem [2, 1, 11]. In this paper, we follow the approach of our system PlanICS [5, 6], which has been inspired by [1].

The main assumption in PlanICS is that all the web services in the domain of interest as well as the objects that are processed by the services, can be strictly classified in a hierarchy of *classes*, organised in an *ontology*. Another key idea is to divide the planning into several stages. The first phase deals with *classes of services*, where each class represents a set of real-world services, while the others work in the space of *concrete services*. The first stage produces an *abstract plan* composed of service classes [9]. Next, offers are retrieved by the offer collector (OC) (a module of PlanICS) and used by in a concrete planning (CP). As a result of CP a *concrete plan* is obtained, which is a sequence of offers satisfying predefined optimization criteria. Such an approach enables to reduce dramatically the number of web services to be considered, and inquired for offers. This paper deals with the concrete planning realised by SMT- and GA-based planners.

* This work has been supported by the National Science Centre under the grant No. 2011/01/B/ST6/01477.

While CPP has been extensively studied in the literature as shown by numerous papers concerning an application of GA, the main contribution of our paper is an application of an SMT-based planner for finding optimal concrete plans. Such an approach based on SMT-solvers is quite promising and competitive comparing to applications of GA. The second contribution is a comparison of SMT-based approach performance with the results obtained from GA. While dealing with very large state spaces, an SMT-solver may be time demanding, but its advantage is demonstrated in finding always optimal concrete plans. Since a planner based on GA, being quite fast, may have difficulties in finding optimal concrete plans, we find both the approaches complementary.

In the last few years the CPP has been extensively studied in the literature. In [4] a simple GA was used to obtain a good quality concrete plan. In [12] CPP was transformed to a multicriteria optimization problem and GA was used to find a concrete plan. However, the authors present the experiments on a relatively small search space that could not provide valuable conclusions. Our paper fills the gap by presenting the results that allow to examine scalability of the algorithms and their efficiency when the large search space is considered.

Most of the applications of SMT in the domain of WSC is related to the automatic verification and testing. For example, a message race detection problem is investigated in [7], [3] deals with a service substitutability problem, while [8] exploits SMT to verification of WS-BPEL specifications against business rules. However, to the best of our knowledge, there are no other approaches dealing with SMT-solvers as an engine to WSC.

The rest of the paper is organized as follows. Sect. 2 deals with CPP in PlanICS. Sect. 3 presents our SMT-based approach to solve CPP. Sect. 4 discusses experimental results compared to results of the standard GA, while the last section summarizes the results.

2 Concrete Planning Problem

This section defines CPP as the third stage of the WSC in PlanICS framework and provides the basic definitions. We introduce the main ideas behind PlanICS and define CPP as a constrained optimization problem. PlanICS is a system implementing our original approach which solves the composition problem in clearly separated stages. An ontology, managed by the *ontology provider*, contains a system of *classes* describing the types of the services as well as the types of the objects they process [10]. A class consists of a unique name and a set of the attributes. By an *object* we mean an instance of a class. Below, we present a simplified ontology, used further as a running example.

Example 1 (Ontology). Consider a simple ontology describing a fragment of some financial market consisting of service types, inheriting from the class $Investment$, representing various types of financial instruments, and three object types: $Money$ having the attribute *amount*, $Transaction$ having the two attributes *amount* and $profit$, and $Charge$ having the attribute fee. Suppose that each investment

service takes m - an instance of $Money$ as input, produces t and c - instances of $Transaction$ and $Charge$, and updates the amount of money remaining after the operation, i.e, the attribute $m.amount$.

Two fundamental concepts of PlanICS are *worlds* and *world transformations*.

Definition 1 (World, Abstract World). *Let $\mathcal{D} = \mathbb{Z} \cup \mathbb{R} \cup \mathbb{A}$, where $\mathbb{Z}$ is the set of integers, $\mathbb{R}$ is the set of real numbers, and $\mathbb{A} = \{\mathbf{set}, \mathbf{null}\}$ is the set of* abstract values*. Let $\mathcal{O}$ be the set of all objects, $\mathcal{A}$ be the set of all attributes, and $attr : \mathcal{O} \mapsto 2^{\mathcal{A}}$ be the function returning the attributes of an object. A* world *is a pair (O, val), where $O \subseteq \mathcal{O}$ is a set of objects and $val : O \times attr(O) \mapsto \mathcal{D}$ is a valuation function, which to every attribute of the objects from O assigns a value from a respective domain or* **null***, if the attribute does not have a value. A world is* abstract *if all its object attributes have values from $\mathbb{A}$.*

Since the main part of this paper deals with an optimization problem, the domains under consideration are the integer and the real numbers[3]. PlanICS uses a state-based approach in which the worlds represent 'snapshots' of the reality, while the services transform them. A *transformation* of a world w by a service of type s into a world w', denoted by $w \xrightarrow{s} w'$, consists in processing a subset of w, by changing values of object attributes and/or adding new objects, according to the specification of s [10]. Often, a world w can be transformed by s in more than one way. For example when w contains multiple objects of the same type and one can designate more than one subset which can be processed by s. Thus, we define a *transformation context* cx as a mapping from the objects of the input and output of s to the objects of w. The transformation of w by s in the context cx to w' is denoted by $w \xrightarrow{s,cx} w'$ [9].

The user expresses a goal by a *query*, referring to objects and their attributes, and adding constraints while defining *initial worlds* to start with and *expected worlds* to be reached. Composition is thus understood as searching for a set of services capable to process certain states in a desired way, that is, transforming a subset of an initial world into a superset of an expected world (called a *final world*). This is obtained by executing services according to a *plan*.

A specification of a user query consists of the following components: three sets of objects IN, IO, and OU, two boolean formulas Pre (over $IN \cup IO$) and $Post$ (over $IN \cup IO \cup OU$) specifying the initial and the expected worlds, resp., a set of aggregate conditions and a set of quality expressions, to be defined later. The objects of IN are read-only, these of IO can change values of their attributes, and the objects of OU are produced in subsequent transformations. The Pre and $Post$ formulas define two families of valuation functions V_{Pre} and V_{Post}, determining values of objects from the initial and the expected worlds, resp. A *set of worlds* is defined by a pair composed of a set of objects and a family of valuation functions. In general, there are three main cases, when Pre or $Post$ formula defines a family of valuation functions instead of a single function.

[3] Note that other types of values used in PlanICS framework, like strings, dates, and Boolean values can be easily coded by integers.

That is, when a formula contains: (i) an alternative, (ii) constraints that can be satisfied by more than one valuation, or (iii) the formula does not specify values of some attributes. In order to define a user query in a formal way, we need to define the aggregate conditions and the quality expressions which, contrary to the *Pre* and *Post* formulas, are evaluated over final worlds, so they can take into account also objects not foreseen by the user, but created as by-products of the transformations leading to the final worlds.

Definition 2 (Aggregate conditions, Quality expressions). *A quality expression is a tuple* $(cl, sel, form, type)$*, where cl is an object type (a class from the ontology), sel is a boolean formula over attributes belonging to cl,* $form$ *is a real valued expression (built using standard arithmetic operators, like addition, subtraction, multiplication and division) over attributes of class cl, and* $type \in \{Sum, Min, Max\}$*. An aggregate condition is a tuple* $(cl, sel, form, type, \sim, lim)$*, where the first four components are defined as above,* $\sim \in \{<, \leq, =, \neq, >, \geq\}$ *is a comparison operator, and* $lim \in \mathbb{R}$*. A set of aggregate conditions is denoted by Aggr, while a set of quality expressions is denoted by Qual.*

The purpose of *Qual* is to specify criteria of the best plan, while *Aggr* is used in order to add sophisticated restrictions on the resulting plan. Their interpretation is the following. In order to evaluate a single aggregate condition or a quality expression, first we need to separate a subset of a final world containing the objects of type *cl* only. Next, we restrict this subset to the objects satisfying *sel* condition. Then, for each object from the remaining set we compute the value of $form$ expression. Finally, the aggregation function *type* is applied to the obtained set of values and as a result we get a single (real) value. In the case of an aggregate condition, the obtained value is compared with lim constant, using the provided operator $\sim$, and as a result we get a boolean value.

Example 2 (Query specification). Consider the ontology from Example 1. Assume that the user would like to invest up to \$100 in three financial instruments, but he wants to locate more than \$50 in two investments. The above is expressed by: $IN = \emptyset$, $IO = \{m : Money\}$, $OU = \{t_1, t_2, t_3 : Transaction\}$, $Pre = (m.amount \leq 100)$, and $Post = (t_1.amount + t_2.amount > 50)$. The best plan is clearly this which is the most profitable, i.e., the user wants to maximize the sum of profits. Moreover, he wants to use only services of handling fees less than \$3. The above conditions are expressed by the following aggregate condition and the quality expression: $Aggr = \{(Charge, true, fee, Max, <, 3)\}$, and $Qual = \{(Transaction, true, profit, Sum)\}$.

Formally, a user query is defined as follows:

Definition 3 (User query). *A user query is a tuple* $(W^I, W^E, Aggr, Qual)$*, where* $W^I = (IN \cup IO, V_{Pre})$ *and* $W^E = (IN \cup IO \cup OU, V_{Post})$ *are sets of initial and expected worlds, respectively, Aggr is a set of aggregate conditions, and Qual is a set of quality expressions.*

In the first stage of composition an *abstract planner* matches services at the level of input/output types and the abstract values. The result of this stage is a

Context Abstract Plan (CAP, for short), to be defined after introducing auxiliary definitions. At this planning stage it is enough to know if an attribute does have a value, so we abstract from the concrete values of the object attributes [9], using the following definition.

Definition 4 (World correspondence). *Let $w = (O, val)$ be a world and $w' = (O, val')$ be an abstract world. We say that w'* corresponds *to w iff for every $o \in O$ and for every $a \in attr(o)$* $val'(o,a) = \begin{cases} \textbf{set}, & \text{for } val(o,a) \neq \textbf{null}, \\ \textbf{null}, & \text{for } val(o,a) = \textbf{null}. \end{cases}$

In order to compose services, we define the transformation sequences.

Definition 5 (Transformation sequence). *Let k be a natural number and $seq = \big((s_1, cx_1), \ldots, (s_k, cx_k)\big)$ be a sequence of length k, where s_i is a service type and cx_i is a transformation context for $i = 1, \ldots, k$. We say that a world w_0* is transformed by *seq into a world w_k iff there exists a sequence of worlds $(w_1, w_2, \ldots, w_{k-1})$ such that $\forall_{1 \leq i \leq k}\ w_{i-1} \xrightarrow{s_i, cx_i} w_i$. A sequence seq is called a* transformation sequence*, if there are two worlds w, w' such that w is transformed by seq into w'. The world w' is called a* final world *of seq.*

Finally, we are in a position to define the result of the abstract planning phase.

Definition 6 (Abstract solution, CAP). *Given a transformation sequence seq and a user query $q = (W^I, W^E, Aggr, Qual)$. We say that seq is an abstract solution for q iff for some $w_0 \in W^I$, $w_k \in W^E$, there are abstract worlds w_I, w_F, such that w_I corresponds to w_0 and w_F corresponds to w_k and w_I is transformed by seq into w_F. A CAP for a query q is a pair $CAP_q = (seq, w_F)$, where seq is an abstract solution for q and w_F is a final world of seq.*

Thus, each CAP (seq, w_F) contains information on the service types, the context mappings, and a final world of *seq*. Note that using CAP, the ontology, and the user query we are able to reproduce all the worlds of the transformation sequence. A sequence *seq* is just a representative of a class of equivalent sequences [9, 10].

Collecting offers. In the second planning stage CAP is used by an *offer collector* (OC), i.e., a tool which in cooperation with the service registry queries real-world services. The service registry keeps an evidence of real-world web services, registered accordingly to the service type system. During the registration the service provider defines a mapping between input/output data of the real-world service and the object attributes processed by the declared service type. OC communicates with the real-world services of types present in a CAP, sending the constraints on the data, which can potentially be sent to the service in an inquiry, and on the data expected to be received in an offer in order to keep on building a potential plan. Usually, each service type represents a set of real-world services. Moreover, querying a single service can result in a number of offers. Thus, we define an offer set as a result of the second planning stage.

Definition 7 (Offer, Offer set). *Assume that the n-th instance of a service type from a CAP processes some number of objects having in total m attributes. A single offer collected by OC is a vector $P = [v_1, v_2, \ldots, v_m]$, where v_j is a value of a single object attribute from the n-th intermediate world of the CAP.*

An offer set O^n is a $k \times m$ matrix, where each row corresponds to a single offer and k is the number of offers in the set. Thus, the element $o^n_{i,j}$ is the j-th value of the i-th offer collected from the n-th service type instance from the CAP.

Example 3 (Offer, Offer sets). Consider the user query from Example 2 and an exemplary CAP consisting of three instances of $Investment$ service type. A single offer collected by OC is a vector $[v_1, v_2, v_3, v_4, v_5]$, where v_1 corresponds to $m.amount$, v_2 to $t.amount$, v_3 to $t.profit$, and v_4 to $c.fee$. Since the attribute $m.amount$ is updated during the transformation, the offers should contain values from the world before and after the transformation. Thus v_5 stands for the value of $m.amount$ after modification. Assuming that instances of $Investment$ return k_1, k_2, and k_3 offers in response to subsequent inquiries, we obtain three offer sets: O^1, O^2, and O^3, where O^i is a $k_i \times 5$ matrix of offer values.

At the moment we develop two implementations of OC realizing the "simple", and the "intelligent" concept. The goal of the first approach is to rule out the offers violating simple constraints from the user query. An intelligent OC, taking advantage of an inference mechanism, a symbolic computation engine, and the semantic knowledge from the ontology, aims at discovering more sophisticated dependencies between offers and use them while collecting offers. Regardless of the approach chosen, every implementation of OC should satisfy some common requirements: *a*) the ability of a reconstruction of the intermediate worlds from a CAP, *b*) returning offer sets corresponding to the objects processed by the service types instances from a CAP, filled with the values acquired from real-world services, *c*) propagating the values and constraints present in the user query and returning them as expressions over offer sets, *d*) capturing the dependencies between the values of object attributes from the worlds of a CAP and returning them as expressions over offer sets, *e*) translating the set of quality expressions specified as a part of the query to a scalar function defined over offer sets, being the sum of all quality constraints.

In the third planning stage, the offers are searched by a *concrete planner* in order to find the best solution satisfying all constraints and maximising a quality function. Thus, we can formulate CPP as a constrained optimization problem.

Definition 8 (CPP). *Let n be the length of CAP and let $\mathbb{O} = (O^1, \ldots, O^n)$ be the vector of offer sets collected by OC such that for every $i = 1, \ldots, n$*

$O^i = \begin{bmatrix} o^i_{1,1} & \cdots & o^i_{1,m_i} \\ \vdots & \ddots & \vdots \\ o^i_{k_i,1} & \cdots & o^i_{k_i,m_i} \end{bmatrix}$*, and the j-th row of O^i is denoted by P^i_j. Let $\mathbb{P}$ denote the set of all possible sequences $(P^1_{j_1}, \ldots, P^n_{j_n})$, such that $j_i \in \{1, \ldots, k_i\}$ and $i \in \{1, \ldots, n\}$. The Concrete Planning Problem is defined as:*

$$max\{Q(S) \mid S \in \mathbb{P}\} \text{ subject to } \mathbb{C}(S), \tag{1}$$

where $Q : \mathbb{P} \mapsto \mathbb{R}$ is an objective function defined as the sum of all quality constraints and $\mathbb{C}(S) = \{C_j(S) \mid j = 1, \ldots, c \text{ for } c \in \mathbb{N}\}$, where $S \in \mathbb{P}$, is a set of constraints to be satisfied.

A solution of CPP consists in selecting one offer from each offer set such that all constraints are satisfied and the value of the objective function is maximized.

Theorem 1. *The concrete planning problem (CPP) is NP-hard.*

Proof. See Appendix.

3 Concrete Planning using SMT

This section deals with our novel application of SMT to CPP viewed as a constrained optimization problem. The idea is to encode CPP as an SMT formula such that there is a solution to CPP iff the formula is satisfiable. First, a set $\mathcal{V}$ of all necessary SMT-variables (for simplicity called just *variables*) is allocated:

- $\mathbf{q}$ - for storing the subsequent values of the quality function found,
- $\mathbf{oid}^i$, where $i = 1 \ldots n$ and n is the length of the abstract plan. These variables are needed to store the identifiers of offers constituting a solution. A single $\mathbf{oid}^i$ variable takes a value between 1 and k_i.
- $\mathbf{o}^i_j$, where $i = 1 \ldots n$, $j = 1 \ldots m_i$, and m_i is the number of offer values in the i-th offer set. We use them to encode the values of S, i.e., the values from the offers chosen as a solution. From each offer set O^i we extract the subset R^i of offer values which are present in the constraint set and in the quality function, and we allocate only the variables relevant for the plan.

Next, using the variables from $\mathcal{V}$, the offer values from the offer sets $\mathbb{O} = (O^1, \ldots, O^n)$ are encoded as the formula

$$ofr(\mathbb{O}, \mathcal{V}) = \bigwedge_{i=1}^{n} \bigvee_{d=1}^{k_i} \Big(\mathbf{oid}^i = d \ \wedge \bigwedge_{o^i_{d,j} \in R^i} \mathbf{o}^i_j = o^i_{d,j} \Big). \tag{2}$$

Then, the conjunction of all constraints is encoded as the formula $ctr\big(\mathbb{C}(S), \mathcal{V}\big)$, and the objective function as the formula $qual\big(Q(S), \mathcal{V}\big)$. For convenience its value is bound with the variable $\mathbf{q}$ by $\mathbf{q} = qual\big(Q(S), \mathcal{V}\big)$. Thus, the formula encoding the solutions of CPP is as follows:

$$cpp\big(\mathbb{O}, Q(S), \mathbb{C}(S), \mathcal{V}\big) = ofr\big(\mathbb{O}, \mathcal{V}\big) \wedge ctr\big(\mathbb{C}(S), \mathcal{V}\big) \wedge \mathbf{q} = qual\big(Q(S), \mathcal{V}\big) \tag{3}$$

The maximal value of $\mathbf{q}$ is searched using the $SMTsearch$ procedure presented in Alg. 1 adapting the *binary search* method. The *assumptions* mechanism of an SMT-solver is exploited, which consists in checking satisfiability of an SMT-formula assuming that a set of boolean conditions are satisfied. In every iteration the searched interval is divided in half and, since the objective function is to be

Procedure SMTsearch($cpp\big(\mathbb{O}, Q(S), \mathbb{C}(S), \mathcal{V}\big)$, δ, min, max)
Input: encoded formula, accuracy, estimated min. and max. value of $Q(S)$
Result: the maximal value of $\mathbf{q}$ with an accuracy of δ

begin
  $pivot \leftarrow (min + max)/2$; $a_1 \leftarrow (\mathbf{q} > pivot)$; $i \leftarrow 1$; $result \leftarrow null$;
  $A \leftarrow \{a_1\}$; `// a single assumption` a_1 `in the assumption set` A
  while $(|max - min| > \delta)$ **do**
    if $checkSat(cpp\big(\mathbb{O}, Q(S), \mathbb{C}(S), \mathcal{V}\big), A)$ **then**
      $i \leftarrow i + 1$; $result \leftarrow \mathbf{q}$; $min \leftarrow \mathbf{q}$; $pivot \leftarrow (min + max)/2$;
    else
      $A \leftarrow (A \setminus \{a_i\}) \cup \{\neg a_i\}$; `// replace` a_i `by` $\neg a_i$ `in` A
      $max \leftarrow pivot$; $pivot \leftarrow (min + max)/2$; $i \leftarrow i + 1$;
    end
    $a_i \leftarrow (\mathbf{q} > pivot)$; $A \leftarrow A \cup \{a_i\}$;
  end
  return $result$
end

Algorithm 1: Pseudocode of the SMT-based CPP algorithm

maximized, a solution of value greater than a half ($pivot$) is searched. To this end the whole formula is checked for satisfiability under the assumption ($\mathbf{q} > pivot$). If there is a solution, then its value becomes min. Otherwise, the searched value is less or equal $pivot$, the last assumption is replaced by its negation, and $pivot$ value is assigned to max. Then, a new $pivot$ value is computed and the algorithm iterates again, while the length of the searched interval is greater than δ.

4 Experimental Results

In this section we present the results of the experiments performed using the Z3 SMT-solver running on a standard PC equipped with 2GHz CPU and 8GB RAM. Since Genetic Algorithms are widely used in many optimization problems, we compare the efficiency of our new SMT-based approach with the results obtained using the standard GA, which we have implemented to this aim.

Implementation of GA. The only non-standard elements of our GA are the concrete plan encoding scheme and the computation of the fitness function. An individual is a sequence of indices of the offers chosen from the consecutive offer sets. The fitness value of an individual is the sum of the optimization objective and the ratio of the number of the satisfied constraints to the number of all constraints (see Def. 8), multiplied by some constant β:

$$fitness(Ind) = Q(S_{Ind}) + \beta \cdot \frac{|sat(\mathbb{C}\big(S_{Ind}\big))|}{c}, \tag{4}$$

where Ind stands for an individual, S_{Ind} is a sequence of the offer values corresponding to Ind, $sat\big(\mathbb{C}(S_{Ind})\big)$ is a set of the constraints satisfied by a candidate

solution represented by *Ind*, and c is the number of all constraints. The role of β is to reduce both of the sum components to the same order of magnitude and to control the impact of the components on the final result.

Experiments. In each of the experiments we use different optimization objectives and constraints, and compare the obtained results. Equation 5 presents the objectives Q_1, Q_2, Q_3 used in the experiments 1-5, while the constraints are combinations of $\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3$ defined by Equation 6. In the experiments 6 and 7, we use the constraints and the objective function of our working example.

$$Q_1 = \sum_{i=1}^{n} o^i_{j_i,1}, \qquad Q_2 = \sum_{i=1}^{n} o^i_{j_i,2}, \qquad Q_3 = \sum_{i=1}^{n} (o^i_{j_i,1} + o^i_{j_i,2}), \tag{5}$$

$$\mathbb{C}_1 = \{(o^i_{j_i,1} < o^{i+1}_{j_{i+1},1})\}, \quad \mathbb{C}_2 = \{(o^i_{j_i,2} < o^{i+1}_{j_{i+1},2})\}, \quad \mathbb{C}_3 = \{(o^i_{j_i,2} = o^{i+1}_{j_{i+1},2})\} \text{ for } i = 1, \ldots, n-1. \tag{6}$$

In all the experiments sets of offers generated randomly by our Offer Generator (OG) have been used. The values have been uniformly distributed in the range between 0 and 100, but the α parameter has been introduced, which specifies a percentage of values below 33.3, in order to get different distributions of high quality concrete plans. The following values of the GA parameters have been used: the number of the individuals equals to 1000, the probability of mutation equals to 0.5%, the probability of the one-point crossover operator equals to 95%, and the algorithm was run 100 times for each setup. As to the SMT-based algorithm, the 500 sec. time-out has been set. Each experiment has been repeated only 10 times as the run-times obtained have been very similar, and the quality values have been the same each time.

In Experiment 1 Q_1 has been used as an optimization objective and $\mathbb{C}_1$ as a set of constraints. In all experiments we have tested instances with 5, 10, and 15 offer sets, containing 256, 512, and 1024 offers each. Two sets of offers with $\alpha = 5\%$ and $\alpha = 40\%$ have been generated. The results are presented in Table 1. The SMT-based planner always returns optimum, while GA, as a non-deterministic algorithm, finds optimum in at most 20% cases, for instances with 5 service types. All solutions are obtained within 0.25 to nearly 3 seconds. Comparing quality we can observe that the difference between the SMT-solver and GA ranges from 0.5% up to about 31% for instances with 15 service types.

In Experiment 2 the constraints remain the same ($\mathbb{C}_1$), the objective function is similar (Q_2), but we maximize the sum of other values than these used in the constraints. Table 2 presents the impact of these changes on the SMT-based planner. In comparison to the results of Exp. 1, the runtime of SMT-solver increases.The biggest difference can be noticed for plans of length 15, however, as it follows from the probability results, these are also hard to find for GA. On the other hand, the power of SMT is in the ability to take advantage of the constraints in order to reduce the search space. The results suggest that working with constraints related to different variables than these used in the objective function leads to longer runtimes of the SMT-solver.

Table 1. The results of Experiment 1: left $\alpha = 40\%$, right $\alpha = 5\%$.

			SMT		GA				SMT		GA			
Sp.	**n**	**Offs**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Bs.**	**Pr.**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Bs.**	**Pr.**
2^{45}	5	512	0.28	485	0.41	479.23	18	100	0.36	490	0.42	484.95	13	100
2^{50}		1024	0.52	490	0.86	481.95	4	100	0.52	490	0.76	484.9	9	100
2^{80}	10	256	0.51	920	0.47	720.21	0	92	0.47	955	0.47	794.64	0	91
2^{90}		512	0.68	955	0.83	797.97		91	0.72	955	0.83	824.96		97
2^{100}		1024	1.27	955	1.59	802.44		100	1.33	955	1.59	853.56		98
2^{120}	15	256	0.73	1350	0.9	929.3		10	0.95	1388	0.76	1102.07		14
2^{135}		512	1.49	1377	1.47	998.8		15	1.28	1395	1.35	1092.33		12
2^{150}		1024	2.06	1395	2.78	1027.43		16	2.09	1395	2.53	1086.85		14

Table 2. The results of Experiment 2: left $\alpha = 40\%$, right $\alpha = 5\%$.

			SMT		GA				SMT		GA			
Sp.	**n**	**Offs**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Bs.**	**Pr.**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Bs.**	**Pr.**
2^{45}	5	512	0.87	499	0.37	493.62	4	100	0.50	499	0.39	498	17	100
2^{50}		1024	1.55	499	0.72	495.91	9	100	0.65	499	0.67	498.72	28	100
2^{80}	10	256	4.47	979	0.46	934.69	2	89	2.19	994	0.45	957.78	1	88
2^{90}		512	3.44	992	0.83	923.62	1	89	2.76	996	0.83	967.37		90
2^{100}		1024	6.02	995	1.53	949.09		93	3.34	998	1.57	959.55		98
2^{120}	15	256	249.46	1443	0.74	1269.87	0	8	100.26	1475	0.73	1416	0	9
2^{135}		512	425.32	1467	1.5	1322.92		12	97.20	1489	1.35	1362.28		7
2^{150}		1024	57.74	1493	2.67	1300.58		12	63.94	1495	2.49	1398.38		18

Table 3. The results of Experiment 3 (left) and Experiment 4 (right).

			SMT		GA			SMT		GA		
Sp.	**n**	**Offs**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Pr.**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Pr.**
2^{45}	5	512	7.73	924	0.36	865.8	100	1.59	841	0.32	754.51	100
2^{50}		1024	3.95	947	0.61	892.13	100	1.41	904	0.62	781.3	100
2^{80}	10	256	> 500	1633*	0.4	1464.64	93	150.58	1207	0.4	1025.5	2
2^{90}		512	215.13	1803	0.7	1479.98	86	24.09	1562	0.69	1187.25	4
2^{100}		1024	243.24	1862	1.29	1535.52	96	345.57	1655	1.34	1211.72	11
2^{120}	15	256	> 500	2448*	0.64	2067.45	11	> 500	1550*	—		0
2^{135}		512		2291*	1.15	2148.08	12		1567*			
2^{150}		1024		2630*	2.11	2134.35	17		2270*			

Table 4. The results of Exp. 5 (left), Exp. 6 (center), and Exp. 7 (right).

			SMT		GA			SMT		GA			SMT	
Sp.	**n**	**Offs**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Pr.**	**t[s]**	**Q**	**t[s]**	**AvgQ**	**Pr.**	**t[s]**	**Q**
2^{45}	5	512	0.54	752	0.34	356.6	14	3.63	393	0.32	266.26	30	0.41	5
2^{50}		1024	0.85	866	0.64	420.2	10	3.48	469	0.6	274.86	30	0.68	10
2^{80}	10	256	0.73	noSol	—		0	35.24	683	0.4	289	8	0.32	noSol
2^{90}		512	5.01	780				159.09	753	0.68	399.22	18	1.18	12
2^{100}		1024	3.51	1508				143.07	888	1.3	378.5	24	2.78	20
2^{120}	15	256	1.10	noSol				> 500	942*	0.64	446	1	1.00	noSol
2^{135}		512	10.43	1164					767*	1.14	536	2	5.54	18
2^{150}		1024	40.82	1287					755*	2.08	481	5	16.97	29

In order to discover the limitations of both the planners, in Experiments 3, 4, and 5 we use the harder set of data, i.e., the offers generated with $\alpha = 40\%$. We examine how the approaches deal with the "sum of sums" in the optimization function, and thus in Exp. 3 we use Q_3 as the optimization objective and $\mathbb{C}_1$ as the set of constraints. Moreover, we aim at confirming our conjecture on behaviour of the SMT-solver in presence of a larger number of constraints. Thus, in Exp. 4 we use Q_3 as the optimization objective and $\mathbb{C}_1 \cup \mathbb{C}_2$ as the set of constraints. Table 3 presents the results.

Comparing the results of Exp. 2 and 3 one can notice that the more complicated objective function has almost no influence on the runtime and the probability of finding solutions by GA. On the other hand, the instances from Exp. 3 seem to be a bit harder for GA because the quality difference between SMT and GA is greater and ranges from 3.4% up to about 19%, and furthermore, GA could not find an optimal solution. The results of our SMT-based approach indicate that the constraints used in Exp. 3 are too weak to significantly bound the search space. We were unable to find the optimal solution in four cases.

In Exp. 4 we use two times more constraints than in Exp. 3. Firstly, we found the limit beyond which an application of GA is pointless. In Exp. 4 we use $2 \cdot (n-1)$ constraints. For plans of length 10 and for 18 constraints GA barely finds a few solutions quality of which differ from optimum by 12% to almost 27%. Secondly, adding more constraints improves slightly the efficiency of an SMT-solver. However, not only the number of constraints is important, but also their influence on the number of existing solutions in the search space. We prove it by choosing $\mathbb{C}_1 \cup \mathbb{C}_3$ as the set of constraints (i.e., we change the half of constraints from "less than" to "equal") and running Exp. 5. The results are in Table 4. The SMT-runtime decreases tremendously, as well as quality and the probability of finding a solution by GA. Now, GA barely finds solutions of length 5 quality of which differ from optimum by 13.7% up to almost 53%.

Experiments 6 and 7 are based on our working example and have been performed on datasets with $\alpha = 90\%$. In Exp. 6 we used 6, 11, and 14 constraints for plans of length 5, 10, and 15, respectively. The quality of solutions found by GA is worse by about 30% to almost 58% than the ones found by the SMT-solver. Moreover, the solutions have been found with a low probability. Unfortunately, the runtimes of SMT-based planner are quite long in this case. However, using a set of 11, 21, and 29 constraints for plans of length 5, 10, and 15, respectively, which significantly reduces the number of solutions existing in the search space, we obtain a very nice behaviour of our SMT-based planner in Exp. 7. Table 4 presents the results, where *noSol* means that there is no solution at all.

5 Conclusions

In the paper we have presented the concrete planning in the Planics framework, its reduction to the constrained optimization problem, and a new SMT-based approach to solve it. The experimental results, compared with results of the standard GA, present advantages and disadvantages of both the approaches. The

most important feature of the SMT-based planner is its ability of finding always the optimal solution, provided that it is enough time and memory. In contrast, GA finds sometimes the optimal solution of length at most 5, but it consumes less time and memory. The ability of GA to find a concrete plan depends on the number of constraints. The more optimization constraints the smaller probability of finding a concrete plan. These drawbacks of GA are not common to our SMT-based approach. Moreover, our experiments show that a large number of constraints helps the SMT-solver to reduce the search space and to find the optimal solution faster. Our experimental results show that an application of the SMT-based method to solve CPP is promising and valuable against the well known GA-based approach. Overall, both the approaches are complementary and behave differently depending upon a particular problem instance.

References

1. S. Ambroszkiewicz. Entish: A language for describing data processing in open distributed systems. *Fundam. Inform.*, 60(1-4):41–66, 2004.
2. M. Bell. *Introduction to Service-Oriented Modeling.* John Wiley & Sons, 2008.
3. M. M. Bersani, L. Cavallaro, A. Frigeri, M. Pradella, and M. Rossi. SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In *SEFM*, pages 244–254, 2010.
4. G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 1069–1075, 2005.
5. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, and J. Skaruz. HarmonICS - a tool for composing medical services. In *ZEUS*, pages 25–33, 2012.
6. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, and A. Zbrzezny. PlanICS - a web service compositon toolset. *Fundam. Inform.*, 112(1):47–71, 2011.
7. M. Elwakil, Z. Yang, L. Wang, and Q. Chen. Message race detection for web services by an SMT-based analysis. In *Proc. of the 7th Int. Conference on Autonomic and Trusted Computing*, ATC'10, pages 182–194. Springer, 2010.
8. G. Monakova, O. Kopp, F. Leymann, S. Moser, and K. Schäfers. Verifying business rules using an SMT solver for BPEL processes. In *BPSC*, pages 81–94, 2009.
9. A. Niewiadomski and W. Penczek. Towards SMT-based Abstract Planning in PlanICS Ontology. In *Proc. of KEOD 2013 – International Conference on Knowledge Engineering and Ontology Development*, 2013 (to appear).
10. A. Niewiadomski, W. Penczek, and A. Półrola. Abstract Planning in PlanICS Ontology. An SMT-based Approach. Technical Report 1027, ICS PAS, 2012.
11. J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of SWSWPC'04*, volume 3387 of *LNCS*, pages 43–54. Springer, 2004.
12. Y. Wu and X. Wang. Applying multi-objective genetic algorithms to qos-aware web service global selection. *Advances in Information Sciences and Service Sciences*, 3(11):134–144, 2011.

Appendix

Proof (NP-hardness of CPP). We show that concrete planning problem is NP-hard by showing the linear reduction of 3-SAT problem to CPP. Consider a set of propositional variables $\mathcal{PV}$ and 3-CNF formula $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_n$, where $c_i = (x_i^1 \vee x_i^2 \vee x_i^3)$, $x_i^j = p$ or $x_i^j = \neg p$, for every $p \in \mathcal{PV}$, $i = 1 \ldots n$ and $j = 1 \ldots 3$. We encode the satisfiability problem of φ as a Concrete Planning Problem CPP as follows.

We take a Context Abstract Plan (CAP) of length n, and n offer sets. Each offer contains 3 values from the set $\{0, 1\}$, and each value corresponds to a single propositional variable used in i-th clause. Each offer set contains all the possible combinations of offer values (8 offers per set), that is, each offer set is an 8×3 matrix. Thus, $\mathbb{P}$ is the set of all possible binary sequences of length $3n$.

We transform the formula φ to a set of constraints $\mathbb{C}$ in such a way that every clause c_i became a single constraint, where $x_i^j = p$ is encoded as $o^i_{k_i,j} = 1$ and $x_i^j = \neg p$ as $o^i_{k_i,j} = 0$, for $k_i = 1 \ldots 8$ and $p \in \mathcal{PV}$. Moreover, for every propositional variable p occurring in φ we take two subsets of offer variables P_p and N_p, which encode p and $\neg p$, respectively: $P_p = \{o^i_{k_i,j} \mid \text{for every } i, j \text{ such that } x_i^j = p\}$ and $N_p = \{o^i_{k_i,j} \mid \text{for every } i, j \text{ such that } x_i^j = \neg p\}$. Now, for each non-empty set X_p, where $X \in \{P, N\}$ and $p \in \mathcal{PV}$, we order the elements of X_p according to increasing values of their indices and we build the sequence $\overline{X_p} = (a_1, a_2, \ldots, a_{|X_p|})$, where $a_i \in X_p$. Next, we add the following constraints to our constraint set: $\{(a_i = a_{i+1}) \mid \text{for } a_i \in X_p \text{ and } i = 1, \ldots, (|X_p| - 1)\}$, that is, we require the neighbouring elements of the sequence to be equal. Moreover, for every pair of non-empty sequences $(\overline{P_p}, \overline{N_p})$, where $\overline{P_p} = (a_1, \ldots, a_{|P_p|})$ and $\overline{N_p} = (b_1, \ldots, b_{|N_p|})$, we add a single constraint: $(a_1 \neq b_1)$.

Finally, we take the constant objective function (e.g. $Q(S) = 1$, for $S \in \mathbb{P}$). Then, CPP has a solution iff φ is satisfiable.

Granular Mereotopology: A First Sketch

Lech Polkowski[1,2] and Maria Semeniuk–Polkowska[3]

[1] Polish-Japanese Institute of Information Technology
Koszykowa str. 86, 02-008 Warszawa, Poland
[2] Department of Mathematics and Computer Science
University of Warmia and Mazury, Sloneczna 54, 10-504 Olsztyn, Poland
[3] Chair of Formal Linguistics
University of Warsaw, Dobra str. 55, 00-312 Warszawa, Poland
polkow@pjwstk.edu.pl, m.polkowska@uw.edu.pl

Abstract. Mereotopology aims at a reconstruction of notions of set topology in mereological universa. Because of foundational differences between set theory and mereology, most notably, the absence of points in the latter, the rendering of notions of topology in mereology faces serious difficulties. On the other hand, some of those notions, e.g., the notion of a boundary, belong in the canon of the most important notions of mereotopology, because of applications in, e.g., geographic information systems. Rough mereology allows for a formal theory of knowledge granulation, and, granules may serve as approximations to open sets, hence, it is reasonable to explore the possibility of their usage in buildup of mereotopological constructs. This work is segmented into sections on mereology, rough mereology, granule theory, mereotopology.
Keywords: spatial reasoning, mereotopology, rough mereology, boundary, open set.

1 Standard Mereology

Under the term *Standard Mereology* we understand the theory of parts constructed by Stanislas Leśniewski, cf. [8], [10], [13]. Given some collection (a universe), say U, of things, a relation of part on them is a binary relation *part* which is required to be

M1 *Irreflexive*: *For each* $x \in U$ *it is not true that* $part(x, x)$

M2 *Transitive*: *For each triple* x, y, z *of things in* U, *if* $part(x, y)$ *and* $part(y, z)$, *then* $part(x, z)$

Fig. 1 shows the chessboard with parts being white and black squares.

The relation of *part* gives rise to the relation of an *ingredient*, *ingr*, defined as

$$ingr(x, y) \Leftrightarrow \pi(x, y) \vee x = y. \tag{1}$$

Clearly, the relation of an ingredient is a partial order on things.

We formulate the third axiom of Standard Mereology which does involve the notion of an ingredient. Before it, we introduce a property of things. For things x, y, we let,

Fig. 1. White and black squares as parts of the chessboard

$I(x, y)$: *For each thing* t *such that* $ingr(t, x)$, *there exist things* w, z *such that* $ingr(w, t), ingr(w, z), ingr(z, y)$

Now, we can state an axiom.

M3 (*Inference Rule*) *For each pair of things* x, y, the property $I(x, y)$ implies that $ingr(x, y)$

The predicate of *overlap*, Ov in symbols, is defined by means of

$$Ov(x, y) \Leftrightarrow \exists z.ingr(z, x) \wedge ingr(z, y). \tag{2}$$

1.1 The class operator

Aggregation of things into a composite thing is done in set theory by means of the union of sets operator. Its counterpart, and a generalization, in mereology, is the class operator. For a non–empty property Φ of things, the *class of* Φ, denoted $Cls\Phi$, is defined by the conditions

C1 *If* $\Phi(x)$, *then* $ingr(x, Cls\Phi)$

C2 *If* $ingr(x, Cls\Phi)$, *then there exists* z *such that* $\Phi(z)$ *and* $I(x, z)$

In plain language, the class of Φ collects in an individual object all objects satisfying the property Φ.
The existence of classes is guaranteed by an axiom.

M4 *For each non–vacuous property* Φ *there exists a class* $Cls\Phi$
The uniqueness of the class follows by M3.

In Fig. 1, we can discuss the class of white squares, the class of black squares, or, the class of occupied squares.

Example 1. 1. The strict inclusion $\subset$ on sets is a part relation. The corresponding ingredient relation is the inclusion $\subseteq$. The overlap relation is the non–empty intersection. For a non–vacuous family F of sets, the class $ClsF$ is the union $\bigcup F$;

2. For reals in the interval $[0,1]$, the strict order $<$ is a part relation and the corresponding ingredient relation is the weak order $\leq$. Any two reals overlap; for a set $F \subseteq [0,1]$, the class of F is $supF$.

The notion opposite to that of overlap is the notion of *disjointness*: its symbol is $extr$, and, for things x, y,

$$extr(x,y) \Leftrightarrow \text{it is not true that } Ov(x,y). \tag{3}$$

The notion of a *complement* to an object, relative to another object, is rendered as a ternary predicate $comp(x,y,z)$, [8], par. 14, Def. IX, to be read:'x is the complement to y relative to z', and it is defined by means of the following requirements,

1. $x = ClsEXTR(y,z)$;
2. $ingr(y,z)$,

where $EXTR(y,z)(t)$ holds if and only if $ingr(t,z)$ and $extr(t,y)$.

This definition implies that the notion of a complement is valid only when there exists an ingredient of z exterior to y.

The notion of a class has been extensively studied motivated by its fundamental importance for foundations of mathematics, logics and mereology, cf., e.g., Lewis [9].

For the property $Ind(x) \Leftrightarrow ingr(x,x)$, one calls the class $ClsInd$, *the universe*, in symbols V,cf.,[8], par. 12, Def. VII. The complement with respect to the universe of a thing serves as the complement in algebraic sense.

We let for an object x,

$$-x = ClsEXTR(x,V). \tag{4}$$

It follows that

1. $-(-x) = x$ *for each object* x;
2. $-V$ *does not exist.*

In Fig. 1, the complement to the class of white squares is the class of black squares (we assume that classes of squares are ingredients of the chessboard as well). The operator $-x$ can be a candidate for the Boolean complement in a structure of a Boolean algebra within Mereology, constructed in [18], and anticipated in [17]; in this respect, cf., [5]. This algebra will be obviously rid of the null element, as the empty object is not allowed in Mereology , and the meet of two objects will be possible only when these objects overlap. Under this caveat, the construction of Boolean operators of join and meet proceeds as in [18].

We define the Boolean sum $x + y$ by letting

$$x + y = Cls(t : ingr(t,x) \vee ingr(t,y)). \tag{5}$$

In Fig. 2, we give an example of the sum which is the full moon as the sum of the two quarters: 4th and 1st ('halves').

Fig. 2. The 4th quarter of the moon =x; the 1st quarter of the moon =y; the full moon= x+y

The product $x \cdot y$, cf., [18] is defined in a parallel way,

$$\text{If } Ov(x,y) \text{ then } x \cdot y = Cls(t : ingr(t,x) \wedge ingr(t,y)). \tag{6}$$

Operators $+, \cdot, -$ and the unit V introduce the structure of a complete Boolean algebra without the null element, cf., [18], [13].

An often invoked example of a mereological universe is the collection ROM_n of regular open sets in the Euclidean space E^n; we recall that an open set A is *regular open* when

$$A = IntClA, \tag{7}$$

where Int, Cl are , respectively, the interior and the closure operators of topology, see, e.g., [10], Ch.2. In this universe, mereological notions are rendered as

1. $ingr(A,B) \Leftrightarrow A \subseteq B$;
2. $part(A,B) \Leftrightarrow A \subset B$;
3. $Ov(A,B) \Leftrightarrow A \cap B \neq \emptyset$;
4. $A \cdot B = A \cap B$;
5. $-A = R^n \setminus ClA$;
6. $A + B = A \cup B$.

2 Rough Mereology

Rough Mereology, cf., , [10], [11], [12], introduces the notion of a part to a degree, $\mu(x,y,r)$ read '*x is a part in y to a degree of r*' with requirements

RM1 $\mu(x,y,1) \Leftrightarrow ingr(x,y)$

RM2 $\mu(x,y,1) \wedge \mu(z,x,r) \Rightarrow \mu(z,y,r)$

RM3 $\mu(x,y,r) \wedge s \leq r \Rightarrow \mu(x,y,s)$

where *ingr* is the ingredient relation in an a priori assumed Mereology.

The relation μ called *a rough inclusion* in [11] can be induced in some ways from t–norms, for t–norms, see, e.g., [6], [10], Ch. 4.

2.1 Rough inclusions from residua of continuous t–norms

In the first case, for a continuous t–norm t, cf., e.g., [6], [10], Ch. 4, Ch. 6.2., the *residual implication* $x \Rightarrow_t y$ defined as

$$x \Rightarrow_t y = max\{r : t(x,r) \leq y\}, \tag{8}$$

yields the rough inclusion

$$\mu_t(x,y,r) \Leftrightarrow x \Rightarrow_t y \geq r. \tag{9}$$

2.2 Rough inclusions from archimedean t–norms

In the other case, for the *t–norm of Łukasiewicz*,

$$t_L(x,y) = max\{0, x+y-1\}, \tag{10}$$

or, the *product* t–norm,

$$t_P(x,y) = xy, \tag{11}$$

see, e.g., [10], Ch. 4, which admit representations,

$$t_L(x,y) = g_L(f_L(x) + f_L(y)), t_P(x,y) = g_P(f_P(x) + f_P(y)) \tag{12}$$

with

$$g_L(x) = 1 - x = f_L(x), g_P(x) = exp(-x), f(x) = -lnx, \tag{13}$$

cf., [6], [10], Ch. 4, one defines the rough inclusion

$$\mu^L(x,y,r) \Leftrightarrow g_L(|x-y|) \geq r, \tag{14}$$

respectively,

$$\mu^P(x,y,r) \Leftrightarrow g_P(|x-y|) \geq r. \tag{15}$$

The last formula can be transferred to the realm of finite sets, with g either g_L or g_P, as

$$\mu^L_s(X,Y) = g(\frac{|X \triangle Y|}{|X|}) = \frac{|X \cap Y|}{|X|}, \tag{16}$$

to the case of bounded measurable sets in E^n as

$$\mu^L_G(X,Y) = g(\frac{||X \triangle Y||}{||X||}) = \frac{||X \cap Y||}{||X||}, \tag{17}$$

where $a \triangle b$ denotes the symmetric difference of a, b, $|a|$ is the cardinality of a, and, $||a||$ is the measure (area) of a.

2.3 Transitivity of rough inclusions

An important property of rough inclusions is the *transitivity property*. For rough inclusions of the form μ^t with t being L or P, as well as for rough inclusions of the form μ_t this property has the form, see Polkowski [10], Props. 6.7, 6.16,

$$\mu^t(x,y,r) \wedge \mu^t(y,z,s) \Rightarrow \mu^t(x,z,t(r,s)). \tag{18}$$

In case of rough inclusions of the form μ_t, it becomes,

$$\mu_t(x,y,r) \wedge \mu_t(y,z,s) \Rightarrow \mu_t(x,z,t(r,s)). \tag{19}$$

3 Granules as weakly open sets in rough mereology

We begin our study of mereotopology in a rough mereological universe U with a given rough inclusion μ. In order to introduce topological structures, we first introduce a mechanism of granulation in U. For a thing x in U and a real number r in the interval $[0,1]$, we define the *granule* $g(x,r,\mu)$, about x of radius r, as

$$g(x,r,\mu) \text{ is } ClsM(x,r,\mu), \tag{20}$$

where

$$M(x,r,\mu)(y) \Leftrightarrow \mu(y,x,r). \tag{21}$$

Granules can be characterized in terms of rough inclusions as follows, see Polkowski [10], Ch. 7, Props. 7.1, 7.2.

Proposition 1. *For granules induced by rough inclusions of the form μ^t as well as for granules induced by the rough inclusion μ_M, we have for each pair x, y of things, $ingr(y, g(x,r,\mu))$ if and only if $\mu(y,x,r)$.*

For granules induced by rough inclusions μ_L, μ_P, the situation is more complicated, see Polkowski [10], 7.3.

3.1 Open sets

We apply the granules to define neighborhoods of things in U. To this end, we define a property $N(x,r,\mu)$ by letting,

$$N(x,r,\mu)(y) \Leftrightarrow \exists s > r.\mu(y,x,s). \tag{22}$$

The *neighborhood* $n(x,r,\mu)$ of a thing x of radius r relative to μ is defined as

$$n(x,r,\mu) \text{ is } ClsN(x,r,\mu). \tag{23}$$

The neighborhood system has properties of open sets, viz., see [10], Ch. 7,

1. If $ingr(y, n(x,r,\mu))$, then $\exists s.ingr(n(y,s,\mu), n(x,r,\mu))$;
2. If $s > r$, then $ingr(n(x,s,\mu), n(x,r,\mu))$;

3. If $ingr(y, n(x, r, \mu))$ and $ingr(y, n(z, s, \mu))$, then

$$\exists q.ingr(n(z, sq, \mu), n(x, r, \mu)) \text{ and } ingr(n(z, q, \mu), n(y, s, \mu)).$$

We define an open set as a collection of neighborhoods; the predicate $open(F)$ is therefore defined as,

$$open(F) \Leftrightarrow \forall z.[z \in F \Leftrightarrow z \ is \ n(x, r, \mu) \ for \ some \ x, r \ ]. \tag{24}$$

It is now possible to define open things as classes of open collections,

$$open(x) \Leftrightarrow \exists F.open(F) \wedge x \ is \ ClsF. \tag{25}$$

Closed things are defined as complements to open things,

$$closed(x) \Leftrightarrow open(-x). \tag{26}$$

We may need as well the notion of a closed collection, as the complement to an open collection,

$$closed(F) \Leftrightarrow open(-F), \tag{27}$$

where, clearly, the complement $-F$ is the collection obtained by applying the mereological complement $-$ to each member of F.

4 Boundaries

The practical importance of boundaries stems from their role as separating regions among areas of interest like roads, rivers, fields, forests etc., and this causes the theoretical interest in them. The notion of a boundary has been studied in philosophy, mathematics, computer science by means of mereology. Mathematics resolved the problem of boundaries by topological notion of the boundary (frontier) BdX of a set X in a topological space (U, τ) which was defined as

$$BdX = ClX \setminus IntX, \tag{28}$$

i.e., any point $x \in U$ satisfies

$$x \in BdX \Leftrightarrow \forall P.P \ open \wedge x \in P \Rightarrow P \cap X \neq \emptyset \neq P \cap (U \setminus X).$$

It is evident from this definition that the notion of the boundary of X involves in the symmetrical way the complement:

$$BdX = Bd(U \setminus X). \tag{29}$$

It also follows that the notion of a boundary is of infinitesimal character as detecting whether $x \in BdX$ involves neighborhoods of x of arbitrarily small size.

Philosophers noticed this duality of boundaries between things and their complements and went even further, in the extreme cases, assigning the boundary character to any thing by considering it as a potential boundary (the phenomenon of *plerosis*, e.g., a point in the open disc can be the end point of any radius from it to the perimeter of the disc, a fortiori, in the boundary of continuum many segments. Moreover, e.g., the perimeter of the planar disc, considered, e.g., in 3D space, can be the boundary of continuum many bubbles spanned on the perimeter , see [2], [3], [15]).

4.1 Mereoboundaries

Topological definition of boundary led Smith [14] toward a scheme for defining mereoboundaries. First, He proposes an axiomatic introduction of open sets as *interior parts*, IP in symbols. In this context, the notion of *straddling*, Str in symbols, is defined as,

$$Str(x, y) \Leftrightarrow [\forall z.IP(x, z) \Rightarrow Ov(z, y) \wedge Ov(z, -y)]. \tag{30}$$

The notion of a *boundary part* is introduced in Smith [14] by means of an auxiliary predicate

$$B(x, y) \Leftrightarrow \forall z.ingr(z, x) \Rightarrow Str(z, y). \tag{31}$$

Boundary $Bd(y)$ of a thing y is defined as

$$Bdy(y) = Cls\{x : B(x, y))\}. \tag{32}$$

It is a straightforward task to verify that in the space ROM_n of regular open sets, each set x is an interior set of each of its supersets and requirements for IP are fulfilled, $Str(x, y)$ is satisfied in case $Ov(x, y) \wedge Ov(x, -y)$ and $B(x, y)$ is satisfied for no x, y hence the boundary is not defined being empty. The reason is a too liberal definition of straddling, allowing mere ingredients of a given thing x.

4.2 Granular mereoboundaries

For this reason, we re–model the approach by Smith in [14] by allowing granular neighborhoods as open things, a fortiori interior parts, and by restricting interior parts to granular neighborhoods about the same thing. In detail, our approach presents itself as follows.

We say that a granular neighborhood $n(x, r, \mu)$ *granular straddles* a thing y if and only if the following property $GStr(x, r, y)$ holds,

$$GStr(x, r, y) \Leftrightarrow \forall s \in (r, 1).Ov(n(x, s, \mu), y) \wedge Ov(n(x, s, \mu), -y). \tag{33}$$

Let us observe that the notion of granular straddling is *downward hereditary* in the sense that

$$GStr(x, r, y) \wedge s > r \Rightarrow GStr(x, s, y). \tag{34}$$

Also, it is manifest that this notion is *upward hereditary*, i.e.,

$$GStr(x,r,y) \Rightarrow \forall s < r.GStr(x,s,y). \tag{35}$$

With each granule $g(x,r,\mu)$, we associate the collection $GN(x,\mu) = \{n(x,s,\mu) : s \in (0,1)\}$, which we call the $x - ultrafilter\ base$. We say that an x-ultrafilter base $GN(x,\mu)$ *granular straddles* a thing y if and only if there exists an $s \in (0,1)$ such that $GStr(x,s,\mu),y)$ holds. We denote this fact with the symbol $B(x,y)$.

We regard the collection $GN(x,\mu)$ as a *point at infinity* and, according to the topological nature of boundary, we assign to the thing x such that the x-ultrafilter base $GN(x,\mu)$ granular straddles a thing y this point at infinity as the boundary point of y. Hence, we define the boundary of y, in symbols Bdy, as the collection of those points,

$$Bdy \text{ is } \{x :\ B(x,y)\}. \tag{36}$$

Boundaries defined in this way are $ingr - upward - hereditary$ in the sense,

$$ingr(z,x) \wedge B(z,y) \Rightarrow B(x,y). \tag{37}$$

The proof follows from definitions by M3 and transitivity of the applied rough inclusion. In view of the correspondence between things and ultrafilter bases, we may say that the thing x is a boundary point of the thing y in case the x–ultrafilter base granular straddles y. This approach does satisfy philosophical postulates about boundary like

1. The boundary of a thing may not belong in the universe of considered things; in other words, the boundary is of different topological type then the thing;
2. in order to preserve the typology of the boundary one has to preserve its infinitesimal character;
3. the boundary of a thing may be a boundary of a plethora of other things , in particular, by necessity, it has to be the boundary of each complement to the thing.

Let us observe that the set–theoretic complement to Bdy is open as it is the collection,

$$\{z : \exists s.ingr(n(z,s,\mu),y) \vee ingr(n(z,s,\mu),-y)\}, \tag{38}$$

hence, Bdy is a closed collection for each thing y.

It is a straightforward task to check that in the space ROM_n, for a regular open set A, the granular boundary is defined by

$$B(Z,A) \Leftrightarrow ClZ \cap A \neq \emptyset. \tag{39}$$

5 Conclusion

We admit an infinitesimal nature of boundaries along with the fact that their nature is distinct from the nature of things they bound, like it happens to closed

nowhere dense boundaries of regular open sets, and we represent them by means of ultrafilters constructed in the meta–space of collections of things. We have aimed at giving a definition of boundary in purely mereological terms, without any resort to augmentations which are necessary for a more exact description, like geographic directions, notions of touching, contact, beacons, in a word many other than mereological primitive notions, see, e.g., [1], [7].

References

1. Aurnague, M., Vieu, L., Borillo, A.: Representation formelle des concepts spatiaux dans la langue. In: Denis, M. (ed.): Langage et Cognition Spatiale, Masson, Paris, pp 69–102 (1997).
2. Brentano, F.: The Theory of Categories. Nijhoff, The Hague (1981).
3. Chisholm, R.: Spatial continuity and the theory of part and whole. A Brentano study. Brentano Studien 4, pp 11–23 (1992–3).
4. Clarke, B. L.: A calculus of individuals based on connection. Notre Dame Journal of Formal Logic 22(2), pp 204–218 (1981).
5. Clay, R.: Relation of Leśniewski's Mereology to Boolean Algebra. The Journal of Symbolic Logic 39, pp 638–648 (1974).
6. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer, Dordrecht (1998).
7. Hayes, P. J.: The second naive physics manifesto. In: Hobbs, J. R., Moore, R. C. (eds.): Formal Theories of the Common–Sense World, Ablex, Norwood, pp 1–36 (1985).
8. Leśniewski, S.: Podstawy Ogólnej Teoryi Mnogości, I (Foundations of General Set Theory, I, in Polish). Prace Polskiego Koła Naukowego w Moskwie, Sekcya Matematyczno–przyrodnicza (Trans. Polish Scientific Circle in Moscow, Section of Mathematics and Natural Sciences) No. 2, Moscow (1916); see also Leśniewski, S.: On the foundations of mathematics. Topoi 2, pp 7–52 (1982) (translation by E. Luschei of the former).
9. Lewis, D.: Parts of Classes. Blackwell, Oxford UK (1991).
10. Polkowski, L.: Approximate Reasoning by Parts. An Introduction to Rough Mereology. Springer Verlag, Berlin (2011).
11. Polkowski, L., Skowron, A.: Rough mereology. In: Proceedings of ISMIS'94. Lecture Notes in Artificial Intelligence 869, Springer Verlag, Berlin, pp 85–94 (1994).
12. Polkowski, L., Skowron, A.: Rough mereology: a new paradigm for approximate reasoning. International Journal of Approximate Reasoning 15 (4), pp 333–365 (1997).
13. Simons, P.: Parts. A Study in Ontology. Clarendon Press, Oxford UK (1987, repr. 2003).
14. Smith, B.: Mereotopology: A theory of parts and boundaries.Data and Knowledge Engineering 20, pp 287–303 (1996).
15. Smith, B.: Boundaries: An essay in mereotopology. In: Hahn, L. (ed.): The Philosophy of Roderick Chisholm. La Salle: Open Court, pp 534–561 (1997).
16. Sobociński, B.: Studies in Leśniewski's Mereology. Yearbook for 1954-55 of the Polish Society of Art and Sciences Abroad V, pp 34–43 (1954–5).
17. Tarski, A.: Les fondements de la géométrie des corps. Supplement to Annales de la Société Polonaise de Mathématique 7, pp 29–33 (1929).
18. Tarski, A.: Zur Grundlegung der Booleschen Algebra. I. Fundamenta Mathematicae 24, pp 177–198 (1935).

SMT-Based Reachability Checking for Bounded Time Petri Nets

Extended Abstract

A. Półrola, P. Cybula, and A. Męski

[1] University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland
{polrola,cybula}@math.uni.lodz.pl
[2] Institute of Computer Science, PAS, Jana Kazimierza 5, 01-248 Warsaw, Poland
meski@ipipan.waw.pl

Abstract. Time Petri nets by Merlin and Farber are a powerful modelling formalism. However, symbolic model checking methods for them consider in most cases the nets which are 1-safe, i.e., allow the places to contain at most one token. In our paper we present a preliminary version of the approach aimed at testing reachability for time Petri nets without this restriction. We deal with the class of bounded nets restricted to disallow multiple enabledness of transitions, and present the method of reachability testing based on a translation into a satisfiability modulo theory (SMT).

1 Introduction

The process of design and production of both systems and software – among them, the concurrent ones – involves testing whether the product conforms to its specification. This can be achieved using various kinds of formal methods. However, in order to apply any of these methods, the system to be tested needs to be modelled using an appropriate formalism. One of such formalisms, applicable to modelling systems with time dependencies, are time Petri nets by Merlin and Farber [1]. Numerous subclasses of time Petri nets have been proposed, i.e., 1-safe, bounded, unbounded, distributed nets, and many others.

For concurrent systems, the size of the state-space of the analysed system is a significant factor in the efficiency of the verification. The fact that verification methods need to explore the reachable state-space can lead to the state-space explosion problem. This follows from the fact that the size of the model grows exponentially with the number of the components of the system. Therefore, the development of methods that alleviate this problem is considered an important research subject.

There exist many papers dealing with model checking time Petri nets [2–11]. Most of them describe techniques based on explicitly-represented abstractions of the state spaces. The developed fully symbolic methods include decision diagrams-based ones: reachability verification for Integer Timed Petri Nets [10]

as well as LTL and ECTL verification for (1-safe) Distributed Time Petri Nets [6], and SAT-based methods for the distributed nets [6, 8].

This paper presents our first attempt to apply a satisfiability modulo theory (SMT) to verification of (bounded) time Petri nets, i.e., reachability testing for the nets restricted to disallow multiple enabledness of transitions. We use a bounded model checking technique (BMC), i.e., consider models truncated up to a specific depth, and transform the reachability problem into a test of satisfiability of a set of (integer) inequalities. Although the current version of the method was implemented, and we provide some preliminary experimental results, we consider our work to be in progress, and we plan on improving the efficiency of our implementation, as well as to extend it in a way allowing to test more involved reachability-related properties and to handle bounded nets with multiple enabledness of transitions [12].

The rest of the paper is organised as follows: in Sec. 2 we introduce bounded time Petri nets, their concrete state spaces and concrete models. The next Sec. 3 discusses reachability verification using SMT. In Sec. 4 and Sec. 5 we provide some preliminary experimental results and concluding remarks.

2 Time Petri Nets

Let $\mathbb{N}$ ($\mathbb{N}_+$) denote the set of (nonnegative) integers, and $\mathbb{R}$ ($\mathbb{R}_+$) - the set of (nonnegative) reals. We start with a definition of time Petri nets:

Definition 1. *A* time Petri net *(TPN for short) is a tuple* $\mathcal{N} = (P, T, F, cap, w, m^0, Eft, Lft)$, *where*

- $P = \{p_1, \ldots, p_{n_P}\}$ *is a finite set of places,*
- $T = \{t_1, \ldots, t_{n_T}\}$ *is a finite set of transition s.t.* $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation,*
- $cap : P \to \mathbb{N}_+ \cup \{\infty\}$ *is a partial capacity restriction,*
- $w : F \to \mathbb{N}_+$ *is an arc weight function,*
- $m^0 : P \to \mathbb{N}$ *with* $m^0(p) \leq cap(p)$ *for each* $p \in P$ *is an initial marking,*
- $Eft : T \to \mathbb{N}$, $Lft : T \to \mathbb{N} \cup \{\infty\}$ *are functions describing the earliest and the latest firing time of a transition, where for each* $t \in T$ *we have* $Eft(t) \leq Lft(t)$.

A TPN $\mathcal{N}$ is called *k-bounded*, for some $k \in \mathbb{N}_+$, if $cap(p) \leq k$ for each $p \in P$, and is called *bounded* if there is $k \in \mathbb{N}$ such that $\mathcal{N}$ is k-bounded. The value k is called a *bound* of $\mathcal{N}$. In what follows we consider bounded time Petri nets only.

For a transition $t \in T$ we define its *preset* $\bullet t = \{p \in P \mid (p, t) \in F\}$ and *postset* $t\bullet = \{p \in P \mid (t, p) \in F\}$, and consider only the nets such that for each transition the preset and the postset are nonempty. Moreover, we restrict to the nets satisfying the condition $\forall_{p \in P}\ cap(p) < 2 \cdot \min\{w(p, t) \mid t \in T \text{ s.t. } p \in \bullet t\}$ which prevents multiple enabledness of transitions.

We introduce the following notations and definitions:

- a *marking* of $\mathcal{N}$ is a function $m : P \to \mathbb{N}$,
- a transition $t \in T$ is *enabled* at a marking m ($m[t\rangle$ for short) if
  - ⋆ for each $p \in {\bullet}t$ it holds $w(p,t) \leq m(p)$, and
  - ⋆ for each $p \in t{\bullet}$ it holds $m(p) - w(p,t) + w(t,p) \leq cap(p)$ if $p \in {\bullet}t \cap t{\bullet}$, and $m(p) + w(t,p) \leq cap(p)$ otherwise,
- $en(m) = \{t \in T \mid m[t\rangle\}$ is the set of all the transitions enabled in a marking m of $\mathcal{N}$,
- the marking m' obtained by firing $t \in en(m)$ at m is given by
$$m'(p) = \begin{cases} m(p) & \text{if } (p,t) \notin F \wedge (t,p) \notin F; \\ m(p) - w(p,t) & \text{if } (p,t) \in F \wedge (t,p) \notin F \\ m(p) + w(t,p) & \text{if } (p,t) \notin F \wedge (t,p) \in F \\ m(p) - w(p,t) + w(t,p) & \text{if } (p,t) \in F \wedge (t,p) \in F. \end{cases}$$
The marking is denoted by $m[t\rangle$ as well, if it does not lead to misunderstanding,
- $newly\_en(m,t) = \{u \in T \mid u \in en(m[t\rangle) \ \wedge \ (\exists_{p \in (t{\bullet} \setminus {\bullet}t) \cap {\bullet}u} m(p) < w(p,u) \vee \exists_{p \in {\bullet}t \cap t{\bullet} \cap {\bullet}u} m(p) - w(p,t) < w(p,u) \vee \exists_{p \in u{\bullet} \cap {\bullet}t} m(p) + w(u,p) > cap(p))\}$ is a set of transitions which became (newly) enabled by firing t at m.

2.1 Concrete State Spaces and Models

The current state of the net is given by its marking and the time passed since each of the enabled transitions became enabled (which influences the further behaviour of the net). A *concrete state* σ of a TPN $\mathcal{N}$ is thus a pair $(m, clock)$, where $m : P \to \mathbb{N}$ is a marking, and $clock : T \to \mathbb{R}_+$ is a function which for each transition $t \in en(m)$ gives the time elapsed since t became enabled most recently. The set of all the states of $\mathcal{N}$ is denoted by Σ. The initial state of $\mathcal{N}$ is $\sigma^0 = (m^0, clock^0)$, where m^0 is the initial marking of $\mathcal{N}$, and $clock^0(t) = 0$ for each $t \in T$.

For $\delta \in \mathbb{R}_+$, let $clock + \delta$ denote the function given by $(clock + \delta)(t) = clock(t) + \delta$, and let $(m, clock) + \delta$ denote $(m, clock + \delta)$. The states of $\mathcal{N}$ can change due to a passage of time or due to a firing of a transition. The transition relation $\to \subseteq \Sigma \times (T \cup \mathbb{R}_+) \times \Sigma$ of the net $\mathcal{N}$ is thus given by:

- in a state $\sigma = (m, clock)$ a time $\delta \in \mathbb{R}_+$ can pass leading to a new state $\sigma' = (m, clock')$ (denoted $\sigma \xrightarrow{\delta} \sigma'$) iff $(clock + \delta)(t) \leq Lft(t)$ for each $t \in en(m)$ (*time successor relation*),
- in a state $\sigma = (m, clock)$ a transition $t \in T$ can fire leading to a new state $\sigma' = (m', clock')$ (denoted $\sigma \xrightarrow{t} \sigma'$) if $t \in en(m)$, $Eft(t) \leq clock(t) \leq Lft(t)$, $m' = m[t\rangle$, and for all $u \in T$ we have $clock'(u) = 0$ for $u \in newly\_en(m,t)$, and $clock'(u) = clock(u)$ otherwise (*action successor relation*).

Intuitively, the time successor relation does not change the marking of the net, but increases the clocks of all the transitions, provided no enabled transition becomes disabled by passage of time. Firing of a transition t takes no time (the only change it introduces to the clocks is reseting these related to the newly enabled transitions) and is allowed provided that t is enabled and its clock is

not smaller than $Eft(t)$ and not greater than $Lft(t)$. The structure $(\Sigma, \sigma^0, \rightarrow)$ is called a *concrete state space* of $\mathcal{N}$.

A *timed run* of $\mathcal{N}$ starting at a state $\sigma_0 \in \Sigma$ (σ_0-*run*) is a maximal sequence of concrete states, transitions and time passings $\rho = \sigma_0 \stackrel{a_0}{\rightarrow} \sigma_1 \stackrel{a_1}{\rightarrow} \ldots$, where $\sigma_i \in \Sigma$, and $a_i \in T \cup \mathbb{R}_+$ for all $i \in \mathbb{N}$. A state $\sigma_\star$ is reachable if there exists a σ^0-run ρ and $i \in \mathbb{N}$ such that $\sigma_\star = \sigma_i$, where σ_i is an element of ρ. The set of all the reachable states of $\mathcal{N}$ is denoted by $Reach_{\mathcal{N}}$.

Given a set of propositional variables PV, we introduce a *valuation function* $V : \Sigma \rightarrow 2^{PV}$ which assigns the same propositions to the states with the same markings. We assume the set PV to be such that each $q \in PV$ corresponds to an (in)equality $I(q)$ of the form $m(p) \sim a$ or $m(p) \oplus m(p') \sim a$, where $p, p' \in P$, $\sim \in \{\leq, <, =, >, \geq\}$, $\oplus \in \{+, -\}$, and $a \in \{0, 1, \ldots, 2k\}$, where k is a bound of $\mathcal{N}$. The function V is defined by $q \in V((m, clock))$ iff $I(q)$ holds for m. The structure $M(\mathcal{N}) = (\Sigma, \sigma^0, \rightarrow, V)$ is called a *concrete model of* $\mathcal{N}$.

3 Testing Reachability

The reachability problem for a time Petri net $\mathcal{N}$ consists in checking, given a property φ, whether $\mathcal{N}$ can ever be in a state where φ holds. The property is expressed in terms of propositional variables. Therefore, the problem can be translated into testing whether the set $Reach_{\mathcal{N}}$ contains a marking of certain features expressed by φ. Checking this can be performed by an explicit exploration of the concrete model for $\mathcal{N}$, but such an approach is usually very inefficient.

If a reachable state satisfying φ exists then proving this can be done using a part of the model only. This enables us to apply the bounded model checking method [13]. The main idea of testing reachability using BMC consists in searching for an *reachability witness* of a bounded length l (i.e., for a path leading from the initial state to a state satisfying φ). One of the possible approaches to this problem consists in generating a logical formula satisfiable iff such a witness exists, and checking its satisfiability using an appropriate solver. The most common choice here is to use a SAT-solver and a propositional formula, but alternatively an SMT-solver (i.e. a solver capable to test satisfiability of a first-order logic over a built-in theory) and the corresponding first-order logic can be used. We apply the second approach.

In order to check whether a state satisfying φ is reachable, we first replace the model $M(\mathcal{N})$ by a model with a restricted set of timed labels, i.e., with $\mathbb{R}_+$ substituted by $[0, c_{\max} + 1]$, where by $c_{\max}$ we mean the greatest finite value of Eft and Lft of the transitions in $\mathcal{N}$. It can be shown that such a model is bisimilar with $M(\mathcal{N})$. Moreover, we restrict to integer time steps only, as they are sufficient to prove reachability [14, 15]. Then, we represent the states of the obtained model using integer variables, encode its transition relation in terms of the logic over integer arithmetics, and encode all the paths of a given length l starting at the initial state of $\mathcal{N}$ as a formula $Path(l)$. We encode also the fact that the property φ holds in the last state of a path as a formula $encode\_prop(\sigma_l, \varphi)$ (we omit technical details of the encodings in the current version of the work), and

check satisfiability of the formula $\alpha := Path(l) \wedge encode\_prop(\sigma_l, \varphi)$. The above procedure is started from the length of a potential witness $l = 0$, and repeated iteratively up to (at most) $l = |M(\mathcal{N})|$. The process is stopped when the formula α is satisfiable, as this means that reachability of a state satisfying φ is proven, and therefore no further tests are necessary.

As usually in the case of bounded model checking methods, the above procedure can be inefficient if no state satisfying φ exists, since the depth of the part of the model considered (i.e., l) strongly influences the size of the encoding. Proving unreachability should therefore be done using a different algorithm we do not deal with in the current version of the paper.

3.1 Solving Other Reachability-Related Problems

The language of the propositions used enables to express not only simple properties of the form "a place p contains exactly x tokens", but also more involved ones, corresponding to various features of a marking of the net, e.g., "a place p contains more tokens than a place q", "the difference between the numbers of tokens in p and p' is greater than 5" etc. However, augmenting the net with an additional component enables testing also certain time-related properties.

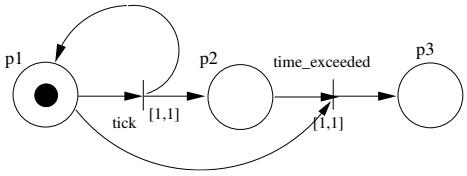


Fig. 1. An additional component

An example of such a component is shown in Fig. 1. If we set $cap(p2)$ and $w(p2, time\_exceeded)$ to a certain value $x \in \mathbb{N}_+$, the number of tokens in the place $p2$ corresponds to the number of time units (not greater than x) passed since the net started, and a token in $p3$ means that the time assumed has been exceeded. So, augmenting the net with this component enables to express in terms of reachability of a marking the properties like "a state satisfying φ is reachable in less / more than x time units". It is also possible to search for minimal / maximal time in which a state satisfying φ is reached, using the method similar to that described in [8].

4 Experimental Results

In this section, we present preliminary experimental results for the described method. The implemented tool consists of a module that generates an SMT input (the translation) for the SMT-solver, and a guiding module that performs the

execution of the verification task which calls the generator module and the SMT-solver for the increasing lengths of the paths, until the SMT-solver terminates with a satisfiable formula.

We also provide a comparison with Sift which is a module of the Tina toolbox [16] providing *on-the-fly* verification capabilities. The results we present were generated using the switch `-D`, i.e., reachability was tested on the graph of essential states [14], similarly as in our method.

The translation module is implemented in C++ language, and the guiding module is implemented as a simple UNIX shell script. The SMT-solver used in the experiments was Z3 (version 4.3.2) [17]. The experiments were executed on a Linux 3.9.8 system, equipped with AMD X6 FX-6100 processor, and 8GB of memory. However, for all our experiments we assume the time limit of 2000 seconds, and the memory limit of 2000 MiB.

4.1 Fischer's Mutual Exclusion Benchmark

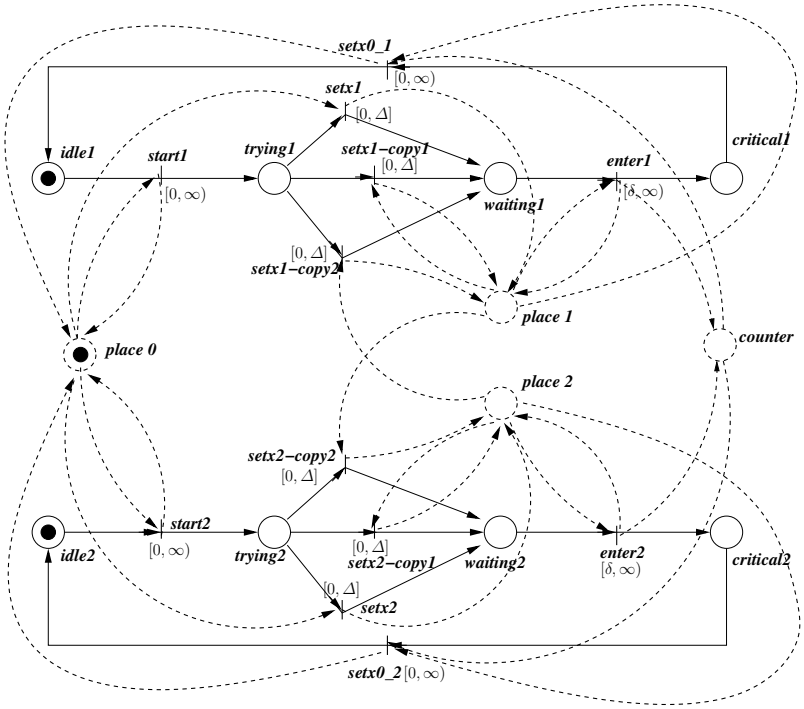


Fig. 2. A net for Fischer's mutual exclusion protocol for $n = 2$

In this benchmark we consider a net for the Fischer's mutual exclusion protocol [18] depicted in Figure 2. In our figures we assume that unless stated (or

denoted in the figure) otherwise, the weight of all the arcs is 1, and the capacity restriction is also 1.

The net models n processes with critical sections. When the i-th process enters (leaves) its critical section $critical_i$ (for $i \in \{1, \cdots, n\}$), the number of tokens in the *counter* place is incremented (decremented). We assume that $cap(counter) = n$. The mutual exclusion property of the protocol depends on the values of the time-delay constants δ and Δ, i.e., the property is preserved iff $\Delta < \delta$.

For this benchmark, we assume $\Delta = 2$ and $\delta = 1$, and we test the reachability of a marking m such that $m(counter) > 1$. The experimental results for this benchmark are presented in Table 1. The time is given in seconds (cpu time), and the memory in MiB. Where the assumed time or memory limit was exceeded, we denote this in the tables with the symbol $\star$.

4.2 Assembly Line Benchmark

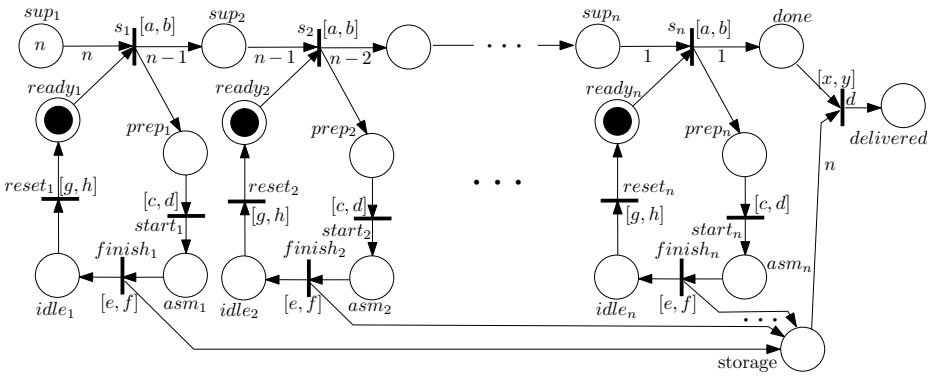


Fig. 3. Assembly line

A net for an abstract assembly line inspired by the generalised transfer chain model from [19] is presented in Figure 3. It consists of n assembly workers, and a supplier that provides resources needed in the assembly process. There are n resource packages which are represented by tokens in places sup_i for $i \in \{1, \cdots, n\}$. When an i-th worker receives the resource package via the s_i transition, it first prepares for the assembly process ($prep_i$), then performs the assembly itself (asm_i), and when finishing ($finish_i$) it also delivers the assembled product to the storage ($storage$). Then, it idles ($idle_i$), and becomes ready again ($ready_i$).

For all $p \in \{sup_1, \cdots, sup_n, storage, delivered\}$ we assume $cap(p) = n$. However, note that it could also be assumed that $cap(sup_i) = n - i + 1$ for $i \in \{1, \cdots, n\}$.

In our benchmark, we also assume the following time constraints in the system: $a = 0$, $b = 5$, $c = 0$, $d = 1$, $e = 0$, $f = 1$, $g = 0$, $h = 1$, $x = 0$, and $y = 100$. Then, we test the reachability of a marking m, where $m(done) > 0$ and $m(storage) > 0$. The length of the witness for this property is $n + 2$, where n is the number of assembly workers. The experimental results for this benchmark are presented in Table 2. Similarly as before, with $\star$ we mark the cases when the assumed time or memory limit was exceeded.

	SMT-BMC		TINA (sift)	
n	**time**	**memory**	**time**	**memory**
2	1.143	12.4	0.01	4.02
3	2.864	12.4	0.01	4.02
4	5.165	12.72	0.01	4.02
5	7.860	16.07	0.01	4.02
6	11.543	22.80	0.01	4.02
7	16.632	27.13	0.01	4.02
8	23.782	33.60	0.01	4.02
9	34.094	40.93	0.03	21.05
10	45.082	52.79	0.05	39.62
11	62.156	60.41	0.06	37.93
12	88.648	73.39	0.09	42.87
13	113.881	82.92	0.12	72.55
14	144.093	102.99	0.12	133.74
15	188.220	119.49	0.28	199.49
16	230.758	132.14	0.39	296.74
17	287.631	147.20	0.57	392.93
18	349.011	168.58	0.68	514.87
19	425.466	194.11	0.89	684.80
20	517.590	212.50	1.16	903.99
21	634.941	262.36	1.64	1202.24
22	775.141	282.44	1.91	1539.24
23	980.982	334.33	2.47	1877.18
24	1097.792	359.29	3.35	2387.43*
25	1311.071	385.24	4.29	2932.49*
26	1508.893	423.76	5.34	3715.05*
27	1721.292	452.26	5.45	3631.68*
28	1986.422	506.51	5.28	3606.80*
29	2242.422*	551.32		

Table 1. Results for Fischer's Protocol

	SMT-BMC		TINA (sift)	
n	**time**	**memory**	**time**	**memory**
2	1.073	15.70	0.01	4.02
3	2.387	27.52	0.01	4.02
4	4.574	43.74	0.01	4.02
5	8.071	60.08	0.01	4.02
6	13.315	85.56	0.01	4.02
7	20.373	106.97	0.02	4.02
8	30.039	142.03	0.03	28.24
9	41.742	172.50	0.05	38.12
10	54.235	202.65	0.13	95.74
11	71.999	235.27	0.25	214.80
12	94.619	295.05	0.56	519.68
13	124.712	336.16	1.28	1202.62
14	159.626	382.76	2.88	2664.55*
15	184.786	428.20	4.33	3552.93*
16	239.593	477.61	5.44	3891.19*
17	314.841	570.93	6.14	3891.19*
18	371.564	632.92		
19	479.219	686.16		
20	581.981	753.00		
21	717.623	818.28		
22	920.044	894.76		
23	1135.285	964.96		
24	1238.729	1115.49		
25	1674.850	1195.32		
26	1860.204	1280.82		
27	2219.660*	1374.26		

Table 2. Results for Assembly Line system

It can be seen from the above results that our implementation has much longer execution times than Tina. However, our method requires less memory than the corresponding Tina execution, and the differences are also significant.

One should also comment on the lack of any comparison with the implementation of the SAT-based reachability verification described in [8]. The paper mentioned provides experimental results for the Fischer's mutual exclusion protocol obtained on a computer equipped with Intel Pentium Dual CPU (2.00 GHz) and 2 GB of main memory, and confirms verifying the system consisting of 40 processes in 2999.5 seconds and using 1153.2 MB of memory. This seems to be a far better result. However, it should be noticed that the method of [8] was developed for distributed TPNs, of a semantics aimed at making verification more efficient than in the standard case (i.e., of the one in which clocks are assigned to the processes of the net instead of to the transitions which enables to reduce their number from $6n$ to $n + 1$). The distributed nets are also 1-safe, which reduces further the complexity of their description.

5 Final Remarks

We have presented a preliminary version of a reachability verification method for (bounded) time Petri nets, based on bounded model checking and SMT. In our future work we are going to build upon this method an approach allowing to verify more involved properties. In the preliminary comparison with Tina our method performed efficiently in terms of the memory consumption when testing the reachability property, which is very promising for the mentioned extension of our work.

Acknowledgements. Artur Męski acknowledges the support of the EU, European Social Fund. Project PO KL "Information technologies: Research and their interdisciplinary applications" (UDA-POKL.04.01.01-00-051/10-00).

References

1. Merlin, P., Farber, D.J.: Recoverability of communication protocols – implication of a theoretical study. IEEE Trans. on Communications **24(9)** (1976) 1036–1043
2. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. on Software Eng. **17(3)** (1991) 259–273
3. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time Petri nets. In: Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03). Volume 2619 of LNCS., Springer-Verlag (2003) 442–457
4. Boucheneb, H., Hadjidj, R.: CTL* model checking for time Petri nets. Theoretical Computer Science **353(1)** (2006) 208–227
5. Lime, D., Roux, O.H.: Model checking of time Petri nets using the state class timed automaton. Discrete Event Dynamic Systems **16(2)** (2006) 179–205
6. Męski, A., Penczek, W., Półrola, A., Woźna-Szcześniak, B., Zbrzezny, A.: Bounded model checking approaches for verificaton of distributed time Petri nets. In: Proc. of the Int. Workshop on Petri Nets and Software Engineering (PNSE'11), University of Hamburg (2011) 72–91

7. Penczek, W., Półrola, A.: Abstractions and partial order reductions for checking branching properties of time Petri nets. In: Proc. of the 22nd Int. Conf. on Applications and Theory of Petri Nets (ICATPN'01). Volume 2075 of LNCS., Springer-Verlag (2001) 323–342
8. Penczek, W., Półrola, A., Zbrzezny, A.: SAT-based (parametric) reachability for a class of distributed time Petri nets. In: Trans. on Petri Nets and Other Models of Concurrency IV. Volume 6550 of LNCS. Springer-Verlag (2010) 72–97
9. Virbitskaite, I.B., Pokozy, E.A.: A partial order method for the verification of time Petri nets. In: Fundamental of Computation Theory. Volume 1684 of LNCS. Springer-Verlag (1999) 547–558
10. Wan, M., Ciardo, G.: Symbolic reachability analysis of integer timed Petri nets. In: Proc. 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM). (2009) 595–608
11. Yoneda, T., Ryuba, H.: CTL model checking of time Petri nets using geometric regions. IEICE Trans. Inf. and Syst. **3** (1998) 1–10
12. Boyer, M., Diaz, M.: Multiple enabledness in Petri nets with time. In: Proc. of the 9th Int. Workshop on Petri Nets and Performance Models (PNPM'01). (2001) 219–228
13. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99). Volume 1579 of LNCS., Springer-Verlag (1999) 193–207
14. Popova-Zeugmann, L.: Time Petri nets state space reduction using dynamic programming. Journal of Control and Cybernetics **35(3)** (2006) 721–748
15. Janowska, A., Penczek, W., Półrola, A., Zbrzezny, A.: Towards discrete-time verification of time Petri nets with dense-time semantics. In: Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'11), Bialystok University of Technology (2011) 215–228
16. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA - construction of abstract state spaces for Petri nets and time Petri nets. International Journal of Production Research **42(14)** (2004)
17. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of the 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). Volume 4963 of LNCS., Springer-Verlag (2008) 337–340
18. Abadi, M., Lamport, L.: An old-fashioned recipe for real time. In: REX workshop on Real-Time: Theory in Practice. Volume 600 of LNCS., Springer-Verlag (1991) 1–27
19. Tsinarakis, G., Tsourveloudis, N., Valavanis, K.: Modular Petri net based modeling, analysis, synthesis and performance evaluation of random topology dedicated production systems. Journal of Intelligent Manufacturing **16**(1) (2005) 67–92

A Bi-objective Optimization Framework for Query Plans

Piotr Przymus[1], Krzysztof Kaczmarski[2], and Krzysztof Stencel[3]

[1] Nicolaus Copernicus University, Poland, eror@mat.umk.pl
[2] Warsaw University of Technology, Poland, k.kaczmarski@mini.pw.edu.pl
[3] The University of Warsaw, Poland, stencel@mimuw.edu.pl

Abstract. Graphics Processing Units (GPU) have significantly more applications than just rendering images. They are also used in general-purpose computing to solve problems that can benefit from massive parallel processing. However, there are tasks that either hardly suit GPU or fit GPU only partially. The latter class is the focus of this paper. We elaborate on hybrid CPU/GPU computation and build optimisation methods that seek the equilibrium between these two computation platforms. The method is based on heuristic search for bi-objective Pareto optimal execution plans in presence of multiple concurrent queries. The underlying model mimics the commodity market where devices are producers and queries are consumers. The value of resources of computing devices is controlled by supply-and-demand laws. Our model of the optimization criteria allows finding solutions of problems not yet addressed in heterogeneous query processing. Furthermore, it also offers lower time complexity and higher accuracy than other methods.

1 Introduction

General-Purpose computing on Graphics Processing Units (GPGPU) involves utilization of graphics processing units (GPU) in tasks traditionally handled by central processing units (CPU). GPUs offer a notable processing power for streams.

Execution of database queries is an example of a successful application of GPGPU. The current research focuses on using the GPU as a co-processor [5]. GPU as co-processor may accelerate numerous database computations, e.g. relational query processing, query optimization, database compression or supporting time series databases [5,13,14].

An application of GPU requires transferring data from the CPU memory to the graphical device memory. The data transfer is usually time-consuming. It may diminish the gain of the acceleration credited to GPU. This situation can be improved by using lightweight compression methods that can significantly reduce the costs associated with communication [13,14]. However, this does not solve all the problems. In particular, GPU is optimized for numerical computation. Thus, only selected operations will benefit from GPU. Small data sets are another problem. For such sets the data transfer may dominate processing time and

destroy the performance gain. Therefore, joint processing capabilities of both CPU and GPU are worth considering. Furthermore, as it is common to have more than one GPU in a computer, a potential use of various GPU devices should be considered. This type of query plans is called heterogeneous.

The previous research efforts focused on the creation of query plans based on a cost model. This approach finds plans with the best throughput. However, it does not allow modelling all phenomena that can occur in heterogeneous systems. Performing a query as soon as possible is not always cost effective [8]. For this reason, we propose a query processing model based on concepts of markets that are known to be suitable for describing the interactions in a heterogeneous world. They have already gained a considerable interest in the context of task processing in heterogeneous systems [6]. In market models, manufacturers (processing devices) compete with each other for customers (query plans). Similar competition occurs among customers.

In this paper, we propose a query optimization model based on the commodity market. A query plan is bi-objectively optimized to minimize: the processing time and the value of consumed resources. For the user, a small difference in execution time can be negligible. Thus, it is worth optimizing a query, so that the execution time satisfies the user while other costs are minimized. In this case, the cost may be, e.g. the responsiveness of the system, power consumption, heat production, etc. One can also consider expressing the cost in financial terms.

2 Preliminaries

2.1 GPU and Heterogeneous query processing

From the parallel processing's point of view, CPU accompanied by a GPU co-processor is a *shared nothing architecture*. A GPU card has its own memory or a separate area in the CPU main memory. Thus, the data has to be explicitly transferred from the CPU main memory to the GPU main memory. Similarly, the results produced by GPU have to be transferred back to the CPU main memory. This data transfer often introduces significant overhead. Thus, it is important to include the transfer cost in the total execution time of an operation. This cost is also a component of the execution time prediction.

Contemporary computer systems often include more than one GPU. Then, it is possible to combine multiple computational units in a single query plan. Such plans are called *heterogeneous query processing*. Each device may have a different communication cost (e.g. PCIe or shared memory) with the CPU main memory. Furthermore, devices can often communicate directly between each other. Therefore, the main problem of heterogeneous query processing is the construction of such a query plan that uses only computational units from which query performance will benefit most and yet will minimize used resources.

Bress et. al. [5] identified problems of hybrid (CPU/GPU) query processing which are also true in heterogeneous query processing:

Problem 1 Execution Time Prediction - as multiple database operations may be executed concurrently it is hard to predict influence of concurrent tasks on execution times.

Problem 2 Critical Query - since the GPU memory, the concurrent GPU kernels execution and the PCIe bus bandwidth are all limited, only the *critical queries* should be selected to use GPU (i.e., queries that benefit from GPU usage and are *important* from global perspective).

Problem 3 Optimization Impact - as concurrent heterogeneous queries will influence each other, it is important to consider this aspect in the planning process.

2.2 Commodity market approach in query processing context

In this paper we address these problems by showing that they may be solved by applying a supply-and-demand pricing model taken from a commodity market. In such a market resource owners (processing devices) price their assets and charge their customers (queries) for consumed resources. Other pricing models may also be used [6].

In the supply-and-demand model when supply (available resources) or demand (needed resources) changes, the prices will be changed until an equilibrium between supply and demand is found. Typically the value of a resource is influenced by: its strength, physical cost, service overhead, demand and preferences [6]. A consumer may be charged for various resources like CPU cycles, memory used, the bus usage or the network usage. Typically, a broker mediates between the resource owners and the consumer. The resource owners announce their valuation and the resource quality information (e.g. estimated time) in response to the broker's enquiry. Then, the broker selects resources that meet the consumer utility function and objectives, like cost and estimated time constraints or minimization of one of the objectives.

2.3 Bi-objective optimization

Bi-objective optimization is a problem where optimal decisions need to be taken in the presence of trade-offs between two conflicting objectives. It is a special case of multiple criteria decision making. Typically there are no solutions that meets all objectives. Thus, a definition of an optimum solution set should be established. In this paper we use the predominant Pareto optimality [11]. Given a set of choices and a way of valuing them, the Pareto set consists of choices that are Pareto efficient. A set of choices is said to be Pareto efficient if we cannot find a reallocation of those choices such that the value of a single choice is improved without worsening values of others choices. As bi-objective query optimization is NP-hard, we need an approximate solution [12].

3 Heterogeneous Query Planer

The main aim of the new query planner is to propose a solution to the problems listed in Section 2.1, i.e., Execution Time Prediction, Critical Query and

Optimization Impact. Furthermore, this planner also addresses heterogeneous GPU cards and distributed processing. In this paper we propose a method to build heterogeneous query plans based on the economics of commodity markets. It is characterized by the fact that the resource producers determine the cost of their resources and resource consumers jostle for resources. Furthermore, the resources owners provide information on the quality of their resources, i.e., the estimated processing time.

3.1 Notation

Table 1 contains a summary of the notation used in this paper. Assume a set of units U, a logical query sequence QS_{log} and a dataset D. The goal is to build a heterogeneous query sequence. Let QS_{het} be a heterogeneous query sequence defined in Equation (1). Let D_{k+1} be the data returned by an operation $A^{o_k}_{u_{i_k}}(D_k)$. The first row of QS_{het} is created by replacing each operation $o_i \in QS_{log}$ with an algorithm $A^{o_i}_{u_j} \in AP_{o_i}$. The second row is created by inserting an operation $M_{u_k,u_{k'}}(D)$ that copies the output of an algorithm on the unit u to the input of the current algorithm on the unit u'.

Symbol	**Description**
$U = \{u_1, u_2, \dots u_n\}$	set of computational units available to process data
D	dataset
$M_{u_k,u_{k'}}(D)$	**if** $u_k \neq u_{k'}$ move D from u_k to $u_{k'}$ **else** pass
$o_i \in O$	database operation o_i from set of operations O
$A^{o_i}_{u_k}$	algorithm that computes the operation o_i on u_k
$AP_{o_i} = \{A^{o_i}_{u_1}, A^{o_i}_{u_2}, \dots, A^{o_i}_{u_n}\}$	algorithm pool for the operation o_i
$t_{run}(A^{o_i}_{u_j}, D)$	estimated run time of the algorithm $A^{o_i}_{u_j}$ on the data D
$t_{copy}(M_{u_i,u_j}, D))$	estimated copy time of the data D from u_i to u_j
$c_{run}(A^{o_i}_{u_j}, D)$	estimated run cost of the algorithm $A^{o_i}_{u_j}$ on the data D
$c_{copy}(M_{u_i,u_j}, D)$	estimated copy cost of the data D from u_i to u_j
$QS_{log} = o_1 o_2 \dots o_n$	logical query sequence
QS_{het}	heterogeneous query sequence see Eq. 1
f_t, g_t	estimated algorithm run time and copy time see Sec. 3.2
f_c, g_c	estimated algorithm run cost and copy cost see Sec. 3.3
f_b, g_b	estimated algorithm run and copy bi-objective scalarization see Sec. 3.4
$F_x(QS_{het})$	sum of f_x and g_x over columns of QS_{het} where $x \in \{t, c, b\}$ see Eq. 2

Table 1: Symbols used in the definition of our optimisation model

$$QS_{het} = \begin{pmatrix} A^{o_1}_{u_{i_1}}(D_1), & A^{o_2}_{u_{i_2}}(D_2), & A^{o_3}_{u_{i_3}}(D_3), & \dots, & A^{o_n}_{u_{i_n}}(D_n) \\ M_{u^*,u_{i_1}}(D_1), & M_{u_{i_1},u_{i_2}}(D_2), & M_{u_{i_2},u_{i_3}}(D_3), & \dots, & M_{u_{i_n},u^*}(D_n) \end{pmatrix} \quad (1)$$

$$F_x(QS_{het}) = \sum_{A_u^o(D)} f_x(A_u^o, D) + \sum_{M_{u',u''}(D)} g_x(M_{u',u''}, D) \tag{2}$$

3.2 Single Objective Heterogeneous Query Planer

Procedure *OptimalSeq*(QS_{log}, u^*, x)

Input: $QS_{log} = o_1 o_2 o_3 \dots o_n$ - logical query sequence, u^* - base unit, $x \in$ { t - time, c - cost, b - bioptimization } - optimization type
Result: QS_{hybrid}

```
1  seq_list = [];
2  for u in U do
3  |  Q_u = S_u(QS_log, u*);
4  |  QF_u = F_x(Q_u) ;                                   /* e.g. F_t(Q_u) */
5  |  append (u, Q_u, QF_u) to Seq_list;
6  end
7  QS_hybrid = pop minimum Q_u (by QF_u) sequence from Seq_list;
8  for (u, Q_u, QF_u) in Seq_list do
9  |  A, B, C = DiffSeq (QS_hybrid, Q_u, u, x);
10 |  val, start, end = MaxSubseq(A, B, C);
11 |  if val > 0 then
12 |  |  Q_hybrid(start : end) = Q_u(start : end) ;   /* subarray subsitution */
13 |  end
14 end
15 return Q_base
```

In this section, we introduce the algorithm that searches for a heterogeneous query plan, i.e., a plan that operates on more than two devices.

For simplicity let us assume that $x = t$, $f_t(A_u^o, D_i) = t_{run}(A_u^o, D_i)$ and $g_t(M_{u,u'}, D) = t_{copy}(M_{u,u'}, D)$. Later in this article we will define functions f_c, g_c and f_b, g_b to fit the model of the commodity market and the bi-objective optimization. Let $S_u(QS_{log}, u^*)$ return such QS_{het} that each operation $o_i \in QS_{log}$ is replaced with an algorithm from unit u algorithm pool, i.e., $A_u^{o_i} \in AP_{o_i}$ and the base device is set to u^*. Note that there is a specially designated computing unit u^* from which the processing starts. It also collects the data in the end of processing, since the GPU computing is controlled by a CPU side program.

The algorithm *OptimalSeq* starts by creating a query sequence for each computing unit and estimating the processing cost for each item of this sequences (lines 2-6). Next, one sequence (which minimizes $F_x(Q_u)$) is selected as the base sequence. It will be improved in later steps (line 7). Then, the algorithm iterates over remaining query sequences in QS_{hybrid} in order to find such segments in the remaining query sequences which improve original sequence (by replacing

Procedure DiffSeq(seq_{base}, seq_u, u, x)

Input: seq_{base} - base sequence, seq_u - unit query sequence, u - seq_u unit, $x \in \{$ t - time, c - cost, b - bioptimization $\}$ - optimization type
Result: A - operations improvement array; B, C - copy to/from unit arrays

```
1 A, B, C = [], [], [];
2 for i in enumerate columns seq_base do
3 |   A^o_{u_b}(D_i), M_{u_f,u_t}(D_i) = seq_base[i];
4 |   A^o_u(D_i), M_{u,u}(D_i) = seq_u[i];
5 |   append f_x(A^o_{u_b}, D_i) − f_x(A^o_u, D_i) to A ;   /* e.g. f_t(A^o_u, D) */
6 |   append g_x(M_{u_f,u}, D_i) to B ;                     /* e.g. g_t(M_{u,u'}, D) */
7 |   append g_x(M_{u,u_t}, D_i) to C;
8 end
9 return A, B, C
```

corresponding segment of the original sequence). This is done by calculating the improvement and copy cost arrays in DiffSeq and finding maximal sequence segment in MaxSubseq. A following variant of the proposed algorithm should also be considered. Suppose that only one query sequence segment may be inserted (i.e., choose one sequence segment from remaining $k-1$ sequences with the biggest), this minimizes number of involved computational units and reduces overall communication costs.

The procedure DiffSeq simply calculates element wise difference between two query sequences $f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i)$ and copy costs from/to unit. The procedure MaxSubseq is based on Kadane's algorithm for maximum subarray problem [1]. It scans through the improvement array, computing at each position the maximum subsequence ending at this position. This subsequence is either empty or consists of one more element than the maximum subsequence ending at the previous position. Additionally, the *copy to* and *copy from* costs are included in the calculation of the maximum subsequence (B and C arrays). The algorithm returns the maximum improvement for a subsequence (which may be zero if the subsequence does not improve the original query), the start and end items of subsequence.

The complexity of *OptimalSeq* is $O(k * n)$ where k is the number of devices (usually small) and n is the number of operations of the sequence. S_u,F_x, DiffSeq and MaxSubseq have the complexity $O(n)$.

3.3 Economics in Heterogeneous Environment

To cope with the problems mentioned in Section 2.1, additional criteria are necessary – in this work an approach based on a simple economic model is proposed. Each consumer (client) has a query budget that can be used to pay for the resources used to process queries. Each computational unit is a service provider (producer) of services available in units algorithm pool AP_{u_i}. Each service provider establishes its own pricing for execution of any service from AP_{u_i}. Pricing of the service depends on:

Procedure MaxSubseq(A, B, C)

```
   Input: A - operations improvement array; B, C - copy to/from unit arrays
   Result: maximum_improvment, start, end
 1 max_ending_here = max_so_far = 0 ;
 2 begin = tbegin = end = 0 ;
 3 for i, x in enumerate(A) do
 4 |   max_ending_here = max(0, max_ending_here + x) ;
 5 |   if max_ending_here = 0 then
 6 |   |   tbegin = i ;
 7 |   |   max_ending_here -= B[i];
 8 |   end
 9 |   if max_ending_here - C[i] >= max_so_far then
10 |   |   begin = tbegin ;
11 |   |   end = i ;
12 |   |   max_so_far = max_ending_here ;
13 |   end
14 end
15 return max_so_far - C[end], begin, end
```

- the estimation of needed resources (the size of the data D, the performance of the task $A_{u_i}^{o_i}$),
- pricing of needed resources (the load of device u_i – the greater the load on the device, the higher cost of using the device),
- the preference of the device (e.g. device may prefer larger jobs and/or tasks that give a greater acceleration on the GPU).

First, pricing for using the resources of computational unit is established. This depends on the previous load of the device: the higher demand for computational unit, the higher price for using it. This is a periodic process which calculates prices every Δt_{up} seconds by calculating computational unit price P_u. Let $0 < L_{curr} < 1$, $0 < L_{prev} < 1$ be current and previous computational unit load factors. Additionally, let L_{th} be a threshold below which prices should decrease, and P_{min} be the minimal price. Then the price is calculated using the following formula[4]:

$$P_u := \begin{cases} max(P_{min}, P_u \cdot (1 + \frac{\Delta P_u}{(1-\Delta U)}) & \text{if } (\Delta P > 0 \wedge \Delta U > 0) \vee (\Delta P < 0), \\ P_u & \text{otherwise,} \end{cases} \tag{3}$$

where $\Delta P_u = L_{current} - L_{threshold}$ and $\Delta U_u = L_{current} - L_{Previous}$. This is similar to the dynamic pricing model proposed in [16] with exception to the pricing formula i.e., we use $max(P_{min}, P_u \cdot (1 + \frac{\Delta P_u}{(1-\Delta U)})$ instead of $max(P_{min}, P_u \cdot (1 + \Delta P_u))$, this modification reduces the excessive growth of prices.

To reflect the preference of the device in price we need to define a function returning speedup factor between base device u^* (defined in the previous section)

[4]Slightly abusing notation we will also denote the new price by P_u.

and current device: $speedup(A_u^o, D_i) = t_{run}(A_{u^*}^o, D_i)/t_{run}(A_u^o, D_i)$. Then we define a cost function as $c_{run}(A_u^o, D_i) = \frac{\#D_i}{speedup(A_u^o, D_i)} \cdot P_u$, where $\frac{\#D_i}{speedup(A_u^o, D_i)}$ part combines the estimation of needed resources and the preference of the device. A computational unit with high speedup on given operation will get a discount per data size when pricing this operation. Similarly, operations with a lower speedup factor will be charged more per quantity. Additionally it is observed [13] that often speedup depends on the size of processed data (usually low speed-up on small datasets) so discount depends on data size.

It is also important to include cost of data transfer, let us define it as

$$c_{copy}(M_{u,u'}, D) := \begin{cases} 0 & \text{if u,u' share memory} \\ \frac{\#D_i}{bandwidth(\#D_i, u, u')} \cdot (P_u + P_{u'})/2 & \text{otherwise} \end{cases}$$

where *bandwidth* returns estimated bytes per second between u and u' computational units. If direct data transfer is not available between u and u' devices, then transit device will be used (e.q. two GPU cards without direct memory access will communicate using CPU RAM).

Now let $f_c(A_u^o, D_i) = c_{run}(A_u^o, D_i)$ and $g_c(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)$. A solution minimizing the cost may be found under the previous assumptions and using procedure *OptimalSeq*.

3.4 Bi-objective Heterogeneous Query Planer

As finding Pareto optimal bi-objective query plan is NP-hard (bi-objective shortest path problem) [12], we will use previously described *OptimalSeq* single objective approximation algorithm and extend it to bi-objective case.

We will use *a priori* articulation of preference approach which is often applied to multi-objective optimization problems. It may be realized as the scalarization of objectives, i.e., all objective functions are combined to form a single function. In this work we will use *weighted product* method, where weights express user preference [11]. Let us define:

$$f_b(A_u^o, D) = c_{run}(A_u^o, D)^{w_c} \cdot t_{run}(A_u^o, D)^{w_t},$$
$$g_b(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)^{w_c} \cdot t_{copy}(M_{u,u'}, D)^{w_t}.$$

where w_t and w_c are weights which reflect how important cost and time is (the bigger the weight the more important the feature – values of f_b, g_b are higher than 1). It is worth to mention that a special case with $w_t = w_c = 1$ (i.e., without any preferences) is equivalent to Nash arbitration method (or objective product method) [11].

4 Preliminary Experimental Results

4.1 Simulation settings

In order to evaluate this model we prepared a *proof of concept* and evaluated it using custom developed simulation environment. Simulation environment was de-

Device	o_1	o_2	o_3	o_4	o_5	o_6
GPU1	20	11	6	0.38	14	15
GPU2	5	11	6	0.33	4.66	5
CPU2	1	1	1.09	1	1.27	1.36

(a) Average speedup of operation o_i on given device compared to CPU1

Option	CPU1	CPU2	GPU1	GPU2
Unit threshold	0.75	0.75	0.4	0.4
Unit minimal price	5	5	70	70

(b) Pricing model configuration

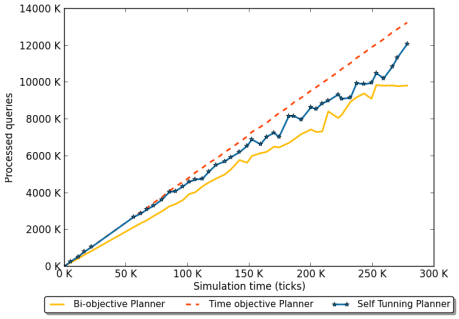
Table 2: Simulation environment configuration

veloped using Python and SimPy framework. All presented experiments are derived from simulation. There where four devices defined in environment: CPU1, CPU2, GPU1, GPU2. Data transfer bandwidth between CPU* $\leftrightarrow$ GPU* was measured on real system, bandwidth of GPU1 $\leftrightarrow$ GPU2 was calculated using CPU1 as transit device. Following weights in bi-objective scalarization were used $w_t = w_c = 1$ (i.e., without any preferences setting). Other settings of the simulation environment are gathered in Tables 2b and 2a. Simulation environment generates new query sequences, when spawn event occurs. Spawn event is generated randomly in a fixed interval and generates randomly set of query sequences (with fixed maximum). Every query sequence consists of maximally six operations and operates on random data volume. In the simulation the processed data size has a direct (linear) influence on processing speed. Each device has got a limited number of resources; a database operation can be performed only if needed resources are available. In other cases the operation is waiting. After generating desired number of query sequences the simulation stops spawning of new tasks and waits until all generated query sequences are processed.

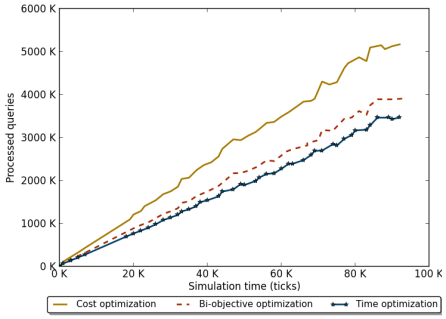
4.2 Simulation Results

Figure 1a presents simulated execution time of three scheduling frameworks processing a pool of generated sequences of queries. To each generated query sequence an optimization criterion (time, cost or bi-optimization) was assigned with equal probability 1/3. Optimization criteria are only used if query scheduling framework supports it, otherwise default criteria is used. All scheduling frameworks process exactly the same pool of generated query sequences.

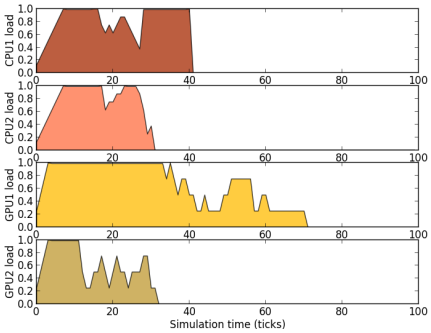
Compared frameworks are based on *OptimalSeq* algorithm but use different objective function. *Time objective planner* uses only f_t and g_t functions as optimization criteria; this means that it has no idea on load of each of devices. *Self Tuning Planner* is based on idea presented in Breß et. al. [3], i.e. it maintains a list of observed execution times on data D for each algorithm $A_{u_j}^{o_i}$. Observations are interpolated (using e.q. cubic splines) to form new estimated execution time function. And finally *Bi-objective planner* is a proof of concept implementation of the model described in this work.



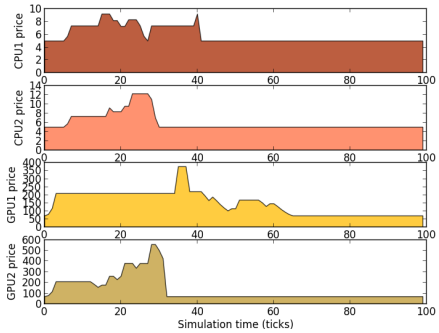
(a) Simulated efficiency of Bi-objective Heterogeneous Query Planer, Time based Query Planer and Self-Tuning Query Planner



(b) Efficiency of Bi-objective Heterogeneous Query Planer for various optimization tasks



(c) Simulated load of devices ($0 < load < 1$)



(d) Pricing of device

Fig. 1: Simulation results. Note that time is in simulation ticks.

As expected *Time Objective Planner* is the slowest one since it has no knowledge on the current load of devices. A solution suggested in [3] performs better. However, there are two problems with this approach: first it adds an additional overhead due to the interpolation of the observed execution times [3]; Secondly as may be observed in 1a it takes some time before it adapts to a new situation (in early stage it performs similarly to the *Time Objective Planner*). This is due the fact that it does not immediately react to load change of the device. Instead, it has to gather enough observations before adapting. The best performance is gained when using *Bi-objective planner*, this is due to the three types of optimization and the cost model which assures proper load balancing.

As our framework support different types of optimization in the Figure 1b, we present an impact of optimization type on processing performance. As it may be observed, time optimization is the most appropriate for query processing with high priority or with execution time constraint (like interactive queries or ad hoc data mining). Cost optimization is appropriate for operations with low priority or without time constraint (like batch processing or periodic jobs). Optimization

of both cost and time (without preferences) leads to moderate processing speed but with better load balancing which is discussed later.

As proposed economic model is an important part of presented framework, in Figure 1 interaction between device load 1c and established device pricing 1d is illustrated. Notice how increased load influences unit pricing according to the formula 3. It is worth noting that pricing model may be tuned for specific applications (see Table 2b for this simulation settings).

4.3 Discussion

In Section 2.1 we cite three challenges of Hybrid Query Processing initially presented in Breß et.al. [3]. As our bi-objective optimization framework was designed in order to address this challenges, an evaluation in the context of the former mentioned problems is needed. We address *Critical Query* problem by allowing different optimization targets for queries. Choosing time optimisation allows to a priori articulate importance of a query. Also, the bi-objective optimization tends to promote queries which may gain more on particular devices (due to the cost-delay trade-off and the fact that the cost objective is designed to promote tasks with greater speed-up 3.3). The problem of *Execution Time Prediction* is addressed indirectly with bi-objective optimisation. This is because the bi-objective optimisation combines the cost objective function, which uses a current device load when pricing a device, with the execution time objective. So in most cases it is preferred to optimize both cost and time (without preferences towards any) through time/cost trade-off. Lastly different types of optimization apply also to *Optimization Impact* challenge. Choosing optimization criteria specifies a possible impact on other queries. Although, the preliminary results are promising and seem to confirm this, an extended evaluation is needed in future.

5 Related Work

Multiobjective query optimization was considered i.a. in Stonebraker et.al. [17] where a wide-area distributed database system (called Mariposa) was presented. An economic auction model was used as cost model. To process a query a user supplied a cost-delay trade-off curve. Because defining this kind of input data was problematic Papadimitriou et.al. proposed a new approach where an algorithm for finding ϵ-Pareto optimal solutions was presented. The solution was that a user would manually choose one of presented solutions. This work differs both in an optimisation method and an economic model involved.

In our framework a user supplies an optimization objective for a query a priori (time, cost or bi-objective). Also as our model addresses the optimisation of co-processing interaction a simpler commodity market model could be used instead of a bidding model.

An extended overview on utilization of a GPU as a coprocessor in database operations may be found in [3]. Breß et. al. [3] proposed a framework for optimisation of hybrid CPU/GPU query plans and present two algorithms for

constructing hybrid query sequences. The first algorithm selected the fastest algorithm for every element of a query sequence (including the cost of transfer between devices) with complexity $O(n)$. Unfortunately, this algorithm had two flaws [3]: the constructed plan could generate too frequent data transfers between devices, which may significantly affect the performance of data processing and also an optimal plan was not always generated. To overcome those problems they proposed the second algorithm. It searched for a continuous segment of operations on GPU that could improve the base CPU sequence. In order to find an optimal solution this algorithm generates all possible GPU sequences. Its complexity is obviously higher: $O(n^2)$. Our work extends this approach by allowing possible many various co-processing devices (Heterogeneous Query Planer in Section 3.1). Secondly our work incorporates commodity market model as well as bi-objective optimisation for better performance overcoming problems mentioned in 2.1. Additionally, the algorithm *OptimalSeq* presented in our work may be used to produce a similar solution as the second algorithm by Breß et.al.[3] but with better complexity (in case of two devices $O(n)$).

It is worth to mention two surveys: the first one describing economic models in grid computing [6] and the second one describing methods for multi-objective optimisation [11].

6 Conclusions and Future Work

In this paper, we proposed a bi-objective optimization framework for heterogeneous query plans. We also presented an algorithm for creating query sequences in a heterogeneous environment with a single objective. This algorithm may be used to construct query sequences similar to [3] but with better complexity. For the purposes of this bi-objective optimization we designed a model including time and cost objectives function. The cost objective function and pricing model is build on foundations of commodity market economic model.

The preliminary experiments are very promising. We achieved good load balancing of the simulated devices combined with better optimization results.

In future work, an extended evaluation of the presented framework is needed, including; examination of parameters' influence on the model behaviour, careful assessment against Hybrid Query challenges. Another interesting field is extension of this model beyond CPU/GPU co-processing. Finally, the framework will be evaluated in a prototype time-series database [14,13].

References

1. J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, Sept. 1984.
2. S. Breß, F. Beier, H. Rauhe, E. Schallehn, K.-U. Sattler, and G. Saake. Automatic selection of processing units for coprocessing in databases. In *Advances in Databases and Information Systems*, pages 57–70. Springer, 2012.

3. S. Breß, I. Geist, E. Schallehn, M. Mory, and G. Saake. A framework for cost based optimization of hybrid cpu/gpu query plans in database systems. *Control and Cybernetics*, pages 27–35, 2013.
4. S. Breß, S. Mohammad, and E. Schallehn. Self-tuning distribution of db-operations on hybrid cpu/gpu platforms. *Grundlagen von Datenbanken, CEUR-WS*, pages 89–94, 2012.
5. S. Breß, E. Schallehn, and I. Geist. Towards optimization of hybrid cpu/gpu query plans in database systems. In *New Trends in Databases and Information Systems*, pages 27–35. Springer, 2013.
6. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.
7. W. Fang, B. He, and Q. Luo. Database compression on graphics processors. *Proceedings of the VLDB Endowment*, 3(1-2):670–680, 2010.
8. D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *ACM Sigmod Record*, 38(1):43–48, 2009.
9. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *ACM SIGMOD Record*, volume 25, pages 149–160. ACM, 1996.
10. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.
11. R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
12. C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 52–59. ACM, 2001.
13. P. Przymus and K. Kaczmarski. Dynamic compression strategy for time series database using gpu. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.
14. P. Przymus and K. Kaczmarski. Time series queries processing with gpu support. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.
15. A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.
16. O. O. Sonmez and A. Gursoy. Comparison of pricing policies for a computational grid market. In *Parallel Processing and Applied Mathematics*, pages 766–773. Springer, 2006.
17. M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):48–63, 1996.
18. T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *ACM SIGMOD Record*, 29(3):55–67, 2000.

Analysis of Multilayer Neural Networks with Direct and Cross-Forward Connection

Stanisław Płaczek and Bijaya Adhikari

Vistula University, Warsaw, Poland
stanislaw.placzek@wp.pl,bijaya.adhikari1991@gmail.com

Abstract. Artificial Neural Networks are of much interest for many practical reasons. As of today, they are widely implemented. Of many possible ANNs, the most widely used ANN is the back-propagation model with direct connection. In this model the input layer is fed with input data and each subsequent layers are fed with the output of preceeding layer. This model can be extended by feeding the input data to each layer. This article argues that this new model, named cross-forward connection, is optimal than the widely used Direct Conection.

1 Introduction

Artificial Neural Networks have broad implementation in Machine Learning, engineering and scientific applications. Their abilities to provide solutions to problems involving imprecisions and uncertainties with trivial implementation have enabled us to find solutions to real life problems as [1]:

1. Result approximation and data interpolation
2. Pattern recognition nad feature classification
3. Data compression
4. Trend prediciton
5. Error identification
6. Control

The problems mentioned above are solved by implementing ANN as universal approximator function with multidimensional variables. The function can be represented as:

$$Y = F(X) \tag{1}$$

where:

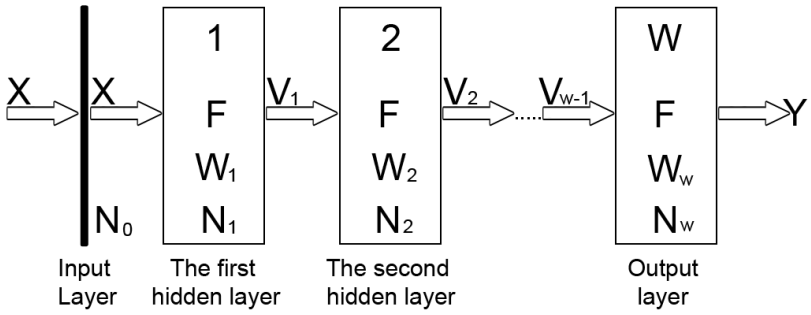
- X-input vector
- Y-output vector

Selecting a network to solve a specific problem is a tedious task. Decision regarding following thing must be made prior to attempting a solution.

- Structure of Neural Network, number of hidden layers and number of neurons in each layer. Conventionally, the size of input and output layers are defined by dimension of X and Y vectors respectively.
- Structure of individual neurons encompassing activation function, which takes requirement of learning algorithm into account.
- Data transfer methods between layers
- Optimization criteria and type of learning algorithm

Structure of Network can be defined in arbitrary way to accomplish complex tasks. The structure plays vital role in determining the functionality of ANN. This paper will compare and contrast two multilayer network structures.

- Direct Connection: This structure consists of at-least one hidden layer. Data tis fed from preceeding layer to succeeding one.



$N_0, N_1, N_2 \ldots N_W$-number of neuron in layer j=0,1...W.

X-input vector with dimension N_0

V_j- output vector of layer j=1,2,...,w-1 with dimension N_j

Y - output vector with dimension N_w

W_j- j=1...W- weight coefficient of matrix

Fig. 1. Structure of Direct Connection ANN

- Cross Forward Connection. In this structure, the input signal is passed on to every layer in the network. Therfore, a layer j=1,2,3.....W,where W is the output layer, has two inputs : vector X and Vector V_{j-1}, output of preceeding layer.

Structure of Cross Forward Connection is simpler than that of Direct Connection, in terms of neuron distribution in hidden layers. Learning time, as second parameter, is shorter for Cross Forward Connection . In later part of the paper,

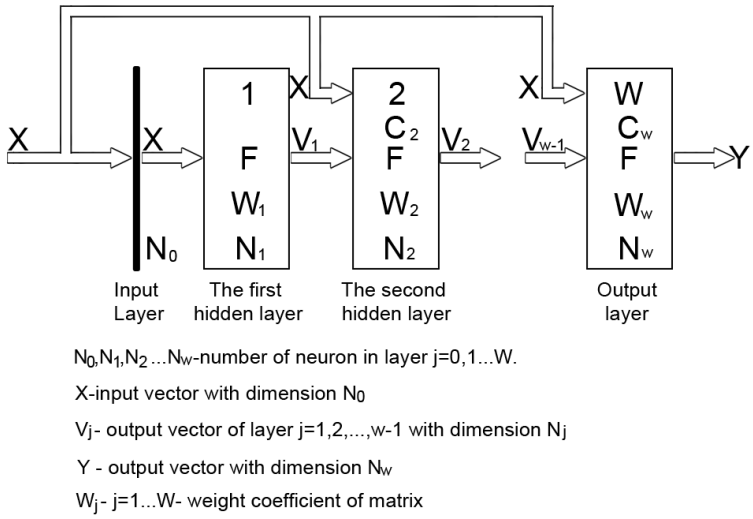


Fig. 2. Structure of Forward Connection ANN

we will analyze a particular optimization problem for ANN where total number of neurons, N, and number of layers , W, are given. Our target is to maximize the total number of subspaces which are created by neurons of every hidden layers. We will solve this complex problem with respect to the relation between dimensionality of feature space, N_0, and neurons' number in all hidden layers, N_i. This problem can be divided into two sub-problems.

- $N_i \leq N_0$ – liner optimization problem,
- $Ni > N0$ – non-linear optimization problem.

Where: i= 1,2,3,. . . . . W-1.

We can solve liner target function using liner-programming method. The non-linear task, with liner constrains, can be solved using Kuhn- Tucker conditions. As examples, we solved both sub-problems and discussed different ANN structures. In conclusion, we summarize our results giving recommendation for different ANN structures.

2 Criteria of ANN Structure Selection

The threshold function for the each neuron is defined as follows:

$$g(x) = \begin{cases} 1, & \text{if } x > 0 \\ -1, & \text{if } x \leq 0 \end{cases} \tag{2}$$

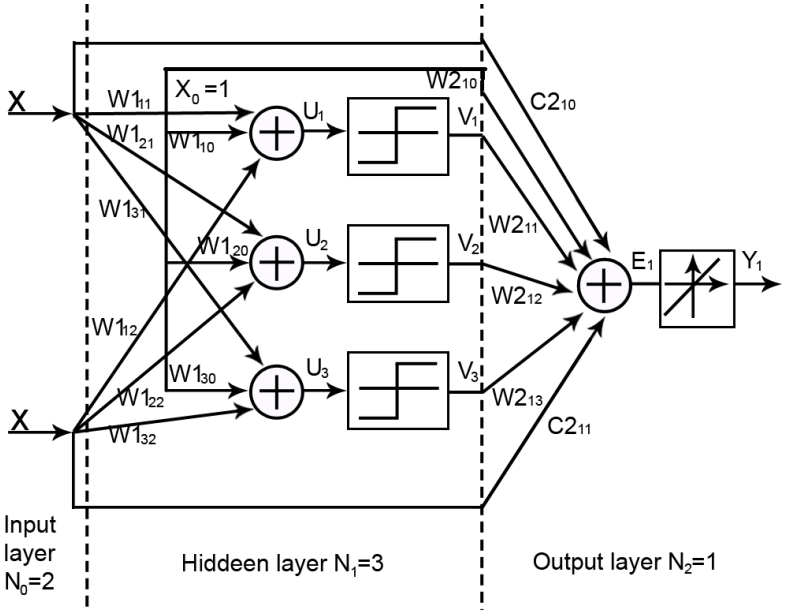


Fig. 3. Two Layer ANN with Cross Forward Connection

We say that the network in Fig. 3 has structure 2-3-1. Where:

- N_0=2; number of neurons in input layer.
- N_1=3; number of neurons in hidden layer.
- N_2=1; number of neurons in output layer.

Signal transfer from input layer to output layer in this structure can be represented in the following way.

$$U = W_1 \cdot X \tag{3}$$

$$V = F_1(U) \tag{4}$$

$$E = W_2 \cdot V + C_2 \cdot X \tag{5}$$

$$Y = F_2(E) \tag{6}$$

Where,

- $X[0 : N_0]$ -input signal
- $W_1[1{:}N_1;0{:}N_0]$ - weight coefficients matrix of hidden layer
- $U[1{:}N_1]$-analog signal of hidden layer
- $V[1{:}N_1]$-output signal of hidden layer

- $W_2[1{:}N_2;0{:}N_1]$ - weight coefficients matrix of output layer
- $E[1{:}N_2]$-analog signal of output layer
- $Y[1{:}N_2]$-output signal of output layer
- $C_2[1:N_2;0:N_0]$ -weight coefficients matrix of Cross connection

This network will be used for pattern recoginition after being trained by teacher datas.

The architecure of ANN in fig(3) could be represented using hyper-spaces. Lets imagine a hyperspace having dimension of the number of neurons in the input layer. The first hidden layer, depicted in equation (3) and (4), divides feature space, X, into subspaces.

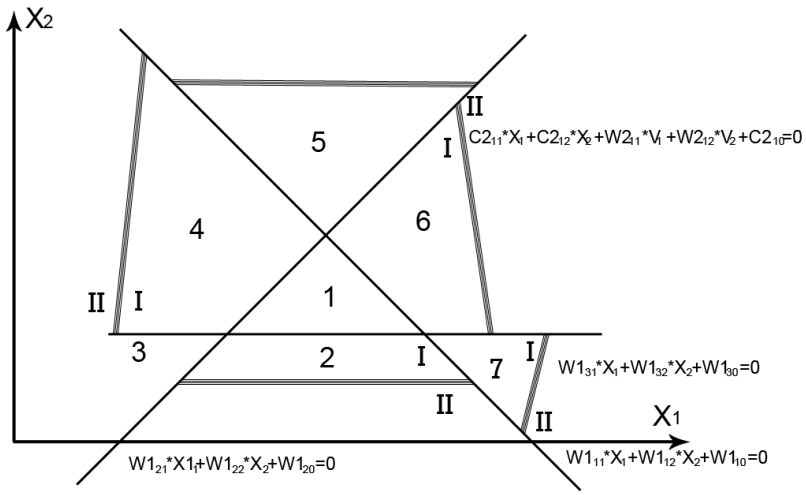


Fig. 4. Structure of division of two dimensional input space by three neurons of the first hidden layer.

Two dimensional feature space is divided into seven sub-spaces. These sub-spaces correspond to internal structure of input data.

The function Φ (p,q) gives the maximum number of p dimensional sub-spaces formed q number of $p-1$ dimensional hyper-planes. The function has following recursive form.[3]

$$\Phi(p,q) = \Phi(p-1,q) + \Phi(p-1,q-1) \tag{7}$$

By definition of $\phi(p,q)$, it is clear that

$$\Phi(p,1) = 2 \tag{8}$$

and

$$\Phi(1, q) = q + 1 \tag{9}$$

In context of Neural Networks, q – number of neurons in the first hidden layer,N_i, and p – dimension of input vector, N_0.

Table 1. Number of sub spaces formed by division of p dimensional input Vector by q neurons present in the first hidden layer

q \ p	1	2	3	4	5	6	7	8	9	10
1	2	2	2	2	2	2	2	2	2	2
2	3	4	4	4	4	4	4	4	4	4
3	4	7	8	8	8	8	8	8	8	8
4	5	11	15	16	16	16	16	16	16	16
5	6	16	26	31	32	32	32	32	32	32
6	7	22	42	57	63	64	64	64	64	64
7	8	29	64	99	120	127	128	128	128	128
8	9	37	93	163	219	247	255	256	256	256
9	10	46	130	256	382	466	502	511	512	512
10	11	56	176	386	638	848	968	1013	1023	1024

Now, re-writing (7), we get:

$$\Phi(p, q) = \Phi(1, q) + \sum_{k=1}^{p-1} \Phi(k, q - 1) \tag{10}$$

Solving recursion (10), we get :

$$\Phi(p, q) = C_{q-1}^{p} + 2 \sum_{k=0}^{p-1} \cdot C_{q-1}^{k} \tag{11}$$

where,

$$C_n^k = \frac{n!}{k! \cdot (n - k)!} \tag{12}$$

In the equations above:

- p-dimension of input vector.
- q- number of neurons in hidden layer

Lets consider an example, for a network having three neurons in first hidden layer and input vector of dimension 2. From (11), We get Φ(2,3)=7.

The number of subspaces formed due to division of the neurons in input layer by the the neurons in the first hidden layer depends solely on the number of neurons. The table presented above shows number of subspaces for different values of p and q.

Coming back to the structure of Cross-Forward Connection, according to Fig.3, input signals to the second hidden layer can be divided into two subsets:

– input received from the output of previous layer-Vector V
– raw input received - vector X

All input signals are multiplied by the adjustable weights of associated neurons i.e. matrices W_2 and C_2 respectively.

For ANN presented in fig.3, we can write:

$$e_k = \sum_{i=1}^{N_1} W_{2k,i} \cdot V_i + \sum_{j=0}^{N_0} C_{2k,j} \cdot X_j \tag{13}$$

And, finally,

For e_k=0,

$$\sum_{j=0}^{N_0} C_{2k,j} \cdot X_j = -\sum_{i=1}^{N_1} W_{2k,i} \cdot V_i \tag{14}$$

The input space, X, in (14) represents the set of parallel hyper-planes. The number of hyper-planes depend on V_i. For two dimension space, the second layer of ANN is composed of four parallel lines formed by all possible combination of values of V_i and V_j i.e.,0,0; 0,1; 1,0; 1,1.

Every subspace which is formed by the hidden layer is further divided into two smaller sub-spaces by output neuron. For N_0, dimensional input space and N_1 number of neurons in the first hidden layer, the maximum number of subspaces is given by:

$$\Psi(N_0, 2) = \Phi(N_0, N_1) \cdot \Phi(N_0, N_2) \tag{15}$$

For, W>2 ,number of sub-spcaes is:

$$\Psi(N_0, W) = \prod_{i=1}^{W} \Phi(N_0, N_i) \tag{16}$$

The number of subspaces of initial feature space in fig 3 is:

$$\Psi_{2,2} = \Phi(2, 3) \cdot \Phi(2, 1) = 7 * 2 = 14$$

For example, to divide input space into 14 subspaces, we require 3 neurons in the first hidden layer and 1 in output layer. Whereas, we need 5 neurons in the first hidden layer and 1 neuron in output layer to obtain the same number of subspaces in the standard Direct Connection. It could be concluded that the ANN with cross forward connection is more optimal than the regular straight Forward Fonnection.

3 Learning Algorithm for Cross Forward Connection Network

Less number of neurons helps convergence of algorithm during learning process. We use standard back propagation algorithm. Aim function(goal of learning

process) is defined as

$$e^2 = \frac{1}{2} \cdot \sum_{k=1}^{N_w} (y_i - z_i)^2 \tag{17}$$

where, z_i is the value provided by the teacher and y_i is the output computed by the network.

And new value of weight coefficient is:

$$W_{ij}(n+1) = W_{ij}(n) - \alpha \cdot \frac{\partial e^2}{\partial W_{ij}}\Big|_n + \beta[W_{ij}(n) - W_{ij}(n-1)] \tag{18}$$

and

$$C_{ij}(n+1) = C_{ij}(n) - \alpha \cdot \frac{\partial e^2}{\partial C_{ij}}\Big|_n + \beta[C_{ij}(n) - C_{ij}(n-1)] \tag{19}$$

4 Structure Optimization of Cross Forward Connection Network

ANN structure optimization is very complicated task and can be solved in different ways. Experience has taught us that ANN with 1 or 2 hidden layer is able to solve most of the practical problems. The problem of ANN optimization structure can be described as :

– maximizing number of subspaces, $\Psi(N_0, W)$.

when total number of neurons,N, and number number of layers, W, are given.

4.1 Optimization task for ANN with one hidden layer

For ANN with 1 hidden layer, the input neurons' number,N_0,is defined by the input vector structure X and is known as apriori. The output neurons' number N_2 is given by the output vector structure, Y - known as task definition. We can calculate the neurons' numbers in the hidden layer N_1 using equation 16. According to the optimization criterion and formula 16, the total number of subspaces for ANN with one hidden layer is given by:

$$\Phi(N_0, W) = \Phi(N_0, 2) = \Phi(N_0, N_1) \cdot \Phi(N_0, N_2) \tag{20}$$

Finally we can calculate number of neurons in one hidden layer N_1.

4.2 Optimization task for more than one hidden layer

For ANN with 2 or more hidden layers, optimization is more complicated. As the first criterion, we assume that:

– the number of layers W is given and,
– total number of neurons N is given for all hidden layers.

N can be calculated using:

$$N = \sum_{i=1}^{W-1} N_i = N_1 + N_2 + N_3 + ... + N_{W-1} \tag{21}$$

In practice we have to calculate neuron's distribution between $\{1 : W - 1\}$ layers. To find neuron's distribution, we have to maximize the number of subspaces according to the equation 22 with 23 as constraint.

$$\psi(N_0, W-1)^{opt} = \max_{N_1, N_2 \ldots N_{W-1}} \prod_{i=1}^{w-1} \Phi_i(N_0, N_i) \tag{22}$$

$$N = \sum_{i=1}^{W-1} N_i = N_1 + N_2 + N_3 + ... + N_{W-1} \tag{23}$$

From 11 and 22,

$$\begin{aligned} \Phi(N_0, N_i) = C_{N_i-1}^{N_0} + 2 \sum_{k=0}^{N_0-1} \cdot C_{N_i-1}^{k} \\ for\ i\ \epsilon\ [1; W-1] \end{aligned} \tag{24}$$

Please note that:

$$\begin{aligned} C_{N_i-1}^{N_0} = 0 \\ when\ N_i - 1 - N_0 < 0 \\ N_i \leq N_0 \end{aligned} \tag{25}$$

Taking 22, 23, 24, and 25 into account, our optimization task can be written as:

$$\Psi(N_0, W-1)^{opt} = \max_{N_1, N_2 \ldots N_{W-1}} \left\{ \prod_{i=1}^{W-1} [C_{N_1-1}^{N_0} + 2 \sum_{k=0}^{N_0-1} C_{N_i-1}^{k}] \right\} \tag{26}$$

with constraints

$$N = \sum_{i=1}^{W-1} N_i \tag{27}$$

$$C_{N_i-1}^{N_0} = 0 \ \ for \ \ N_i \leq N_0 \tag{28}$$

$$C_{N_i-1}^{k} = 0 \ \ for \ \ N_i \leq k \tag{29}$$

The optimization problem in 26 is non-linear and solution space can be divided into :

1. For all hidden layers $N_i \leq N_0$ and $N_i \leq k$ — linear task
2. For all hidden layers $N_i > N_0$ and $N_i > k$ — non-linear task

Set of hidden layers can be divided into two subspaces:

- $S1 = \{N_1, N_2, N_3, ..., N_j\}$ where $j \leq W-1$.For S_1, $N \leq N_0$ and $Ni \leq K$
- $S2 = \{N_{j+1}, N_{j+2}, N_{j+3}, ..., N_{W-1}\}$.For S_1, $N_i > N_0$ and $Ni > K$

Where W = number of layers and W-1 = number of hidden layers. This is a mixed structure, for which final solution can be found using mixture of both methods from point 1 and 2.

4.3 Neuron distribution in the hidden layers, where neurons' number for all hidden layers is less or equal than initial feature space

In this case, we have

$$N_i \leq N_0 \; for \; i \; \epsilon \{ \; 1; W-1\} \tag{30}$$

So, the total number of subspaces is defined by

$$\Phi(N_0, N_i) = \frac{(N_i - 1)!}{N_0!(N_i - 1 - N_0)!} + 2 \cdot \sum_{k=0}^{N_0 - 1} \frac{(N_i - 1)!}{k!(N_i - 1 - k)!} \tag{31}$$

or,

$$\Phi(N_0, N_i) = 0 + 2 \cdot 2^{N_i - 1} = 2^{N_i} \tag{32}$$

Our optimization target can be written as,

$$\begin{gathered} \Psi(N_0, W-1)^{opt} = \max_{N_i \; \epsilon \; [1, W-1]} \left\{ \prod_{i=1}^{W-1} 2^{N_i} \right\} = \max_{N_i \; \epsilon \; [1, W-1]} \left\{ \; 2^{\sum_{i=1}^{W-1} N_i} \; \right\} \\ for \; N = \sum_{i=1}^{W-1} N_i \\ N_i \leq N_0 \; and \; N_i, N_0 \geq 0 \end{gathered} \tag{33}$$

Equation 33 is monotonously increasing and can be written as

$$\begin{gathered} \Psi(N_0, W-1)^{opt} = \max_{N_i \; \epsilon \; [1, W-1]} \left\{ \sum_{i=1}^{W-1} N_i \right\} \\ For \; N = \sum_{i=1}^{W-1} N_i \\ N_i \leq N_0 \; and \; N_i, N_0 \geq 0 \end{gathered} \tag{34}$$

Under the given number of layers, total number of neurons have to satisfy the new constraints

$$N_i \leq N_0 \; and \; N \leq (W-1)N_0 \tag{35}$$

Example:
For ANN with $N_0 = 3, N_1 \leq 3, N_2 \leq 3, N_3 = 1$, $W = 3$, find optimum neurons distribution between two hidden layers N_1, N_2.

It is known that for output layer $N_3 = 1$ and therefore we will only consider two hidden layer for optimization process. For all N_i, where $i = 1, 2$ and $N_i \leq N_0$, using 35 we can write:

$$\begin{aligned} &N \leq (W-1) \cdot N_0 = (3-1) \cdot 3 = 6 \\ &\text{Taking } N_0 = 3 \text{ using 34 we achieve} \\ &\Psi(N_0, W-1) = \Psi(3,2) = max\{N_1 + N_2\} \\ &\text{and constraints} \\ &N_1 \leq 3 \\ &N_2 \leq 3 \\ &we \; use \; N_1 + N_2 = 4 < 6 \end{aligned} \tag{36}$$

To solve this optimization task, we can use linear programming methods or use figure 5.

Using only discrete values of N_1, N_2 for N=4, we can find three solutins $(N_1, N_2) = \{(1,3), (2,2), (3,1)\}$

The following equations indicate the number of subspaces for different number of neurons.

$$\begin{aligned} \Phi(N_0, N_1) &= \Phi(3,1) = 2^1 = 2 \\ \Phi(N_0, N_1) &= \Phi(3,2) = 2^2 = 4 \\ \Phi(N_0, N_1) &= \Phi(3,3) = 2^3 = 8 \end{aligned} \tag{37}$$

Finally, we have three optimal solutions with three different ANN structure. Every structure generates 16 subspaces and are euqivalent. Table 2.

Table 2. Solution of linear programming for N=4

N_0	N_1	N_2	$\Phi(N_0, N_1)$	$\Phi(N_0, N_2)$	$\Psi(N_0, W-1)$
3	1	3	2	8	16
3	2	2	4	4	16
3	3	1	8	2	16

In conclusion, we can say that for every given total number of neurons,N, we have many possible neurons distribution between layers. Optimal number of subspaces in the initial feature space has the same value, Ψ.

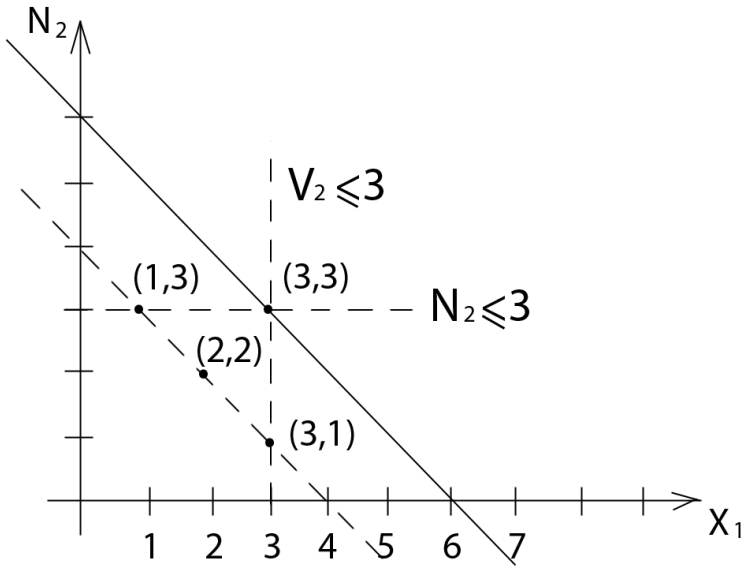


Fig. 5. Graphical solution of linear programming when total number of neurons, N=6 and N=4

4.4 Neurons distribution in the hidden layers, where neurons' number for all hidden layers is greater than initial feature space

Lets assume number of layers, W=3. It implies that we have only two hidden layers. According formula 24.

$$\begin{aligned} \Phi(N_0, N_i) =& C_{N_i-1}^{N_0} + 2\sum_{k=0}^{N_0-1} C_{N_i-1}^{k} \\ & \text{for i } \epsilon\ [1 : W-1] \ and\ N_i > N_0 \\ & \text{For whole ANN, total number of subspaces is given by} \\ \Psi(N_0, W-1) =& \Psi(N_0, 2) = \Phi_1(N_0, N_1) \cdot \Phi_2(N_0, N_2) \\ & \text{and } N_1 + N_2 = N \\ & \text{so, } \ N_1 + N_2 > 2N_0 \end{aligned} \tag{38}$$

Taking all assumptions into account we can write,

$$\begin{aligned} &\Phi(N_0, N_1) = C_{N_i-1}^{N_0} + 2 \cdot (C_{N_i-1}^{0} + C_{N_i-1}^{1} + ... + C_{N_i-1}^{N_0-1})\ for\ N_0 < N_i \\ &\Phi(N_0, N_1) < C_{N_i-1}^{N_0} + 2 \cdot 2^{N_i-1} < 2^{N_i} \end{aligned} \tag{39}$$

In this situation we do not know how many suspaces there are for $\Phi(N_0, N_1)$. To find neurons distribution between the hidden layers we should know relations between N_0, N_i and N.

Example:
For N_0=3, W=3 N=8, and N=10, N=12 find neuron distribution in the layers, were $N_i > 3$. We should maximize the quality criterion

$$\Psi(N_0, W-1)^{OPT} = \max_{N_1, N_2 \ldots N_{W-1}} \prod_{i=1}^{W_1} \left[C_{N_i-1}^{N_0} + 2 \cdot \sum_{k=0}^{N_0-1} C_{N_i-1}^{k} \right] \tag{40}$$

For example,

$$\Psi(3,2)^{OPT} = \max_{N_1, N_2} \prod_{i=1}^{2} \left[C_{N_i-1}^{3} + 2 \cdot \sum_{k=0}^{2} C_{N_i-1}^{k} \right] \tag{41}$$

After simple algebraic operations, we achieve

$$\begin{aligned} &\Psi(3,2)^{OPT} = \max_{N_1, N_2} \left\{ \frac{N_1^3 + 5N_1 + 6}{6} \cdot \frac{N_2^3 + 5N_2 + 6}{6} \right\} \\ &N_1 > 3 \\ &N_2 > 3 \\ &N_1 + N_2 = 8 > 6 \end{aligned} \tag{42}$$

We solve the equation using Kuhn-Tucker conditions. Taking 42 into account. we can write the following Lagrange equation

Table 3. Solution for non-linear Kuhn Tucker conditions for total number of neurons, N=8–12

N	$N_1 > 3$	$N_2 > 3$	$\Phi(3,21)$	*Solution*
8	4	4	225	max
9	5	4	390	max
	4	5	390	max
10	6	4	630	
	5	5	676	max
	4	6	630	
11	4	7	960	
	5	6	1092	max
	6	5	1092	max
	7	4	960	
12	4	8	1395	
	5	7	1664	
	6	6	1774	max
	7	5	1664	
	8	4	1395	

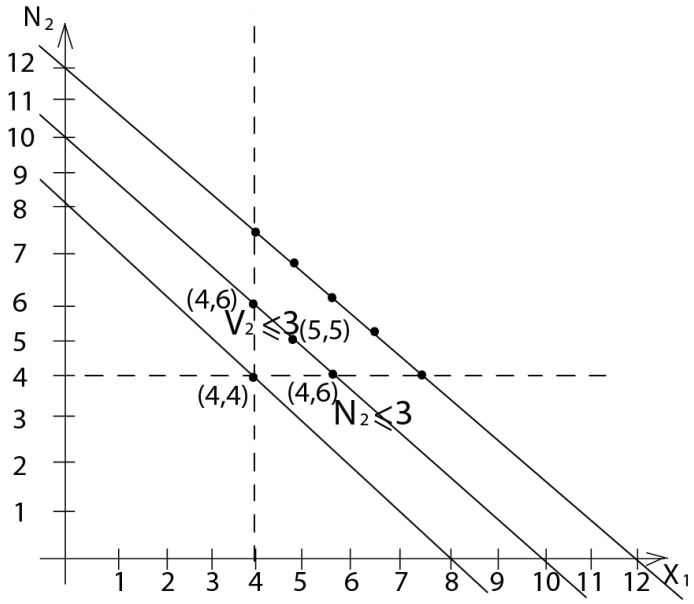


Fig. 6. Graphical solution of Kuhn Tucker conditions. Line $N = N_1 + N_2$ is a solving line with one or more solutions. Only one point is max. Figure shows three solution lines for $N_1 + N_2 = 8$, $N_1 + N_2 = 10$, $N_1 + N_2 = 12$

$$L = \frac{N_1^3 + 5N_1 + 6}{6} \cdot \frac{N_2^3 + 5N_2 + 6}{6} - \lambda_1 \cdot (N_1 - 4) - \lambda_2 \cdot (N_2 - 4) - \lambda_3 \cdot (N_1 + N_2 - 8)$$
$$N_1 - 4 \geq 0$$
$$N_2 - 4 \geq 0$$
$$N_1 + N_2 - 8 = 0 \tag{43}$$

5 Conclusion

For most practical purposes, ANNs with one hidden layer are sufficient. Learning Algorithms for the networks are time consuming and depend on number of layers and number of neurons in each layer. The running time of learning algorithm has dependency, greater than linear, on the number of neurons. Hence, the running time increases faster than the total number of neurons.

Cross Forward connection provides us an opportunity to decrease the number of neurons and thus, the running time of learning algorithm.

We implemented both Direct Connection Neural Networks and Cross Forward Neural Networks with one hidden layer and used them for pattern recognition.

Our implementation required three input neurons and two output neurons. We varied the number of neurons in hidden layer and trained both networks for limited number of epoches and noted the sum of squared errors of each output neurons. The procedure was repeated 20 times and the average sum of square of errors were recorded. Datas for two cases are presented in table 4 and 5.

Table 4. Comparision for Direct Connection and Cross Forward Connection with $N_0 = 3$, $N_1 = 1$,$N_W = 2$

Epoches	10	50	100	500	1000	5000	10000	50000
$\sum \epsilon^2$ for Direct Connection	12.40415	9.10857	8.58351	8.48001	8.38696	8.260625	8.14166	8.0152
$\sum \epsilon^2$ for Cross Forward	2.22719	0.33131	0.12325	0.02912	0.00808	0.00148	0.00076	0.00014

Table 5. Comparision for Direct Connection and Cross Forward Connection with $N_0 = 3$, $N_1 = 4$,$N_W = 2$

Epoches	10	50	100	500	1000	5000	10000	50000
$\sum \epsilon^2$ for Direct Connection	6.91134	0.28018	0.11306	0.01864	0.00542	0.000092	0.000052	0.00009
$\sum \epsilon^2$ for Cross Forward	1.02033	0.12252	0.064224	0.01945	0.00441	0.000823	0.000381	0.00007

Table 4 and 5 clearly demonstrate that for the given number of neurons in the hidden layer, Cross-Forward Connection are optimal. If we closely examine the error term in table four for Direct Connection and the same in table 5 for Cross Forward Connection we will notice that they are fairly comparable. It demonstrates that Cross Forward Connecton Structure with one neuron neuron in hidden layer is almost as good as Direct Connection with four neurons in hidden layer. Thus, Cross-Forward connection reduce the required number of neurons in ANNs.

In addition using optimizations criterion for Cross Forward Connection structures, we have solved two different tasks. For linear one , where $N_i \leq N_0$ for i=1,2,... W-1, we e achieved an equivalent ANN structures with the same number of total subspaces $\Psi(N_0, W-1)$. This means that for given total number of neurons ,N, and number of layers W , there are multiple equivalent ANN structures (Table 2). In practice this ANN structures can be used for tasks with very big dimensionality of input vector X (initial feature space). For nonlinear optimization task, where $N_i > N_0$ for i=1,2,3..... W-1, the target function is nonlinear with liner constraints. There could be one or more optimum solutions. Final solution depends on dimensionality of feature space N_0 and relation between N, N_i and W. In our example, for ANN with $N_0 = 3$, W=3, and

N=8,9,10,11,12,..... we achieved one optimum solution for even N_0s and two solutions for odd N_0s (Table 3).

References

1. Stanisaw Osowski, Sieci Neuronowe do Przetwarzania Informacji. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006.
2. S. Osowski, Sieci neuronowe w ujeciu algorytmicznym.WNT, Warszawa 1996.
3. O.B.Lapunow, On Possibility of Circuit Synthesis of Diverse Elements, Mathematical Institut of B.A. Steklova, 1958.
4. Toshinori Munakate, Fundationals of the New Artificial Intelligence. Second Edition, Springer 2008.
5. Colin Fyle, Artificial Neural networks and Information Theory, Departmeeent of Ciomputing and information Systems, The University of Paisley, 2000.
6. Joarder Kamruzzaman, Rezaul Begg, Artificial Neural Networks in Finance and Manufacturing, Idea Group Publishing, 2006.
7. A. Mariciak, J. Korbicz, J. Kus, Wstepne przetwarzanie danych, Sieci Nuronowe tom 6, Akademicka Oficyna Wydawnicza EXIT 2000.
8. A. Marciniak, J. Korbicz, Neuronowe sieci modularne, Sieci Nuronowe tom 6, Akademicka Oficyna Wydawnicza EXIT 2000.
9. Z. Mikrut, R. Tadeusiewicz, Sieci neuronowe w przetwarzaniu i rozpoznawaniu obrazow, Sieci Nuronowe tom 6, Akademicka Oficyna Wydawnicza EXIT 2000.
10. L. Rutkowski, Metody i techniki sztucznej inteligencji, Wydawnictwo Naukowe PWN, warszawa 2006.
11. Juan R. Rabunal, Julian Dorado, Artificial Neural Networks in Real-Life Applications, Idea Group Publishing 2006.

Fractional Genetic Programming for a More Gradual Evolution*

Artur Rataj

Institute of Theoretical and Applied Computer Science,
Bałtycka 5, Gliwice, Poland
arataj@iitis.gliwice.pl

Abstract. We propose a softening of a genetic program by the so–called fractional instructions. Thanks to their adjustable strengths, a new instruction can be gradually introduced to a program, and the other instructions may gradually adapt to the new member. In this way, a transformation of one candidate into another can be continuous. Such an approach makes it possible to take advantage of properties of real–coded genetic algorithms, but in the realm of genetic programming. We show, that the approach can be successfully applied to a precise generalisation of functions, including those exhibiting periodicity.

Keywords: genetic programming, real–coded genetic algorithm, evolutionary method, gradualism

1 Introduction

One of the basic concepts in genetics is quantitative inheritance, i.e. a gradual regulation of the strength of a single genetic trait. Such an inheritance is realised by e.g. encoding a single trait with several genes, which collaboratively add up to strengthen the trait.

A quantitative inheritance enables an easy way for an effective *exploitative* searching of the candidate space – an offspring, which is similar to its parents, has a high chance of being viable in similar environmental conditions, and thus to be a next step in walking the candidate space. The similarity might make it more difficult to find a substantially different environment by the offspring, yet a need to live in a radically different environment is often not the case. Consider e.g. the formation of limbs in animals – they evolved from fins in the process of a long, exploitative transition, that employed slight changes in expressions of genes and a gradual appearance of new genes. Actually, it has been recently shown, that just varying of the expression of certain genes in fish makes their fins more limb–like [5]. Such a graduality enabled a progressive change of the environment from sea to land, with children being able to live in an environment similar to that of their parents, though.

* This work has been supported under the project *An active creation of a model of space using a 3d scanner and an autonomous robot, N N516 440738*

The concept of graduality or continuity of the candidate space is the basis of real–coded genetic algorithms [6, 14], supporting a number of dedicated genetic operators [7].

We propose genetic programs in a form that is halfway between a tree representation [3] and a linear list [11]—a linear sequence of functions (instructions) is communicating using a stack, as introduced by [12], The basic difference of our approach, though, lies in the graduality of the stack operations, constructed to support the discussed continuity of instructions. The continuity is similar to that of [13]. There are two basic differences, though:

- No need for special continuous equivalents of common operations like addition or multiplication. Instead, the original functions can be used directly. The graduality is provided elsewhere—by a specially constructed stack.
- A program does not strictly follow a tree structure, as a value of a node can be reused by a number of other nodes, which promotes reusability of an instruction result and in effect reduces redundancy. Like in the case of instructions, the dependencies out of the tree hierarchy can also form gradually.

The paper is constructed as follows: Sect. 2 describes fractional stack operations. On the basis of these definitions, fractional operations are discussed in Sect. 3. These build a fractional program, described in Sect. 4. A method of evolving such programs is introduced in Sect. 5. Section 6 presents some tests. Finally, there is a discussion in Sect. 7.

2 A Fractional Stack

Let the stack, in order to be compatible with the fractional instructions, have a continuous length, so that it can support a fractional pushing and popping. Thanks to this, a weak operation, i.e. having a low strength, may interact with the stack so that there is only a minimal change to the stack state.

Let each element on the stack have a real value x and also a real length. The length is specified by the strength s of a push operation $\mathrm{push}(s, x)$, and contributes to the total length of the stack, which is a different value than the total number of elements in that stack, i.e. the stack size. A pop operation, in turn, $y = \mathrm{pop}(s)$, having the strength s, shortens the stack length by s, by removing 0 or more top elements from the stack, and also possibly shortening the length of the element f, which is on the top after that removal. The value popped y is a mean of values of the elements removed or shortened, weighted by the respective lengths of each element removed, and also by the amount of shortening of the length of f.

Let us consider an example. Two operations $\mathrm{push}(0.6, 10)$ and $\mathrm{push}(1.2, 20)$ led to a stack whose top fragment is shown in Fig. 1(a). Then, a $\mathrm{pop}(1.5)$ operation, in order to shorten the stack by a proper value, needed to remove the top element and shorten the neighbouring element by 0.3, as seen in Fig. 1(b). The

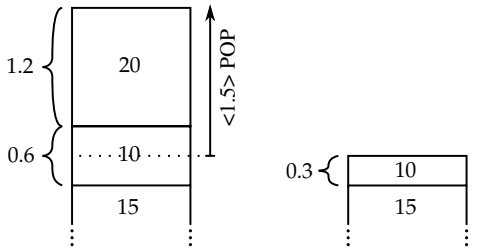


Fig. 1. An example of a fractional pop(s): (a) before, and (b) after the operation.

value returned by that operation would be a weighted average

$$\frac{1.2 \cdot 20 + 0.3 \cdot 10}{1.2 + 0.3} = 18.$$

Let i be an index of an element in the stack, 0 for the bottommost element, $S - 1$ for the topmost element, where S is the total number of elements in the stack. Then let L_i be the length of the ith element in the stack.

3 Fractional Instructions

Given the stack operations, definitions of the fractional instructions are trivial. Possible arguments of such an instruction are always popped from the stack, and a possible result of the instruction is pushed back to the stack. A length of each of the stack operations is given by the instruction *strength*. A low–strength instruction can only minimally modify the stack contents, fulfilling in this way the graduality criterion. In particular, a zero–strength instruction does nothing.

Let the set of instructions be limited to several stack handling operations, and also a handful of arithmetic operations. Each instruction's mnemonic is preceded by the instruction's strength. Table 3 lists the definitions of all instructions. Arguments of binary operators are evaluated from left to right. As seen, $\langle s \rangle$ POP does not use the popped value, the whole role of the instruction is to modify the stack state. We introduce $\langle s \rangle$ COPY $n{:}x$, that multiplies the length of n elements at the stack top by x. The instruction is modelled after the instruction DUP – an equivalent of COPY 1:2, used to simplify programs implemented in stack–based languages, like FORTH, FIFTH [8] or Java bytecode [10]. In our case of fractional instructions, COPY can act as an adapter of n argument passed between operations of different strengths.

4 A Fractional Program

Before the program is executed, its N_i input values X_i, $i = 0, 1, \ldots N_i - 1$, are pushed to the stack with a sequence of operations $\langle 1 + p \rangle$ PUSH X_i. The fixed

Table 1. A list of basic instructions.

Mnemonic	Stack operations
$\langle s \rangle$ PUSH x	$\mathrm{push}(s, x)$
$\langle s \rangle$ POP	$\mathrm{pop}(s)$
$\langle s \rangle$ COPY $n{:}x$	$l_0 = L_{S-1}, v_0 = \mathrm{pop}(l_0)$, $l_1 = L_{S-1}, v_1 = \mathrm{pop}(l_1)$, $\vdots$ $l_{n-1} = L_{S-1}, v_{n-1} = \mathrm{pop}(l_{n-1})$, $\mathrm{push}(xl_{n-1}, v_{n-1})$, $\mathrm{push}(xl_{n-2}, v_{n-2})$, $\vdots$ $\mathrm{push}(xl_0, v_0)$
$\langle s \rangle$ ADD	$\mathrm{push}\Big(s, \mathrm{pop}(s) + \mathrm{pop}(s)\Big)$
$\langle s \rangle$ MUL	$\mathrm{push}\Big(s, \mathrm{pop}(s)\mathrm{pop}(s)\Big)$
$\langle s \rangle$ NEG	$\mathrm{push}\Big(s, -\mathrm{pop}(s)\Big)$
$\langle s \rangle$ INV	$\mathrm{push}\Big(s, \mathrm{pop}(s)^{-1}\Big)$
$\langle s \rangle$ SIN	$\mathrm{push}\Big(s, \sin\Big(\mathrm{pop}(s)\Big)\Big)$

value $p \geq 0$ is a padding, that decreases the probability of a stack underflow – the program may instead have a chance of popping the input value or its part again. This is the only case of an instruction, that has a strength greater than 1, but such an instruction never occurs in a program itself. We will use a padding of 1 in tests.

If it is unlikely that a program needs a very long stack to solve a given problem, the stack size might have some constant limit imposed, so that stack overflows become possible. This way, such programs, as likely too complex, are deemed invalid, which favours simpler solutions and also saves time by preventing a further exploitation of such programs.

A program, after finishing, yields a single output value Y, which is equal to whatever is left in the stack, but excluding its possible bottommost part, if any, never used by the program, as that part contains unprocessed input values. Thus,

$$Y = \mathrm{pop}\left(\left(\sum_{k=0}^{S-1} L_k\right) - \min\left(L_u, N_i\right)\right) \tag{1}$$

where L_u is the global minimum length of the stack across all $\mathrm{pop}(s)$ instructions executed within the program, or ∞ if there were no such instructions.

See that if the first instruction of a program is $\langle 1 \rangle$ COPY $N_i{:}z$, and then all of the following instructions are of a strength z, then for given X_i and any $z > 0$ the program yields exactly the same Y. This shows, that an instruction retains its characteristics for any strength, beside a different level of interaction with the

stack. What is important, though, is the ratio between strengths of instructions that communicate using the same stack, and not the absolute values of these strengths.

5 Evolutionary Method

We have developed an evolutionary method that dynamically adjusts the exploration/exploitation scheme like it is often practised in genetic programming [1, 4, 9]. The exploitation employs a novel technique that emphasises the advantage of gradual instructions. To keep this paper focused, we do not go into the complexities of maintaining a population of candidates.

5.1 Exploration – a new candidate

A candidate program has a fixed length P_n and a fixed maximum stack size P_s, both roughly accommodated to the function to fit to, so that too complex solutions are impossible.

There are several types of instructions to randomly pick. A priority is given to $\langle s \rangle$ PUSH x, as otherwise most candidates would turn out to be invalid due to stack underflows. We have chosen in tests that this type will be chosen with $P_{\text{PUSH}} = 0.3$, while for the 7 other types $P_{\neg\text{PUSH}} = 0.1$. The pushed value will be drawn using a uniform distribution, spanning reasonable limits $\langle -10, 10 \rangle$. Arguments of $\langle s \rangle$ COPY $n{:}x$ will be chosen using an uniform distribution as well. The ranges are $n \in \langle 1, 2 \rangle$ as it accommodates a typical number of values popped by an instruction; $x \in \langle 0, 2 \rangle$ not getting too large, as sequences of $\langle s \rangle$ COPY $n{:}x$ may effectively enlarge the upper limit of multiplying lengths of stack elements.

5.2 Exploitation – tuning of a candidate

A gradual exploitation shows the flexibility of the fractional programs. An instruction may have its strength gradually reduced even until it disappears, or another instruction may appear with its strength near 0. Both such a deletion and such an insertion may have an arbitrarily small influence on the program, as very small strengths translate to only a minimal interaction with the stack.

The exploitation is a random walk in the candidate space, beginning with a new candidate created by exploration. A backtrack to the previous position occurs, if the mean square error (MSE) of the new candidate generalised of fitting to a function $g(x)$ is not better than the respective MSE of the previous candidate, or if the new candidate is invalid because of a stack underflow or overflow, or because of an arithmetic error like a division by zero. Let a single step be called a modification Δ.

We are unsure which Δ is the best one for a given problem. Thus, we choose to introduce a variability of its parameters – they are picked using probability distributions before each step. Let these parameters be as follows – a minimum threshold Θ_{min} and an instruction perturbance level Φ, both chosen from $\langle 0, 1 \rangle$

using an uniform distribution. $\Theta_{\min}$ controls how many instructions are modified on average within Δ. The idea is, that it would sometimes be enough to modify only a single instruction within Δ, and it might even be advantageous to do so, if exactly a single instruction needs to be modified, in order to get a better candidate; parallel modifications of other instructions might create an unwanted drift in the candidate space in such a case. Yet, sometimes more instructions need to be modified at once, in order to omit the Hamming wall [2]. Also, a computation time is obviously crucial in genetic programming, and the random walk might be faster at times if more instructions are modified per Δ. We allow each case by $\Theta_{\min}$ being variable. Φ stems from a similar reasoning – it translates to how much an instruction is modified on average within Δ. Sometimes a fast walk is needed when e.g. a candidate is far from the optimum point in the candidate space. Yet, sometimes small, precise steps are needed instead, when e.g. the exploited candidate oscillates very closely around the optimum point.

Let within a single Δ, each of the instructions I be modified as follows:

1. Pick $\Theta \in \langle 0, 1\rangle$ using an uniform distribution. If $\Theta < \Theta_{\min}$, then do not modify I. Go to 2 only otherwise.
2. Let a strength perturbance $\Phi_s = \delta_s \Phi$ and a value perturbance $\Phi_v = \delta_v \Phi$ control, respectively, how much an instruction strength and an instruction argument can be perturbed. We picked in tests fixed $\delta_s = \delta_v = 1/20$ as a trade–off between the exploitation speed and precision.
3. Let $\langle s\rangle$ be the current strength of I, and $\langle s\rangle'$ the modified one. Let
$$\langle s\rangle' = \max\left(0, \min\left(1, \langle s\rangle + \mathrm{uni}\Big(-\frac{\Phi_s}{2}, \frac{\Phi_s}{2}\Big)\right)\right), \tag{2}$$
where $\mathrm{uni}(a, b)$ picks a random value in the range $\langle a, b\rangle$ using an uniform distribution.
4. If $\langle s\rangle' = 0$, then let I be removed. Go to 5 only otherwise.
5. If I is of the type $\langle s\rangle$ PUSH v, let the pushed value v be modified to become v':
$$v' = \max\left(-10, \min\left(10, v + \mathrm{uni}\Big(-\frac{\Phi_v}{2}, \frac{\Phi_v}{2}\Big)\right)\right). \tag{3}$$
6. If I is of the type $\langle s\rangle$ COPY $n{:}m$, let the length multiplier m be modified to become m':
$$m' = \max\left(0, \min\left(2, m + \mathrm{uni}\Big(-\frac{\Phi_v}{2}, \frac{\Phi_v}{2}\Big)\right)\right). \tag{4}$$

Note that v and m are clipped to the same range as when a new instruction is created during the exploration.

After the instructions are modified, and if there have been $r > 0$ instructions removed during that process, then let r new instructions be picked, exactly as during the exploration, but let these new instructions be randomly inserted to the program, so that it retains the original number of instructions P_n. Let the strength of each be Φ_s, that is, a small value related to a maximum single modification of an instruction strength.

5.3 Adapting the exploration/exploitation ratio

Let the fitting quality of a candidate $\mathcal{C}$ to $g(x)$ be MSE($\mathcal{C}$). Let the best candidate so far be $\mathcal{B}$. If none, assume MSE($\mathcal{B}$) $= \infty$.

The *search loop* consists of two phases: explore by creating a new candidate $\mathcal{T}$, and then exploit $\mathcal{T}$, with the extent related to the ratio of quality of $\mathcal{T}$ to $\mathcal{B}$. If, after the whole exploitation stage, MSE($\mathcal{T}$) $<$ MSE($\mathcal{B}$), then let $\mathcal{T}$ become the new $\mathcal{B}$. The loop ends, if MSE($\mathcal{B}$) $\leq \text{MSE}_{\max}$, where $\text{MSE}_{\max}$ is a fixed value, chosen depending on the minimum acceptable quality of the candidate which we want to find.

The mentioned extent decides on the exploration/exploitation balance, and also on the distribution of the computation time to the exploitations of candidates $\mathcal{T}$, given their relative quality. As MSE($\mathcal{T}$) may decrease during the exploitation, let the extent be dynamically updated using the current value of MSE($\mathcal{T}$). The extent is represented by a maximum failure count (MFC). The value expresses the maximum possible number of subsequent exploitation steps, which do not result in finding a better candidate. Thus, if a new path to improve $\mathcal{T}$ is found, then the path is given a chance, irrespective of the length of exploitation of the candidate so far. If MFC is reached, the exploitation is terminated, and the search loop returns to the exploration phase as described. MFC is computed for a new $\mathcal{T}$, and, to adapt it dynamically as discussed, also whenever the exploitation of $\mathcal{T}$ results in a candidate with lower MSE($\mathcal{T}$).

Let us consider a formula for MFC. Let its minimum value be $\text{MFC}_{\min}$, so that any candidate is given a moderate chance to improve, irrespective of that candidate's quality. We want, though, to give a considerably greater MFC for candidates whose initial quality is estimated to be decent. Let the formula have a fixed parameter $t_r > 0$ to reflect that:

$$\text{MFC} = \lfloor t_r \left(\tanh \left(t_s \frac{\text{MSE}(\mathcal{T})}{\text{MSE}(\mathcal{B})} - 1 \right) + 1 \right) + \text{MFC}_{\min} + \frac{1}{2} \rfloor. \tag{5}$$

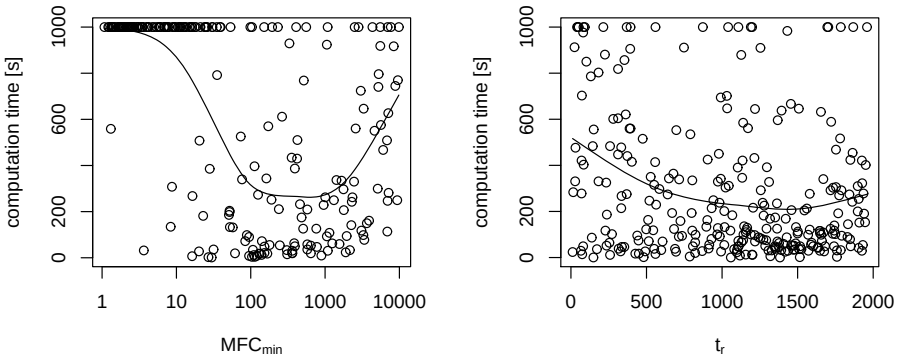
We see a hyperbolic tangent that serves as a threshold function. It is centred around $\frac{\text{MSE}(\mathcal{T})}{\text{MSE}(\mathcal{B})} = 1$. The threshold steepness is regulated by $t_s = 10$.

Let $\text{MFC}_{\min}$ and t_r be adjusted in a test of a relatively complex function, in order to tune the evolutionary method towards more advanced tasks. We will use a set of samples $g^L(x)$, described in detail further in Sect. 6. The parameters in question will be tuned to minimise the average computation time t of the evolutionary method. To save on time, each test will be terminated whenever $t = 1000$. Figure 2(a) shows, that $\text{MFC}_{\min} = 500$ is approximately optimal. Let us then, having that value, test a range of values t_r – a respective diagram in Fig. 2(b) shows a respective optimum value of about $t_r = 1500$.

A diagram of (5), using the adjusted parameters, is shown in Fig. 3.

6 Tests

Let us take advantage of the property of the presented method, that it is not limited to a polynomial approximation, and chose a function to generalise $g(x)$



(a) (b)

Fig. 2. Diagrams of computation time against (a) different $\mathrm{MFC}_{\min}$; for each sample, t_r is randomly picked using $\mathrm{uni}(0, 2000)$; (b) different t_r and $\mathrm{MFC}_{\min} = 500$. The solid lines are smooth approximations.

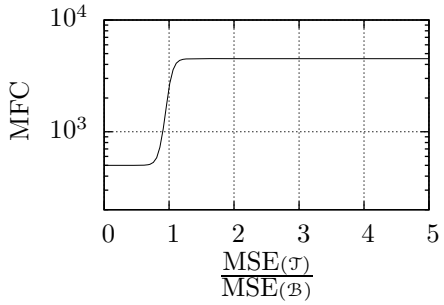


Fig. 3. A diagram of maximum failure count, given the ratio of MSE between the exploited candidate and the best one.

that contains a periodic function. Let us also make $g(x)$ relatively non–trivial for an evolutionary approach, by making the diagrams of its sub–components not resembling $g(x)$ – this may make it less possible, that such a sub–component alone would be considered interesting enough in the exploitation stage to be gradually completed by other sub–components. We will also model $g(x)$ so that along a certain part of its domain it is very similar to a function $y = x^2$. The parabola will serve as a honey–pot, that will attempt to distract our evolutionary method by a deep and, thanks to the symbolic simplicity of the honey pot, easily reachable local minimum. Let $g(x)$ be

$$y = 8\big(1 + \sin(0.59x + 4.6)\big). \tag{6}$$

The diagram in Fig. 4 confirms, that the sub–functions of $g(x)$ are dissimilar to $g(x)$, fitting–wise.

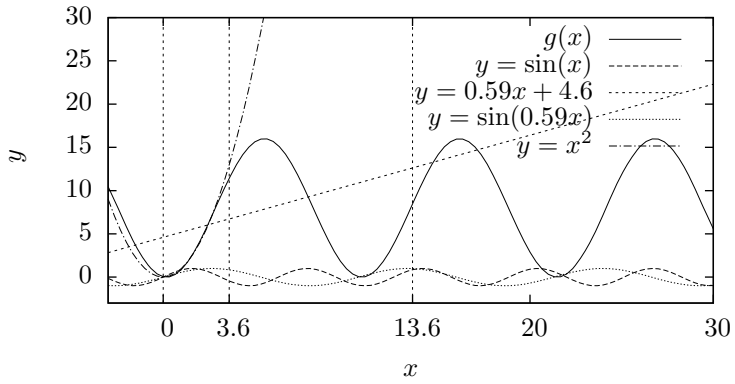


Fig. 4. A diagram of $f(x)$, some of its sub-components, and the honey–pot parabola.

An ideal generalising program is shown in Fig. 5. There are 9 instructions in

```
0   <1> PUSH 0.59
1   <1> MUL
2   <1> PUSH 4.6
3   <1> ADD
4   <1> SIN
5   <1> PUSH 1.0
6   <1> ADD
7   <1> PUSH 8.0
8   <1> MUL
```

Fig. 5. A program representing exactly $g(x)$.

the program, but we will allow a larger $P_n = 15$, as it is unlikely, that such a compact candidate will be found, especially without any preceding intermediate stages with a larger number of instructions. The program requires only two elements in the stack, but let the stack size be $P_s = 10$ because of the reasons similar to the above ones.

Let there be two sets of samples for the evolutionary method. A 'short' set $g^S(x)$ contains 30 samples of $g(x)$ only from within a domain fragment $S = \langle 0.1, 3.6 \rangle$, and a 'long' set $g^L(x)$ contains 30 samples from $L = \langle 0.1, 13.6 \rangle$. Both fragments start from 0.1 and not from 0, so that a lesser percentage of candidates become invalid because of a possible division by zero.

To convey the computational efficiency, any of the tests will be limited with a maximum computation time of 1000 seconds, and a number of tests that succeeded in fulfilling this criterion will be given.

6.1 Extrapolation of an inflection

We see that $g^S(x)$ is very similar to the honey–pot, and as the honey–pot lacks any inflection point, the hint to the evolutionary method that $g(x)$ has an inflection point within S would be rather subtle. Let us use the presented method to

extrapolate $g(x)$ outside S, to see if the method was sensitive to that subtle hint. Let $\mathrm{MSE}_{\max} = 1 \cdot 10^{-5}$ be rather small, as $g^S(x)$ is easy to fit by the discussed method.

Out of the 20 tests run, 6 met the 1000 seconds criterion. Their generalising diagrams are illustrated in Fig. 6. Five of these could be described as reasonable

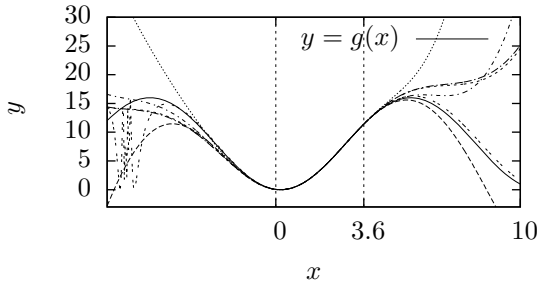


Fig. 6. A diagram $g(x)$, and of a set of 6 functions fitting to $g^S(x)$ at $\mathrm{MSE}_{\max} = 1 \cdot 10^{-5}$.

extrapolations, given the limited domain fragment covered by $g^S(x)$ – various interpretations of the slopes, of a stationary point and of a deflection point are seen. We see that the evolutionary method was able to propose interpretations very different from the honey–pot function. The sixth extrapolations contains garbage to the left of $g^S(x)$.

6.2 Extrapolation of periodicity

The set $g^L(x)$ presents a strong hint of being periodical. We expect the genetic programs to finely extrapolate that periodicity. Let $\mathrm{MSE}_{\max} = 1 \cdot 10^{-4}$ be larger than the error threshold used in the previous test, as $g^L(x)$ turns out to be relatively more difficult to fit. This time, out of 20 tests run, all met the 1000 seconds criterion. The generalising diagrams of the first 10 tests are illustrated in Fig. 7. We see a visually almost perfect fit, despite the relaxed value of $\mathrm{MSE}_{\max}$. A fitting error shows, though, that the generalising programs are not identical to $g(x)$, and that the fitting quality decreases on average as the extrapolation distance increases. Browsing the code of some of the respective final programs, depicted in Fig. 8, confirms, that $g(x)$ appears to be simpler than the generalising programs.

7 Discussion

If we look at the code of the final candidates, shown in Fig. 8, it is seen, that they still contain instructions of very different strengths, even if the generalised function could be defined only in terms of instructions having a binary strength of 0 or 1. The fractional property of the programs serves thus not only the purpose

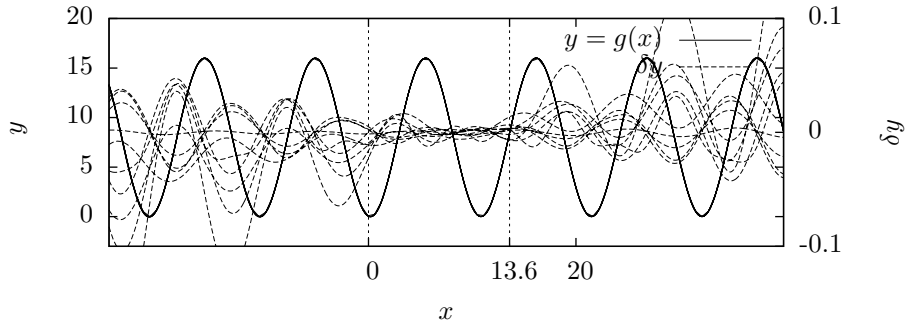


Fig. 7. A diagram $g(x)$, a set of 10 functions fitting to $g^L(x)$ at $\mathrm{MSE}_{\max} = 1 \cdot 10^{-4}$, and a fitting error δy of each.

of a continuous evolution, but also extends the expressivity of the candidates. Consider e.g. the following program: ⟨1⟩ PUSH x; ⟨0.5⟩ MUL, which computes x^2, even that x is pushed only once, and no power instruction is present.

As the programs utilise common algebraic operators and common mathematical functions, a question might be raised, if elegant symbolic equations could be extracted from them. If a generalising program appears to be closely fitting, yet representing a much complex equation that the generalised function, like in the case of the test in Sect. 6.2, then perhaps a decrease of $\mathrm{MSE}_{\max}$ might refine the programs up to the point of making some of the fractional instructions effectively almost disappear, by making their strengths very close to zero. An algorithm simplifying a symbolic equation might then be applied, which would hypothesise, that such instructions are redundant, and would remove them completely before simplifying the rest of the program in order to extract a symbolic equation.

References

1. Alba, E., Dorronsoro, B.: The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. Evolutionary Computation, IEEE Transactions on 9(2), 126–142 (2005)
2. Charbonneau, P.: Release notes for pikaia 1.2. Tech. rep., NCR/NT-451+ STR, NCR Technical Note, Boulder, Colorado (2002)
3. Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: 1st International Conference on Genetic Algorithms. pp. 183–187. Carnegie-Mellon University, Pittsburgh, PA, USA (1986)
4. Eiben, A., Schippers, C.: On evolutionary exploration and exploitation. Fundamenta Informaticae 35(1-4) (1998)
5. Freitas, R., Gómez-Marín, C., Wilson, J.M., Casares, F., Gómez-Skarmeta, J.L.: Hoxd13 Contribution to the Evolution of Vertebrate Appendages. Developmental Cell 23(6), 1219–1229 (2012)
6. Goldberg, D.E.: Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Systems 5, 139–167 (1990)

```
0   <0.5775> MUL
1   <0.3946> PUSH -6.7771
2   <0.9771> POP
3   <0.1400> PUSH 6.8841
4   <0.1872> NEG
5   <0.5722> SIN
6   <0.2324> PUSH 7.4358
7   <0.5234> COPY 2:1.6608
8   <0.1365> PUSH 8.9045
9   <0.1909> PUSH 6.7019
10  <0.2116> PUSH -5.8994
11  <0.2126> NEG
12  <0.8238> MUL
13  <0.2401> ADD
14  <0.1423> ADD
```

```
0   <0.3185> PUSH 7.1064
1   <0.2389> NEG
2   <0.0788> PUSH -6.3361
3   <0.9691> SIN
4   <0.4833> ADD
5   <0.2392> ADD
6   <0.8587> COPY 1:1.1385
7   <0.4592> PUSH 7.4412
8   <0.3479> MUL
9   <0.8930> PUSH 1.8314
10  <0.4190> POP
11  <0.8485> COPY 1:1.1542
12  <0.0793> COPY 1:1.1391
13  <0.0841> ADD
14  <0.0164> COPY 1:0.6350
```

```
0   <0.6323> ADD
1   <0.1373> NEG
2   <0.2402> PUSH -9.4909
3   <0.6012> NEG
4   <0.8460> COPY 1:0.1593
5   <0.1655> PUSH -4.3535
6   <0.2287> COPY 2:1.2507
7   <0.2647> PUSH 6.4488
8   <0.8651> SIN
9   <0.1737> POP
10  <0.3067> PUSH -4.2082
11  <0.6459> PUSH -7.4563
12  <0.8002> MUL
13  <0.1821> ADD
14  <0.1140> ADD
```

```
0   <0.9452> PUSH -9.3873
1   <0.6560> PUSH -5.0818
2   <0.2104> PUSH -5.6119
3   <0.4147> INV
4   <0.7736> COPY 1:1.3795
5   <0.6274> POP
6   <0.3134> COPY 2:0.5906
7   <0.6651> COPY 1:0.9763
8   <0.6518> SIN
9   <0.4214> POP
10  <0.8910> SIN
11  <0.4455> ADD
12  <0.2546> PUSH 7.0741
13  <0.4902> COPY 1:1.7485
14  <0.3605> MUL
```

Fig. 8. Example programs fitting to $g^L(x)$. To save space, digits only up to the 4th decimal place are shown.

7. Herrera, F., Lozano, M., Sánchez, A.M.: A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems 18(3), 309–338 (2003)
8. Kenneth, H., Robbins, K.A., von Ronne, J.: FIFTH: A stack based GP language for vector processing. In: EuroGP'07 Proceedings of the 10th European Conference on Genetic Programming. vol. 1, pp. 102–113. Springer–Verlag Berlin, Heidelberg, Valencia, Spain (2007)
9. Lin, L., Gen, M.: Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation. Soft Computing 13(2), 157–168 (2009)
10. Lindholm, T., Yellin, F.: Java virtual machine specification. Addison-Wesley Longman Publishing Co., Inc. (1999)
11. Oltean, M., Groşan, C., Dioşan, L., Mihăilă, C.: Genetic programming with linear representation: a survey. International Journal on Artificial Intelligence Tools 18(02), 197–238 (2009)
12. Perkis, T.: Stack-based genetic programming. In: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. vol. 1, pp. 148–153. Orlando, FL, USA (1994)
13. Smart, W., Zhang, M.: Continuously evolving programs in genetic programming using gradient descent. In: Mckay, R.I., Cho, S.B. (eds.) Proceedings of The Second Asian-Pacific Workshop on Genetic Programming. Cairns, Australia (2004)
14. Wright, A.H., et al.: Genetic algorithms for real parameter optimization. In: Foundations of Genetic Algorithms. pp. 205–218 (1990)

From EBNF to PEG

Extended Abstract

Roman R. Redziejowski

roman.redz@swipnet.se

1 Introduction

This is a continuation of paper [5] presented at CS&P'2012 and of its improved version [6]. The subject is conversion of grammars from Extended Backus-Naur Form to Parsing Expression Grammars. Parsing Expression Grammar (PEG), as introduced by Ford [1,2], is essentially a recursive-descent parser with limited backtracking. The parser does not require a separate "lexer" to preprocess the input, and the limited backtracking lifts the LL(1) restriction imposed on top-down parsers.

In spite of its apparent similarity to Extended Backus-Naur Form (EBNF), PEG often defines quite a different language. The question is: when an EBNF grammar can be used as its own PEG parser? As found by Medeiros [3, 4], this is, in particular, true for LL(1) languages. Which is of not much use if we use PEG just to circumvent the LL(1) restriction. But, as noticed in [5], this result is valid for a much wider class of grammars. Take as an example this grammar:

$$\begin{array}{llll} S = A|B & & & \\ A = CxZ & B = DyZ & & \\ C = (a|c)^+ & D = (a|d)^+ & Z = z^+ & \end{array} \tag{1}$$

This grammar is not LL(1), and not even LL(k) for any k: each of A and B may start with any number of a's. But, a PEG parser invoked with input "aaayzz" will first try A, call C accepting "aaa", then finding "y" instead of "x" it will backtrack and successfully accept B.

2 Previous Result

In [5,6], we considered a very simple grammar that has only two forms of rules: "choice" $A = e_1|e_2$ and "sequence" $A = e_1e_2$, where A is the name of the rule, and each of e_1, e_2 is a letter of the input alphabet Σ, the name of a rule, or the empty-word marker ε. Any EBNF grammar or PEG can be reduced to this form by introducing additional rules. The names of rules are the "nonterminals" of the grammar. A nonterminal, a letter, the empty-word marker, or a formula $e_1|e_2$ or e_1e_2 is referred to as an "expression". The set of all expressions of the grammar is denoted by $\mathbb{E}$. The grammar is assumed not to be left-recursive.

Following [3,4], we used the method of "natural semantics" to formally define two interpretations of the grammar: as EBNF and as PEG. The method consists in defining two relations, $\overset{\text{BNF}}{\rightsquigarrow}$ and $\overset{\text{PEG}}{\rightsquigarrow}$.

Relation $\overset{\text{BNF}}{\rightsquigarrow}$ is a subset of $\mathbb{E} \times \Sigma^* \times \Sigma^*$. We write $[e]\ x \overset{\text{BNF}}{\rightsquigarrow} y$ to mean that the relation holds for $e \in \mathbb{E}$ and $x, y \in \Sigma^*$. The relation is formally defined by a set of inference rules shown in Figure 1: it holds if and only if it can be proved using these rules. The rules are so constructed that $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ if and only if the prefix x of xy belongs to the language $\mathcal{L}(e)$ of e according to the EBNF interpretation.

$$\frac{}{[\varepsilon]\ x \overset{\text{BNF}}{\rightsquigarrow} x}\ \textbf{(empty.b)} \qquad \frac{}{[a]\ ax \overset{\text{BNF}}{\rightsquigarrow} x}\ \textbf{(letter.b)}$$

$$\frac{[e_1]\ xyz \overset{\text{BNF}}{\rightsquigarrow} yz \quad [e_2]\ yz \overset{\text{BNF}}{\rightsquigarrow} z}{[e_1 e_2]\ xyz \overset{\text{BNF}}{\rightsquigarrow} z}\ \textbf{(seq.b)}$$

$$\frac{[e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}{[e_1|e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}\ \textbf{(choice.b1)} \qquad \frac{[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}{[e_1|e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}\ \textbf{(choice.b2)}$$

Fig. 1. EBNF semantics

Relation $\overset{\text{PEG}}{\rightsquigarrow}$ is a subset of $\mathbb{E} \times \Sigma^* \times \{\Sigma^* \cup \text{fail}\}$. We write $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} Y$ to mean that the relation holds for $e \in \mathbb{E}$, $x \in \Sigma^*$ and $Y \in \{\Sigma^* \cup \text{fail}\}$. The relation is formally defined by a set of inference rules shown in Figure 2: it holds if and only if it can be proved using these rules. The rules are so constructed that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ if and only if parsing expression e applied to xy consumes x, and $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$ if and only if e fails when applied to x.

$$\frac{}{[\varepsilon]\ x \overset{\text{PEG}}{\rightsquigarrow} x}\ \textbf{(empty.p)}$$

$$\frac{}{[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} x}\ \textbf{(letter.p1)} \qquad \frac{b \neq a}{[b]\ ax \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}\ \textbf{(letter.p2)} \qquad \frac{}{[a]\ \varepsilon \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}\ \textbf{(letter.p3)}$$

$$\frac{[e_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz \quad [e_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} Z}{[e_1 e_2]\ xyz \overset{\text{PEG}}{\rightsquigarrow} Z}\ \textbf{(seq.p1)} \qquad \frac{[e_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}{[e_1 e_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}\ \textbf{(seq.p2)}$$

$$\frac{[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y}{[e_1|e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y}\ \textbf{(choice.p1)} \qquad \frac{[e_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y}{[e_1|e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y}\ \textbf{(choice.p2)}$$

where Y denotes y or fail and Z denotes z or fail.

Fig. 2. PEG semantics

Using these definitions, we obtained a sufficient condition for the two interpretations to be equivalent, namely, that each choice $A = e_1|e_2$ satisfies:

$$\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\,\mathrm{Tail}(A) = \varnothing, \tag{2}$$

where $\mathcal{L}(e)$ is the language of e according to the EBNF interpretation, and $\mathrm{Tail}(A)$ is any string that can follow A, up to the end of input. If S denotes the grammar's starting rule, and \$ is the end-of-text symbol, $\mathrm{Tail}(A)$ is formally defined as the set of strings $y\$$ such that the proof of $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$ for some $w \in \mathcal{L}(S)$ contains a partial proof of $[A]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ for some x.

The meaning of (2) is quite obvious: e_1 must not compete with e_2. The problem is in verifying it, as we have there an intersection of context-free languages whose emptiness is, in general, undecidable. The approach proposed in [5, 6] is to approximate the involved languages by languages of the form $X\Sigma^*$ where $X \subseteq \Sigma^+$. It results in the following condition, stronger than (2):

There exist $X, Y \subseteq \Sigma^+$ such that

$$\begin{aligned} &X\Sigma^* \supseteq \mathcal{L}(e_1), \\ &Y\Sigma^* \supseteq \mathcal{L}(e_2)\,\mathrm{Tail}(A), \\ &X \asymp Y, \end{aligned} \tag{3}$$

where $X \asymp Y$ means $X\Sigma^* \cap Y = Y\Sigma^* \cap X = \varnothing$.

The sets X and Y can, in particular, be the sets of possible first letters of words in $\mathcal{L}(e_1)$ respectively $\mathcal{L}(e_2)\,\mathrm{Tail}(A)$. For such sets, $X \asymp Y$ is equivalent to $X \cap Y = \varnothing$, and the condition is identical to LL(1).

Even if the language is not LL(1), it may satisfy (2) if instead of single letters we take some longer prefixes. A natural way to approximate $\mathcal{L}(e)$ by $X\Sigma^*$ is to take as X the set of strings accepted by the first parsing procedures possibly called by e. Or the first procedures called by them. If such approximations X, Y satisfying (3) above exist, we have a parser that chooses its way by examining the input ahead within the reach of one parsing procedure. It was suggested use the name LL(1p) for languages that can be so parsed.

To find the possible approximations by first procedures, we used the relation `first` where `first`(e) is the set of procedures called as first directly from e.

3 Beyond LL(1p)

In the example grammar (1), the first procedures of A and B are, respectively, C and D, and $\mathcal{L}(C) \not\asymp \mathcal{L}(D)$, so this grammar is not LL(1p). However, $X = \mathcal{L}(Cx)$ and $Y = \mathcal{L}(Dz)$ satisfy (3), guaranteeing that the grammar defines the same language under both interpretation. Here the parser chooses its way by looking at the text ahead within the reach of two parsing procedures. We can refer to such grammar as being LL(2p). As we remarked before, it is not LL(2).

Checking if the grammar is LL(kp) requires finding possible sets of first k procedures. This can be done using relation $\texttt{first}_k$, similar to that used for

checking LL(k). Although the sets become large for larger k, it is a mechanical procedure. However, checking of the relation $\asymp$ between these sets may not be simple as it involves intersection of context-free languages. If we are lucky, the languages may be regular, as in the above example. But in general, using approximation by first procedures to check (2) is not always feasible, even for $k = 1$.

4 Looking Farther Ahead

The mechanism used above to look far ahead in the input is the backtracking of PEG. But, this backtracking is limited. When faced with $e_1|e_2$, the parser cannot look ahead beyond e_1 and then backtrack if it does not like what it sees there. There exist grammars where the parser has to look beyond e_1 to make the correct decision. An example is the following grammar, modeled after [3, 4]:

$$\begin{array}{ll} S = Xz & \\ X = A|B & A = ab|C \\ B = a|Cd & C = c \end{array} \tag{4}$$

Interpreted as PEG, this grammar does not accept the string cdz that belongs to $\mathcal{L}(S)$: A succeeds on c via C, leaving no chance to B. The grammar is not LL(1p), nor even LL(kp) in the sense defined above. However, the grammar is LL(2): a top-down parser can choose between A and B by looking at two letters ahead: they are ab or cz for A and az or cd for B. The reason for the failure of PEG is that X cannot look beyond A when faced with c as the first letter.

PEG has a special operation to examine the input ahead: the "and-predicate" $\&e$. It means: "invoke the expression e on the text ahead and backtrack; return success if e succeeded or failure if it failed". This can be formally defined by two inference rules:

$$\frac{[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y}{[\&e]\ xy \overset{\text{PEG}}{\rightsquigarrow} xy}\ \textbf{(and.p1)} \qquad \frac{[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}{[\&e]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}\ \textbf{(and.p2)}$$

In order to look beyond e_1 in $A = e_1|e_2$, we can modify the grammar by adding $\&e_0$ after e_1, obtaining $A = e_1\&e_0|e_2$.

Consider as an example the grammar (4). The only rule that does not satisfy (2) is $X = A|B$. One can easily see that by replacing it with $X = A\&z|B$, we obtain a PEG defining the same language as the EBNF grammar (4). This idea has been used in [3, 4] to construct PEGs for LL(k) languages. We consider it here for a wider class of languages.

We are going to consider the grammar where each choice has the form $A = e_1\&e_0|e_2$. EBNF does not have the and-predicate; in order to speak of two interpretations of the grammar, we define $\&e$ to be a dummy EBNF operation with $\mathcal{L}(\&e) = \varepsilon$:

$$\frac{}{[\&e]\ x \overset{\text{BNF}}{\rightsquigarrow} x} \quad \textbf{(and.b)}$$

The problem is to choose the expression e_0. We are going to show that the two interpretations are equivalent if each choice $A = e_1 \& e_0 | e_2$ satisfies these conditions:

$$\text{Parsing expression } e_0 \text{ succeeds on every } w \in \text{Tail}(A), \tag{5}$$

$$\mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^* \cap \mathcal{L}(e_2)\,\text{Tail}(A) = \varnothing\ . \tag{6}$$

(Note that by taking $e_0 = \varepsilon$, we obtain an &-free choice and (5,6) become identical to (2).)

The demonstration consists of three Propositions:

Proposition 1. *For every $e \in \mathbb{E}$ and $w \in \Sigma^*$, there exists a proof of either $[e]\ w \overset{PEG}{\rightsquigarrow}$ fail or $[e]\ w \overset{PEG}{\rightsquigarrow} y$ where $w = xy$ for some x.*

Proof. This is proved in [3] using a result from [2]. A self-contained proof given in [6] is easily extended to include the and-predicate.

Proposition 2. *For each $e \in \mathbb{E}$ and $x, y \in \Sigma^*$, $[e]\ xy \overset{PEG}{\rightsquigarrow} y$ implies $[e]\ xy \overset{BNF}{\rightsquigarrow} y$.*

Proof. This is proved as Lemma 4.3.1 in [3]. The proof is easily extended to include the and-predicate in EBNF.

Proposition 3. *If every choice $A = e_1 \& e_0 | e_2$ satisfies (5,6) then for every $w \in \mathcal{L}(S)$ there exists a proof of $[S]\ w\$ \overset{PEG}{\rightsquigarrow} \$$.*

Proof. We show that for every partial result $[e]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ in the proof of $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$ there exists a proof of $[e]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$. We use induction on the height n of the proof tree for $[e]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$.

The case of $n = 1$ is easy. Take any $n \geq 1$ and assume the Proposition holds for every tree of height less or equal n. Consider a proof of $[e]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ having height $n + 1$. The only non-trivial situation is a proof tree of height $n + 1$ where the last step results in $[A]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ for $A = e_1 \& e_0 | e_2$. Two cases are possible:

Case 1: The result is derived from $[e_1]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ using **and.b**, **seq.b** and **choice.b1**. By induction hypothesis there exists proof of $[e_1]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$. By definition, $y\$ \in \text{Tail}(A)$. As e_0 succeeds on each string in $\text{Tail}(A)$, we have $[\&e_0]\ y\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$ from **and.p1**. We can construct a proof of $[e_1 \& e_0 | e_2]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$ using **seq.p1** and **choice.p1**.

Case 2: The result is derived from $[e_2]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ using **and.b**, **seq.b** and **choice.b2**. By induction hypothesis there exists proof of $[e_2]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$. In order to use **choice.p2**, we have to show that $[e_1 \& e_0]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow}$ fail.
Suppose this is not true. Then, by Proposition 1 exist proofs of $[e_1]\ uv\$ \overset{\text{PEG}}{\rightsquigarrow} v\$$ and $[\&e_0]\ v\$ \overset{\text{PEG}}{\rightsquigarrow} v\$$ for some u, v such that $uv = xy$. By Proposition 2 exists proof of $[e_1]\ uv\$ \overset{\text{BNF}}{\rightsquigarrow} v\$$, which means $u \in \mathcal{L}(e_1)$. The proof of $[\&e_0]\ v\$ \overset{\text{PEG}}{\rightsquigarrow} v\$$ requires $[e]\ ts\$ \overset{\text{PEG}}{\rightsquigarrow} s\$$ for some t, s such that $ts = v$. By Proposition 2 exists proof of $[e_0]\ ts\$ \overset{\text{BNF}}{\rightsquigarrow} s\$$, which means $t \in \mathcal{L}(e_0)$. Thus $xy = uts \in \mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^*$. But $[e_2]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$ means $xy \in \mathcal{L}(e_2)\,\text{Tail}(A)$, which contradicts (6). □

One can easily see that the necessary condition for the required e_0 to exist is:

$$\mathcal{L}(e_1)\,\mathrm{Tail}(A) \cap \mathcal{L}(e_2)\,\mathrm{Tail}(A) = \varnothing\ .$$

A systematic way of choosing a suitable e_0 is still to be found.
It seems that for the LL(k) languages e_0 should be the expression consuming exactly FOLLOW$_k(A)$.

References

1. Ford, B.: Packrat parsing: a practical linear-time algorithm with backtracking. Master's thesis, Massachusetts Institute of Technology (2002) `http://pdos.csail.mit.edu/papers/packrat-parsing:ford-ms.pdf`
2. Ford, B.: Parsing expression grammars: A recognition-based syntactic foundation. In Jones, N.D., Leroy, X., eds.: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, ACM (2004) 111–122
3. Medeiros, S.: Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro (2010)
4. Mascarenhas, F., Medeiros, S., Ierusalimschy, R.: On the relation between context-free grammars and Parsing Expression Grammars. UFRJ Rio de Janeiro, UFS Aracaju, PUC-Rio, Brazil (2013) `http://arxiv.org/pdf/1304.3177v1`
5. Redziejowski, R.R.: From EBNF to PEG. In Popova-Zeugmann, L., ed.: Proceedings of the 21th International Workshop on Concurrency, Specification and Programming Berlin, Germany, September 26-28, 2012, Humboldt University of Berlin (2012) 324–335 `http://ceur-ws.org/Vol-928/`
6. Redziejowski, R.R.: From EBNF to PEG. Fundamenta Informaticae (2013) to appear

Towards an Object-Oriented Programming Language for Physarum Polycephalum Computing

Andrew Schumann[1] and Krzysztof Pancerz[1,2]

[1] University of Information Technology and Management
Sucharskiego Str. 2, 35-225 Rzeszów, Poland
aschumann@wsiz.rzeszow.pl
[2] University of Management and Administration
Akademicka Str. 4, 22-400 Zamość, Poland
kpancerz@wszia.edu.pl

Abstract. In the paper, we present foundations of a new object-oriented programming language for Physarum polycephalum computing. Both, theoretical foundations and assumptions for a language specification are considered. Physarum polycephalum is a one-cell organism. In the phase of plasmodium, its behavior can be regarded as a biological substrate that implements the Kolmogorov-Uspensky machine which is the most generalized and nature-oriented version of a mathematical machine. The proposed language will be used for developing programs for Physarum polycephalum by the spatial configuration of stationary nodes (inputs).

Keywords: Physarum polycephalum, unconventional computing, nature-inspired computing, object-oriented programming language, Kolmogorov-Uspensky machine

1 Introduction

Physarum polycephalum is a one-cell organism belonging to Physarales, subclass Myxogastromycetidae, class Myxomycetes and division Myxostelida. In the phase of plasmodium, it looks like an amorphous giant amoeba with networks of protoplasmic tubes. It feeds on bacteria, spores and other microbial creatures (substances with potentially high nutritional value) by propagating towards sources of food particles and occupying these sources. A network of protoplasmic tubes connects the masses of protoplasm. As a result, the plasmodium develops a planar graph, where the food sources or pheromones are considered as nodes and protoplasmic tubes as edges. This fact allows us to claim that plasmodium behavior can be regarded as a biological implementation of Kolmogorov-Uspensky machines [7]. The modification of locations of nutrients (food sources) causes a storage modification of plasmodium. Hence, the plasmodium may be used for developing a biological architecture of different abstract automata such as Kolmogorov-Uspensky machines [16, 22], Tarjan's reference machine [21], and

Schönhage's storage modification machines [19, 20]. In *Physarum Chip Project: Growing Computers From Slime Mould* [11] supported by FP7 we are going to implement programmable amorphous biological computers in plasmodium of Physarum. This abstract computer is called *slime mould based computer*.

One of the paths of our research in this area concerns creating a new programming language that simulates plasmodium behavior. The following main tasks can be distinguished in the first step of this path:

1. Constructing the programming language on the basis of storage machines. The static storage structure is represented by a two-dimensional configuration of point-wise sources of chemo-attractants and chemo-repellents.
2. Constructing the programming language on the basis of the Kolmogorov-Uspensky machine (KUM), where edges are represented by protoplasmic strands.
3. Developing programs represented by the spatial configuration of stationary nodes (treated as inputs of the programs). Outputs of the programs may be recorded optically.

The rest of the paper is organized as follows. In Section 3, we give foundations of specification of a new language. Assumptions of specification are preceded by a theoretical background of Physarum automata (see Section 2).

2 Physarum Automata

Plasmodium's active zones of growing pseudopodia interact concurrently and in a parallel manner. At these active zones, three basic operations stimulated by nutrients and some other conditions can be observed: fusion, multiplication, and direction operations. The *fusion* $Fuse$ means that two active zones A_1 and A_2 both produce new active zone A_3 (i.e. there is a collision of the active zones). The *multiplication* $Mult$ means that the active zone A_1 splits into two independent active zones A_2 and A_3 propagating along their own trajectories. The *direction* $Direct$ means that the active zone A is not translated to a source of nutrients but to a domain of an active space with a certain initial velocity vector v. These operations, $Fuse$, $Mult$, $Direct$, can be determined by the following stimuli:

- The set of attractants $\{N_1, N_2, \ldots\}$. Attractants are sources of nutrients or pheromones, on which the plasmodium feeds. Each attractant N is characterized by its position and intensity. It is a function from one active zone to another.
- The set of repellents $\{R_1, R_2, \ldots\}$. Plasmodium of Physarum avoids light and some thermo- and salt-based conditions. Thus, domains of high illumination (or high grade of salt) are repellents such that each repellent R is characterized by its position and intensity, or force of repelling. In other words, each repellent R is a function from one active zone to another.

Such plasmodium behavior can be presented as an implementation of some abstract automata.

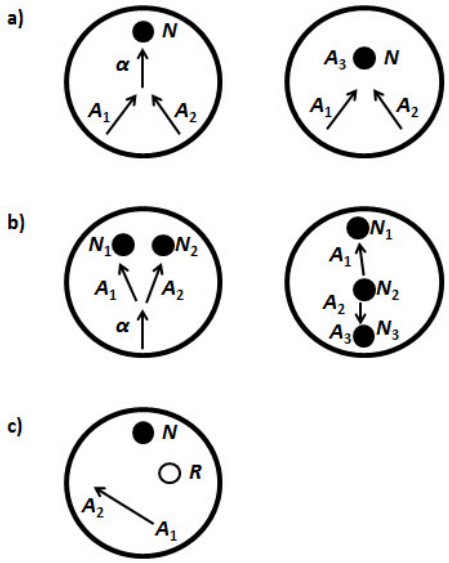


Fig. 1. The stimulation of the following operations in Physarum automata: (a) fusion, (b) multiplication, and (c) direction, where A_1, A_2, A_3 are active zones, N, N_1, N_2, N_3 are attractants, α is a protoplasmic tube, R is a repellent.

2.1 Physarum Cellular Automata

Recall that a cellular automaton is a 4-tuple $\mathcal{A} = \langle \mathbf{Z}^d, S, u, f \rangle$, where (1) $d \in \mathbf{N}$ is a number of dimensions, and the members of $\mathbf{Z}^d$ are referred to as cells, (2) S is a finite set of elements called the states of an automaton $\mathcal{A}$, the members of $\mathbf{Z}^d$ take their values in S, (3) $u \subset \mathbf{Z}^d \setminus \{0\}^d$ is a finite ordered set of n elements, $u(x)$ is said to be a neighborhood for the cell x, (4) $f: S^{n+1} \to S$ that is f is the local transition function (or local rule). As we see an automaton is considered on the endless d-dimensional space of integers, i.e., on $\mathbf{Z}^d$. Discrete time is introduced for $t = 0, 1, 2, \ldots$ For instance, the cell x at time t is denoted by x^t. Each automaton calculates its next state depending on states of its closest neighbors. The cellular automata thus represent locality of physics of information and massive-parallelism in space-time dynamics of natural systems.

In abstract cellular automata, cells are physically identical. They can differ just by one of the possible states of S. In case of Physarum, cells can possess different topological properties. This depends on intensity of chemo-attractants and chemo-repellents. The intensity entails the natural or geographical neighborhood of the set's elements in accordance with the spreading of attractants or repellents. As a result, we obtain Voronoi cells. Let us define what they are mathematically. Let $\mathbf{P}$ be a nonempty finite set of planar points and $|\mathbf{P}| = n$. For points $p = (p_1, p_2)$ and $x = (x_1, x_2)$ let $d(p, x) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2}$ denote their Euclidean distance. A planar Voronoi diagram of the set $\mathbf{P}$ is a partition of the plane into cells, such that for any element of $\mathbf{P}$, a cell corresponding

to a unique point p contains all those points of the plane which are closer to p in respect to the distance d than to any other node of $\mathbf{P}$. A unique region

$$vor(p) = \bigcap_{m \in \mathbf{P}, m \neq p} \{z \in \mathbf{R}^2 : d(p,z) < d(m,z)\}$$

assigned to a point p is called a *Voronoi cell* of the point p. Within one Voronoi cell a reagent has a full power to attract or repel the plasmodium. The distance d is defined by the intensity of reagent spreading. A reagent attracts or repels the plasmodium and the distance, on which it is possible, corresponds to the elements of a given planar set $\mathbf{P}$. When two spreading wave fronts of the two reagents meet, this means that on the board of meeting the plasmodium cannot choose its one further direction and splits (see Figure 2).

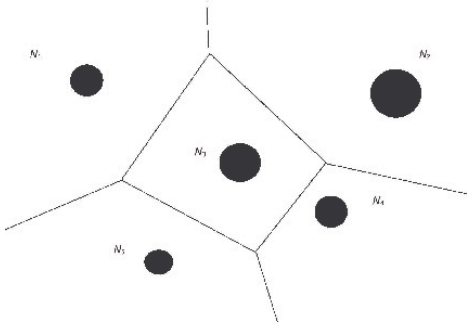


Fig. 2. The Voronoi diagram for Physarum, where different attractants have different intensity and power.

The direction of protoplasmic tubes is defined by concentrations of chemo-attractants or chemo-repellents in Voronoi neighborhood. Each dynamics of protoplasmic tube can be characterized at time step t by its current position x_t and the angle α_t.

2.2 Physarum Kolmogorov-Uspensky Machines

Let Γ be an alphabet, k a natural number. We say that a tree is (Γ, k)-tree, if one of nodes is designated and is called *root* and all edges are directed. Each node is labeled by one of the signs of Γ and each edge from the same node is labeled by different numbers $\{1, \ldots, k\}$ (so, each node has not more than k edges). We see that by this definition of (Γ, k)-tree, the pseudopodia growing from the one active zone, where all attractants are labeled by signs of Γ, and protoplasmic tubes are labeled by numbers of $\{1, \ldots, k\}$, is a (Γ, k)-tree.

Let r be the maximal possible path of $(\Gamma; k)$-tree. We can always design Physarum Voronoi diagrams (using attractants and repellents) for inducing different numbers r and appropriate local properties. The $(\Gamma; k)$-tree limited by

r is called $(\Gamma; k)$-complex. Programming in Kolmogorov-Uspensky machines is considered as transforming one $(\Gamma; k)$-complex to another with the same r by changing nodes and edges using some rules. In case of Physarum implementation of Kolmogorov-Uspensky machines programming is presented as transforming one Voronoi diagram into another with the same r by dynamics of Physarum (e.g. when some attractants become eaten by Physarum).

The simpler version of the Kolmogorov-Uspensky machines is presented by Schönhage's storage modification machines.

2.3 Physarum Schönhage's Storage Modification Machines

These machines consist of a fixed alphabet of input symbols, Γ, and a mutable directed graph with its arrows labeled by Γ. The set of nodes X, identified with attractants is finite, as well. One fixed node $a \in X$ is identified as a distinguished center node of the graph. It is the first active zone of growing pseudopodia. The distinguished node a has an edge x such that $x_\gamma(a) = a$ for all $\gamma \in \Gamma$. That is, all pointers from the distinguished center node point back to the center node. Each $\gamma \in \Gamma$ defines a mapping x_γ from X to X. Each word of symbols in the alphabet Γ is a pathway through the machine from the distinguished center node.

Schönhage's machine modifies storage by adding new elements and redirecting edges. Its basic instructions are as follows:

- Creating a new node: **new** W. The machine reads the word W, following the path represented by the symbols of W until the machine comes to the last symbol in the word. It causes a new node y, associated with the last symbol of W, to be created and added to X. Adding a new node means adding a new attractant within a Physarum Voronoi diagram.
- A pointer redirection: **set** W **to** V. This instruction redirects an edge from the path represented by word W to a former node that represents word V. It means that we can remove some attractants within a Physarum Voronoi diagram.
- A conditional instruction: **if** $V = W$ **then instruction** Z. It compares two paths represented by words W and V and if they end at the same node, then we jump to instruction Z, otherwise we continue. This instruction serves to add edges between existing nodes. It corresponds to the splitting or fusion of Physarum.

3 Foundations of Specification of an Object-Oriented Programming Language for Physarum Polycephalum

The plasmodium of Physarum polycephalum functions as a parallel amorphous computer with parallel inputs and parallel outputs. Data are represented by spatial configurations of sources of nutrients. Therefore, we can generally assume that a program of computation is coded via configurations of repellents and attractants. The plasmodium of Physarum polycephalum is a computing

substrate. In [10], Adamatzky underlined that Physarum does not compute. It obeys physical, chemical and biological laws. Its behavior can be translated to the language of computations.

In this section, we deal with foundations of specification of a new object-oriented programming language for Physarum polycephalum computing on the basis of using a Voronoi diagram for implementing Kolmogorov-Uspensky machines. In an object-oriented programming (OOP) paradigm, concepts are represented as objects that have data fields (properties describing objects) and associated procedures known as methods. The OOP approach assumes that properties describing objects are not directly accessible by the rest of the program. They are accessed by calling special methods, which are bundled in with the properties. This approach has been implemented in our new language. Moreover, we have referred to conventions used in the JavaBeans API [4], i.e., the object properties must be accessible using *get*, *set*, and *is* (used for Boolean properties instead of *get*). They are called accessor methods. For readable properties, there are getter methods reading the property values. For writable properties, there are setter methods allowing the property values to be set or updated.

Our new language has been proposed as a prototype-based programming language like, for example, Self [1], JavaScript and other ECMAScript implementations [2]. Unlike traditional class-based object-oriented languages, it is based on a style of object-oriented programming in which classes are not present. Behavior reuse is performed via a process of cloning existing objects that serve as prototypes. This model is also known as instance-based programming.

Table 1. Main objects identified in Physarum polycephalum computing

Object	Properties
Layer	*id*, *size*, *elements*
Physarum	*id*, *position*, *intensity*
Attractant	*id*, *position*, *intensity*
Repellent	*id*, *position*, *intensity*

The main objects identified in Physarum polycephalum computing are collected in Table 1. We assume that a computational space is divided into two-dimensional computational layers on which Physarum polycephalum, as well as attractants and repellents, can be scattered. Our approach allows interaction between elements placed on different layers. This property enables us to use, in the future, the multi-agent paradigm in Physarum polycephalum computing. The user can define, in the computational space, as many computational layers as needed. For each layer, its size can be determined individually. We apply the point-wise configuration of elements scattered on the layers. Therefore, for each element (Physarum, attractant, repellent), its position can be determined using two integers (coordinates). As it was mentioned in Section 2, attractants and

repellents are characterized by the property called intensity. This property plays an important role in creation of the Voronoi cells. For each attractant and repellent, the intensity is a fuzzy value from the interval $[0, 1]$, where 1 denotes the maximal intensity, while 0 the minimal intensity, i.e., a total lack of impact of a given attractant or repellent on Physarum polycephalum. The force of attracting (repelling) of Physarum is a combination of intensity of attractants (repellents) and distances between plasmodium and attractants (repellents), respectively.

Let $p = (p_1, p_2)$ and $x = (x_1, x_2)$ be points on the layer where Physarum and attractant (repellent), respectively, are located. To create the Voronoi cells, we can use the following measure modyfying a distance, which is commonly used:

$$f(p, x) = \frac{1}{\varepsilon(x)} \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2},$$

where $\varepsilon(x)$ is the intensity of attractant (repellent) placed at x. It means that the Voronoi cells cover the force of attracting (repelling) of plasmodium instead of simple distances between it and attractants (repellents). In the current version of the language, the Voronoi cells are built within layers only.

Analogously to layers, the user can create and scatter on layers as many elements as needed.

Below, we present an exemplary fragment of a code in our language responsible for creating the layer and elements, setting individual properties of elements and scattering elements on the layer.

```
l1=new Layer;
p1=new Physarum;
a1=new Attractant;
a2=new Attractant;
a3=new Attractant;
a4=new Attractant;
l1.add(p1);
p1.setPosition(800,200);
l1.add(a1);
a1.setPosition(500,150);
a1.setIntensity(0.7);
l1.add(a2);
a2.setPosition(500,350);
a2.setIntensity(0.5);
l1.add(a3);
a3.setPosition(400,250);
a3.setIntensity(0.6);
l1.add(a4);
a4.setPosition(600,250);
a4.setIntensity(0.5);
```

For experiments with Physarum polycephalum computing, a specialized computer tool (*PhyChip Programming Platform*) is being developed using the Java environment. The tool consists of two main modules:

1. *Code creation and compilation module.* For generating the compiler of our language, the Java Compiler Compiler (JavaCC) tool [3] is used. JavaCC is the most popular parser generator for use with Java applications.
2. *Simulation module.* It enables the user to perform time simulation of growing pseudopodia, i.e., to run the program.

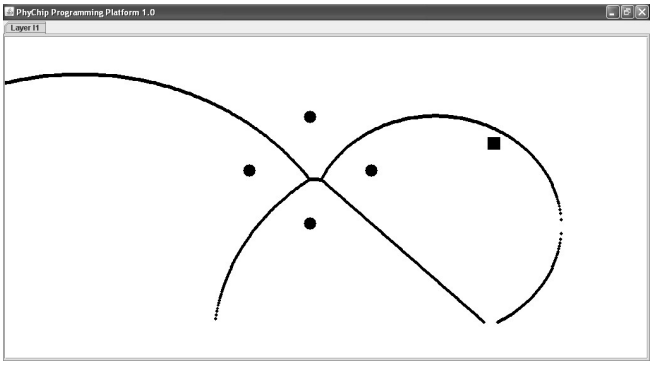


Fig. 3. The Voronoi cells for 4 attractants defined in the exemplary program generated in our tool (attractants are marked with dots whereas Physarum with a square).

In Figure 3, we have shown the Voronoi cells generated in our computer tool for 4 attractants ($a1$, $a2$, $a3$, $a4$) with different intensity assigned to them, defined in the exemplary program. Attractants are marked with dots whereas Physarum with a square. The measure defined earlier has been used to create cells. It is easy to see that Physarum is attracted first of all by the most right attractant.

4 Summation

In the paper, we have outlined theoretical foundations as well as assumptions for a new object-oriented programming language for Physarum polycephalum computing. The next mile steps in our research are the following: implementation of operations based on the π-calculus model [17] of processes and extension of the programming platform to the agent-oriented programming language for computation with raw plasmodium.

Acknowledgments

This research is being fulfilled by the support of FP7-ICT-2011-8.

References

1. Self, http://selflanguage.org/
2. ECMAScript, http://www.ecmascript.org/
3. JavaCC, http://java.net/projects/javacc/
4. JavaBeans. Tech. rep., Sun Microsystems (1997)
5. Adamatzky, A.: Reaction-diffusion algorithm for constructing discrete generalized voronoi diagram. Neural Network World 6, 635–643 (1994)
6. Adamatzky, A.: Computing in Nonlinear Media and Automata Collectives. Institute of Physics Publishing (2001)
7. Adamatzky, A.: Physarum machine: implementation of a kolmogorov-uspensky machine on a biological substrate. Parallel Processing Letters 17(4), 455–467 (2007)
8. Adamatzky, A.: Growing spanning trees in plasmodium machines. Kybernetes 37(2), 258–264 (2008)
9. Adamatzky, A., De Lacy Costello, B., T., S.: Universal computation with limited resources: Belousov-zhabotinsky and physarum computers. International Journal of Bifurcation and Chaos 18(8), 2373–2389 (2008)
10. Adamatzky, A.: Physarum Machines: Computers from Slime Mould. World Scientific (2010)
11. Adamatzky, A., Erokhin, V., Grube, M., Schubert, T., Schumann, A.: Physarum chip project: Growing computers from slime mould. International Journal of Unconventional Computing 8(4), 319–323 (2012)
12. Adamatzky, A., Costello, B.D.L., Asai, T.: Reaction-Diffusion Computers. Elsevier Science, Amsterdam (2005)
13. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press (2005)
14. De Lacy Costello, B., Ratcliffe, N., Adamatzky, A., Zanin, A.L., Liehr, A.W., Purwins, H.G.: The formation of voronoi diagrams in chemical and physical systems: Experimental findings and theoretical models. International Journal of Bifurcation and Chaos 14(7), 2187–2210 (2004)
15. Ilachinski, A.: Cellular Automata: A Discrete Universe. World Scientific (2001)
16. Kolmogorov, A.: On the concept of algorithm. Uspekhi Mat. Nauk 8(4), 175–176 (1953)
17. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Information and Computation 100(1), 1–40 (1992)
18. Pavlović, D., Escardó, M.: Calculus in coinductive form. In: Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science. pp. 408–417 (1998)
19. Schönhage, A.: Real-time simulation of multi-dimensional turing machines by storage modification machines. Project MAC Technical Memorandum 37, MIT (1973)
20. Schönhage, A.: Storage modification machines. SIAM Journal on Computing 9(3), 490–508 (1980)
21. Tarjan, R.: Reference machines require non-linear time to maintain disjoint sets. Tech. Rep. STAN-CS-77-603 (1977)
22. Uspensky, V.: Kolmogorov and mathematical logic. The Journal of Symbolic Logic 57, 385–412 (1992)

About New Version of RSDS System

Zbigniew Suraj and Piotr Grochowalski

Institute of Computer Science
University of Rzeszów, Rzeszów, Poland
{zbigniew.suraj,piotrg}@ur.edu.pl

Abstract. The aim of this paper is to present a new version of a bibliographic database system - Rough Set Database System (RSDS). The RSDS system, among others, includes bibliographic descriptions of publications on rough set theory and its applications. This system is also an experimental environment for research related to the processing of bibliographic data using the domain knowledge and the related information retrieval.

Keywords: rough sets, data mining, knowledge discovery, pattern recognition, database systems.

1 Introduction

The presented RSDS system is a bibliographic system that includes bibliographical references aimed at disseminating information on publications on rough set theory and its applications [4–8]. The system is available for free at http://rsds.univ.rzeszow.pl. Currently about 4000 bibliographical descriptions of publications are collected in its database.
The RSDS system is also an experimental environment for carrying out research related to the broadly defined information processing based on the methods and techniques in the field of ontology and rough sets. In addition, it enables one to analyze the data contained in the database by using the advanced statistical and graphical methods and techniques.

Apart from the bibliographical descriptions, it contains additionally:

- information on software related to the theory and applications of rough sets,
- bibliographies of people who render outstanding services to the development of rough set theory and its applications,
- personal details about the authors of publications whose descriptions are included in the database of this system.

The system was developed in the client-server technology, i.e., the data for the system and mechanisms for handling such data are running on the server and the user with the help of the web browser is able to access the resources.

The work on the system began in 2002 and it is continued. During the works there were developed two versions of the system that have been made available for users [9]-[11],[14]-[15],[19]. In March 2013, the third version of the system

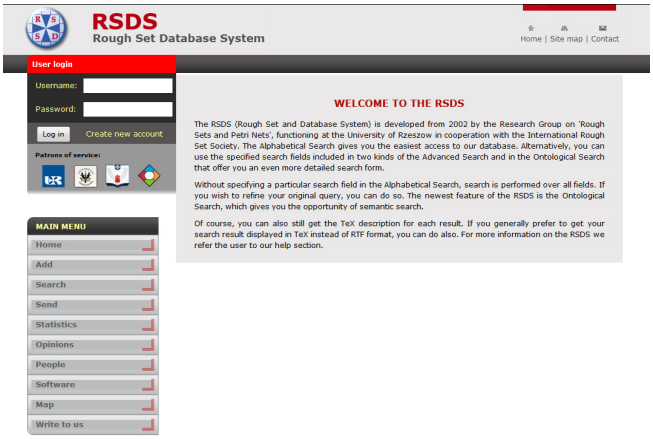


Fig. 1. The main window of the system.

was released. It has been completely rebuilt, allowing the introduction of larger number of facilities for users of the system with the use of modern technology.

In the current version the following changes was introduced:

- exchange and reorganization of the engine of the system (CMS - Drupal),
- rebuilding the website design,
- rebuilding and upgrading the functionalities of the system,
- increasing the role of the administrator(s) of the system - admin panel,
- larger number of facilities for registered users - user panel,
- introduction of the status of data and the possibility of its modification,
- system-to-user communication via e-mail (on important issues).

The rest of the paper is organized as follows. Section 2 describes the logical structure of the RSDS system. Functionalities of the system are presented in Section 3. Section 4 provides description of data for the system. Section 5 gives the system requirements. The future plans for the RSDS system are discussed in section 6.

2 The Logical Structure of the System

The RSDS system structure can be divided into four functional layers. Each of these layers includes modules and with the help of these modules the layers meet specified tasks (see Figure 2):

- The presentation layer with the graphical interface module.
- The application layer with the modules of login, add/edit, search, graph and statistical, download, auxiliary (biographies of people, software, maps).
- The communication layer with the module of communication with the database.

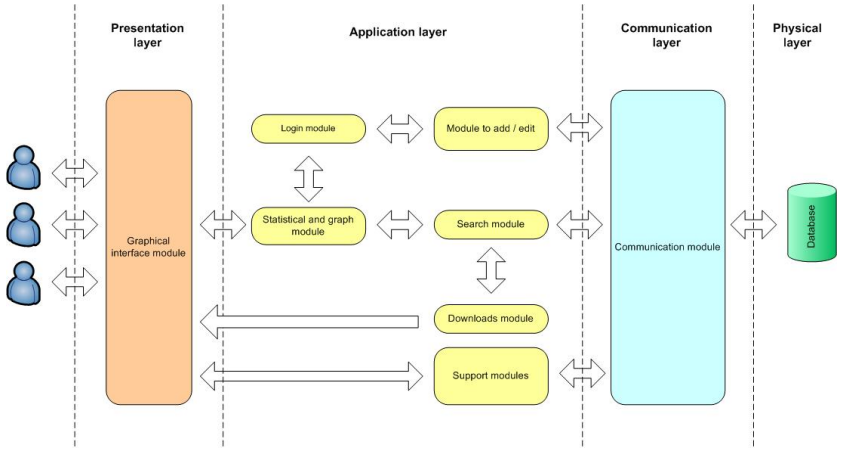


Fig. 2. The logical structure of system.

– The physical layer containing the database.

The purpose of the modules from the presentation layer is to communicate with the user with the usage of created interface.

In the application layer there are the modules implementing the main functionalities of the system. The login module is responsible for the correct handling of the login/logoff process of the users and storing information about the users logged in the system. The add and edit module supports the process of implementing the new data into the system or editing already existing ones. It will ensure the correctness of the input data and its correct assignment to the user-owner. The search module performs the search process of the publications descriptions meeting the criteria set by the user. The operation of this module has been improved by implementing the results of the research connected with the information retrieval in it. The graphic and statistical module is designed not only for the analysis of data in the system but also for the analysis of the activity of the users. The analysis is carried out in various aspects, such as publications, the authors of the publication and the relationships between them. The purpose of this module is the presentation of the results of these analyzes. The download module provides the users with different options of retrieving data from the system in the form of prepared bibliography. Auxiliary modules extend the basic functionality of the system to the biographies of people who render outstanding services to the development of rough set theory and its applications, and descriptions of available software related to the theory of rough sets and also a world map showing where the given problem is growing solved.

The communication layer possesses a module that is responsible for the proper communication with the database, which stores the data for the system.

The physical layer includes a relational database in which the data are stored and presented in the system.

3 System Functionalities

The basic functionalities of the RSDS system include:

- Adding new data.
- Editing existing data.
- Data search.
- Registration of users in the system.
- Saving data to a file.
- Sending data files to the administrator.
- Service of user comments.
- Statistics, analysis of statistical graphs, determining the Pawlak number of the first and second kind, classifier of publications.
- Help.

Capabilities and the content of the RSDS system are constantly extended.

In order to store the held information in the simplest form and to exclude redundancy (redundancy of data), the data for the RSDS system are stored in a relational database. The database structure is based on the BibTeX format [22]. Well defined and uniform structures of the description decided about its choice. Additionally, the possibility of getting the bibliographical descriptions in the BibTeX format included in the system, allowing one to automatically generate bibliographies and attach them to the prepared publication, has been added to the system.

To share data, the system should be first equipped with it. Data entry and other operations allowing a modification of data, require user authentication by logging on to the system. New users, in order to get access to the full functionality of the system, need to register.

Data entry can be performed by two independent pathways:

- by predefined forms;
- with the usage of software able to read the files in the BibTeX format and storing information in the system in the appropriate way.

The usage of predefined forms allows registered users to introduce new data into the system individually. If one does not want to do this action individually or intends to enter a large number of new data, he or she can send the data to the system administrator and then administrator with the use of appropriate software will enter the received data into the system. The advantage of the individual data input by users is that they are assigned to them. In such case the users are authorized to edit the data in the future. This possibility is available only for registered users, in order to avoid entering incorrect information into the system. System with published descriptions (data) provides various options of searching.

Searching for information on the RSDS system was implemented so far in two main ways:

– Alphabetical search by certain keywords, such as titles of publications, their authors, editors, conference names, magazines, or year;
– Advanced search based on specified criteria, which sought description of publication has to fulfill.

Each of the currently available options of finding information in the RSDS system has both positive and negative aspects. Alphabetical search works when you know for example: the author of sought publication, the name of the journal in which the publication was published, who published a publication, or when one knows the year of the publication. The weaker part of this search method, however, is that in the absence of precise information about sought publication the system provides the large number of publication descriptions meeting the search criteria, which often have to be further analyzed by a painstaking selection process. However, during advanced search, the user defines the criteria which have to be fulfilled for the sought publication and, depending on the accuracy of the selection of these criteria, he or she obtains more or less adequate results. The problem of the further selection of obtained results still exists in many cases. This process involves directing a user's query to the database system to find the matching (appropriate) data (publication) for your search pattern. This matching is based on the finding of exact pattern in the data in the database. If the matching to the pattern data is found, it will be annexed to the result. This is repeated for all data located in the database. Then the edited result set (publication) is sent to the user (see Figure 3).

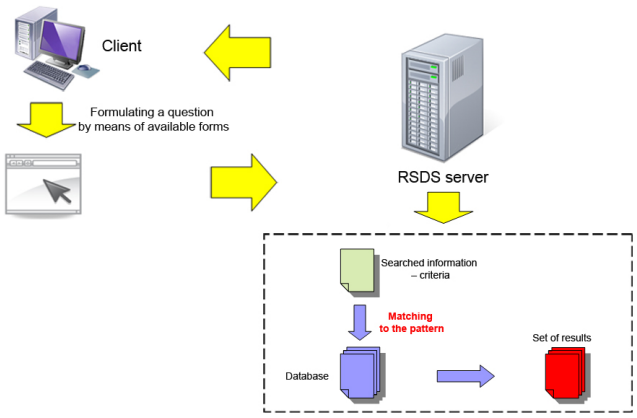


Fig. 3. The current course of the process of searching for the information in the RSDS system.

Finding information in the system process supplemented with the additional knowledge (using ontology and methods of rough set theory) is presented in the general form shown in Figure 4. This process in comparison with the mapping

specified search pattern is different in this that the system after receiving the user's query accede to its confrontation with the information included in the system [1–3, 17, 21]. This is done as follows: from the resources of the system are retrieved the specific ontologies for publication and they are verified for their belonging to the inner circle of the sought information in the domain of general ontology (taking into account the relationships between different types of concept). In the case of positive verification the approximations are appointed for a specific publication. They are then aggregated into a single value representing the total adjustment to the published information. When appointed value is greater than the threshold value, the given publication will be included in the result set. This process is repeated for all the data (publications). Next, after the result set processing (division of publication into groups) it is sent to the user. The developed methodology has been implemented in the system and made available to the users in the ontological search section. In order to streamline the process of building queries, for the system has been prepared the editor to assist the creation of query-by-user. The functionalities of this editor are: auto completion of entered concepts, defined relational operators such as AND, OR, NOT, and taking into account the priority of the process by using parentheses.

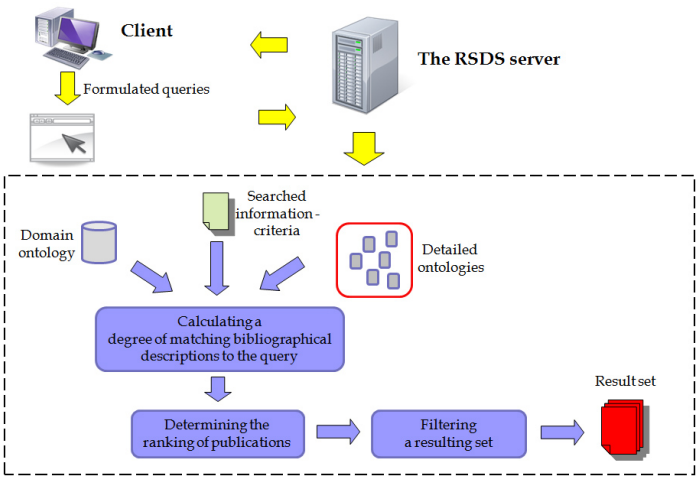


Fig. 4. The process of searching information in the RSDS system based on additional knowledge.

In order to minimize the time complexity of the process, the determining of the detailed ontology is carried out once. Determination of the detailed ontology is a part of the process relying on the preparation for the implementation of the system of bibliographic information for retrieval based on domain knowledge. This process can be carried out in several steps, which also has been shown (see Figure 5). The first step is to develop the general ontology by the domain

expert. Then, on the basis of the system resources, supported by the domain knowledge (general ontology) the detailed ontologies are determined. In the process of the detailed ontology generation new concepts or relationships between concepts can occur. In that case the system in cooperation with the domain expert is able to include such concepts in the general ontology, thus creating extended domain ontology. The final step in this process is to determine the degrees of bibliographical descriptions match to the elementary concepts from the general ontology. This is done because of the minimization of time required for determining the response to a query from user.

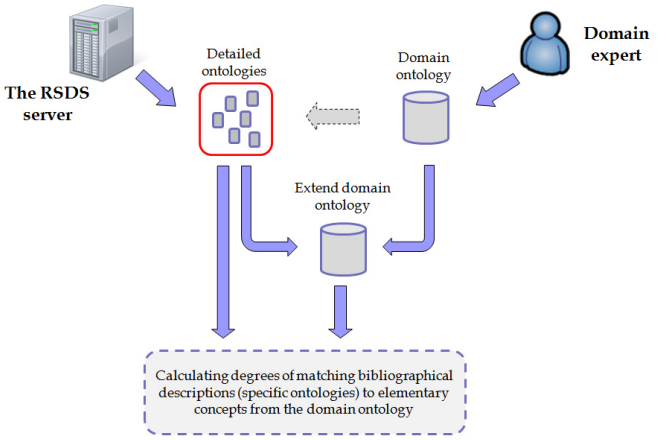


Fig. 5. The process of preparing the bibliographic system for searching information based on additional knowledge.

The system, in addition to bibliographical data (represented in the form of descriptive or BibTeX format) provides a fairly wide range of different statistics and the results of the analyzes of the data [12, 13, 16, 18].

In the carried out research the data collected from the year 1981 to the present are taken into account. They are analyzed in two ways: statistical and graphic. In relation to statistic data are processed in different compartments of time:

1. till the defined periods, five-years in the incremental relation, i.e., 1981-1985, 1981-1990, 1981-1995, etc.
2. in certain five-years, i.e., 1981-1985, 1986-1990, 1991-1995, etc.

In terms of graphic analysis it is made on the basis of the defined cooperation CG graph. The graph vertices are the authors of the publications included in the bibliographic system. Two vertices are connected by the edge when two authors have written at least one common publication.

In statistical analysis were determined the following values characterizing the examined data set, i.e., the number of authors in respect to the number of

written publication, various kinds of means, standard deviations associated with a particular medium, the number of works in respect to the number of authors creating them.

Graph analysis of the defined CG graph lies in its overall analysis that is the appointment of the average degree of vertex, of isolated vertices, etc. CG graph after the rejection of isolated vertices has been further analyzed. It was based on the determination of the components of the graph and the analysis of the largest component. Determination of components allowed on determining the groups of authors writing joint publications - the cooperating authors. These groups are reflected in reality. The analyzed parameters of the largest component allows for its accurate interpretation.

Additionally, the system allows users to read the Pawlak number of the first and second kind, which values indicate the strength of the proximity of publishing author's work with prof. Z. Pawlak, i.e., the less value of the Pawlak number represents the stronger relation between the author of published work and Professor Pawlak [18, 20].

All analyzes are performed dynamically, i.e., the calculation of parameters is taking into account any change in the data collected in system.

4 Input-output Data

Bibliographical descriptions are described in the system according to specifications of BibTeX [22]. This means that the description of each publication is divided into elements defined by BibTeX, such as title, publisher, year of publication, the keywords, abstract, etc. The prepared descriptions are placed in a relational database. Each component is stored in the database structure defined as a string, the importance of which, unfortunately, neither database nor database languages can understand. An example of a bibliographic description located in the system is presented in Table 1. Descriptions of publications are formulated in English.

Table 1. The example of bibliographic description in the BibTeX format.

@INPROCEEDINGS{,
author = {Hu, Xiaohua Tony and Cercone, Nick},
title = {Mining knowledge rules from databases: A rough set approach},
booktitle = {Proceedings of the 12th International Conference on Data Engineering},
conference = {International Conference on Data Engineering (CDE), New Orleans, USA},
pages = {96-105},
publisher = {IEEE Computer Society Press},
address = {Los Alamitos, CA, USA},
month = {February},
year = 1996,
isbn = {0-8186-7240-4},
abstract = {In this paper, the principle and experimental results of an attribute-oriented rough set approach for knowledge discovery in databases are described. ...},
keywords = {knowledge mining and discovery},
}

5 System Requirements

The RSDS system can be run on any computer that is connected to the Internet. The computer must have an operating system equipped with web browser. The presented above requirements must be met, as the RSDS system is an online system and requires a permanent connection to the Internet. In addition, a web browser, in which the system will be running, must support JavaScript scripting language, CSS style sheets and cookies. The system has been tested with the following browsers: Internet Explorer 9, Mozilla Firefox 17, and Chrome 23.

6 Plans for the Future

The directions of further research and work related to the system will be:

- development of a method for the formal verification of the correctness of the defined relations in the general ontology,
- increase of the degree of efficiency of information retrieval,
- automation of the process of processing of the owned information,
- attempt to improve the quality of semantic analysis,
- development of new functionalities of the system increasing its features such as: automatic discovery of scientific user profile, finding new data from Internet resources, extending the analysis of owned data, etc.

Acknowledgment. We would like to thank everyone who contributed to the creation and development of the RSDS system, in particular to Grzegorz Świstak and Przemysław Wanat.

References

1. Grochowalski P. and Pancerz K., The outline of an ontology for the rough set theory and its applications. In Czaja L., Penczek W., Salwicki A., Schlingloff H., Skowron A., Suraj Z., Lindemann G., Burkhard H.-D. (Eds.), *Proceedings of the Workshop on Concurrency, Specification and Programming, CS&P2008*, Gross-Vater See, Berlin, 29 September - 1 October, 2008, vol. 1-3, pp. 192–204. Informatik-Berichte, 2008.
2. Grochowalski P. and Pancerz K., The outline of an ontology for the rough set theory and its applications. *Fundamenta Informaticae*, 93(1-3):143–154, 2009.
3. Pancerz K. and Grochowalski P., Matching Ontological Subgraphs to Concepts: a Preliminary Rough Set Approach. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA'2010)*, Cairo, Egypt, November 29 - December 1, 2010, pp. 1394–1399, IEEE Xplore, 2010.
4. Pawlak Z., Rough Sets. *International Journal of Computer and Information Sciences*, 11:341–356, 1982.
5. Pawlak Z., Grzymała-Busse J.W., Słowiński R., and Ziarko W., Rough Sets. *Communications of the ACM*, 38(11):88–95, November 1995.
6. Pawlak Z. and Skowron A., Rough Sets and Boolean Reasoning. *Information Sciences*, 177(1):41–73, 2007.

7. Polkowski L.T., *Rough Sets. Mathematical Foundations.* Advances in Soft Computing. Physica-Verlag, Heidelberg, 2002.
8. Skowron A. and Pal S.K. (Eds.), Special Volume: Rough Sets, Pattern Recognition and Data Mining, vol. 24(6), *Pattern Recognition Letters.* North Holland, 2003.
9. Suraj Z. and Grochowalski P., The Rough Sets Database System: An Overview. In Komorowski J., Grzymała-Busse J.W., Tsumoto S., Słowiński R. (Eds.), *Proceedings of the 4th International Conference on Rough Sets and Current Trends in Computing, RSCTC 2004*, Uppsala, Sweden, June 2004, vol. 3066, *Lecture Notes in Artificial Intelligent*, pp. 841–849, Springer-Verlag, 2004.
10. Suraj Z. and Grochowalski P., The Rough Set Database System: An Overview. Transactions on Rough Sets III, *Lecture Notes of Computer Sciences*, vol. 3400, Springer-Verlag, Berlin, pp. 190–201, 2005.
11. Suraj Z. and Grochowalski P., Functional extension of the RSDS system. In Hirano S., Inuiguchi M., Miyamoto S., Nguyen H.S., Słowiński R., Greco S., Hata Y. (Eds.), *Proceedings of the 5th International Conference on Rough Sets and Current Trends in Computing, RSCTC 2006*, Kobe, Japan, November 2006, vol. 4259, *Lecture Notes in Artificial Intelligent*, pp. 786–795. Springer-Verlag, 2006.
12. Suraj Z. and Grochowalski P., Patterns of collaborations in rough set research. In Gomez V., Bello R., Falcon R. (Eds.), *Proceedings of the International Symposium on Fuzzy and Rough Sets, ISFUROS 2006*, Santa Clara, Cuba, December 5-8, 2006, pp. 1–7.
13. Suraj Z. and Grochowalski P., Patterns of Collaborations in Rough Set Research. In Bello R., Falcon R., Pedrycz W., Kacprzyk J. (Eds.), Granular Computing: at the Junction of Fuzzy Sets and Rough Sets, *Studies in Fuzziness and Soft Computing*, vol. 224, Springer-Verlag, 2008, pp. 79–92.
14. Suraj Z. and Grochowalski P., The Rough Set Database System. Transactions on Rough Sets VIII, *Lecture Notes of Computer Sciences*, vol. 3400, Springer-Verlag, pp. 307–331, 2008.
15. Suraj Z., Grochowalski P., Garwol K., and Pancerz K., Toward intelligent searching the rough set database system: an ontological approach. In Szczuka M., Czaja L. (Eds.), *Proceedings of the CS&P'2009 Workshop*, Kraków-Przegorzały, 28-30 September, 2009, vol. 1-2, pp. 574–582, Warsaw University, 2009.
16. Suraj Z. and Grochowalski P., Some Comparative Analyses of Data in the RSDS System. In Yu J., Greco S., Lingras P., Wang G., Skowron A. (Eds.), *Proceedings of the 5th International Conference on Rough Sets and Knowledge Technology, RSKT 2010*, Beijing, China, October 15-17, 2010, *Lecture Notes in Artificial Intelligence*, vol. 6401, Springer-Verlag, pp. 8–15, 2010.
17. Suraj Z. and Grochowalski P., Toward intelligent searching the Rough Set Database System (RSDS): an ontological approach. *Fundamenta Informaticae*, 101(1-2):115–123, 2010.
18. Suraj Z., Grochowalski P., and Lew Ł., Pawlak Collaboration Graph and Its Properties. *Proceedings of the 13th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC'2011)*, Moscow, Russia, June 27-30, 2011.
19. Suraj Z. and Grochowalski P., RoSetOn: The Open Project for Ontology of Rough Sets and Related Fields. In J.T. Yao et al. (Eds.), *Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology, RSKT 2011*, Banff, Canada, October 9-12, 2011, *Lecture Notes in Computer Science*, vol. 6954, Springer-Verlag, pp. 414–419, 2011.

20. Suraj Z., Grochowalski P., and Lew Ł, Discovering Patterns of Collaboration in Rough Set Research: Statistical and Graph-Theoretical Approach. In J.T. Yao et al. (Eds.), *Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology, RSKT 2011*, Banff, Canada, October 9-12, 2011, *Lecture Notes in Computer Science*, vol. 6954, Springer-Verlag, pp. 238–247, 2011.
21. Suraj Z., Grochowalski P, and Pancerz K., Knowledge Representation and Automated Methods of Searching for Information in Bibliographical Data Bases: A Rough Set Approach. In Skowron A., Suraj Z. (Eds.), Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak In Memoriam, *Intelligent Systems Reference Library*, vol. 43, pp. 515–538, 2012.
22. BibTeX. Available: *http://www.bibtex.org/*

Generation of Labelled Transition Systems for Alvis Models Using Haskell Model Representation

Marcin Szpyrka, Piotr Matyasik, and Michał Wypych

AGH University of Science and Technology
Department of Applied Computer Science
Al. Mickiewicza 30, 30-059 Kraków, Poland
{mszpyrka,ptm,mwypych}@agh.edu.pl

Abstract. Alvis is a formal modelling language for concurrent systems with the following advantages: a graphical modelling language used to define interconnections among agents, a high level programming language used to define the behaviour of agents and the possibility of a formal model verification. An Alvis model semantics find expression in an LTS graph (*labelled transition system*). Execution of any language statement is expressed as a transition between formally defined states of such a model. An LTS graph is generated using Haskell representation of an Alvis model and user defined Haskell functions can be used to explore the graph. The paper deals with the problem of translation of an Alvis model into its Haskell representation and discusses possibilities of model verification with the so-called Haskell filtering functions.

Keywords: Alvis, formal verification, Haskell, labelled transition system

1 Introduction

Alvis [1], [2], [3] is a formal modelling language being developed at AGH-UST in Krakow, Department of Applied Computer Science. The main aim of the project is to provide a flexible modelling language for concurrent systems with possibilities of a formal models verification. Alvis combines advantages of high level programming languages with a graphical language for modelling interconnections between subsystems (called agents) of a concurrent system. States of a model and transitions among them are represented using a labelled transition system (LTS graph for short [4]) which is used to verify the model formally by using model checking techniques [5], [6], [7], [8]. Previous research on Alvis was focused on the untimed version of the language with α^0 system layer (multiprocessor environments) [1], [2]. Using this system layer makes Alvis an alternative for other formalisms as Petri nets [9], [10], [11], process algebras [12] etc., but main advantages of Alvis approach for systems modelling are: similarity of Alvis syntax and syntax of procedural languages, graphical language for modelling interconnections between agents and the method of models states description which is similar to information provided by software debuggers.

The scheme of the modelling and verification process with Alvis is shown in Fig. 1. From the users point of view, the process starts from designing a model using prototype modelling environment called *Alvis Editor*. The designed model is stored using XML

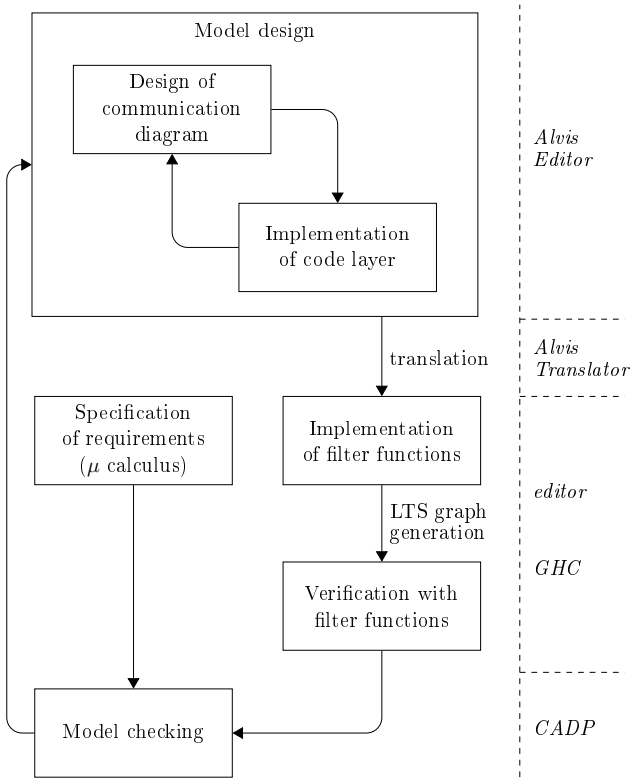


Fig. 1. Modelling and verification process with Alvis

file format. Then it is translated into Haskell [13] source code and its Haskell representation is used to generate the LTS graph. The Haskell functional language has been chosen as middle-stage representation of an Alvis model because Haskell is also used as a part of Alvis language i.e. Alvis uses Haskell to define parameters, data types and data manipulation functions. Haskell has been also used to implement the LTS graph generation algorithm. Such an LTS graph is stored as a Haskell list. A designer has access to such a source code, so user-defined Haskell functions (called *filtering functions*) that search an LTS graph for some states or parts of the graph that meet given requirements can be included into the model. The source code is compiled with GHC compiler. The results of received program execution are the LTS graph for the given model and the report of the model verification with filtering functions.

Another approach to Alvis model verification relies on using CADP toolbox [14]. CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking [5], [6], [7]. An Alvis LTS graph can be converted into BCG (Binary Coded Graphs) format which is one of acceptable input formats for CADP Toolbox. Then the CADP *evaluator* is used to check whether the model satisfies requirements given as regular alternation-free μ-calculus formulae [6], [15], [7].

The paper deals with the problem of translation of an Alvis model into its Haskell representation and discusses possibilities of model verification with filtering functions. Selected ideas connected with Alvis and LTS graphs are presented in Secion 2. Section 3 deals with the middle-stage Haskell model representation. Methods of LTS graph exploration are considered in Section 4. A short summary is given in the final section.

2 Alvis Models

An Alvis *model* is defined as a triple $\mathbf{A} = (H, B, \varphi)$, where H is a *hierarchical communication diagram*, B is a syntactically correct *code layer*, and φ is a *system layer*. In this paper we consider models with α^0 system layer only. This layer is based on the assumption that each active agent has access to its own processor and in case of conflicts agents priorities are taken under consideration. If two or more agents with the same highest priority compete for the same resources, the system works non-deterministically. Moreover, before generation of the Haskell model representation models are transformed into equivalent non-hierarchical form. Thus, from now on we will consider models defined as $\mathbf{A} = (D, B, \alpha^0)$, where $D = (\mathcal{A}, \mathcal{C}, \sigma)$ is a *non-hierarchical communication diagram*, where: $\mathcal{A} = \{X_1, \ldots, X_n\}$ is the set of *agents* consisting of two disjoint sets, $\mathcal{A}_A$, $\mathcal{A}_P$ such that $\mathcal{A} = \mathcal{A}_A \cup \mathcal{A}_P$, containing *active* and *passive* agents respectively; $\mathcal{C} \subseteq \mathcal{P} \times \mathcal{P}$, where $\mathcal{P}$ is the set of all ports, is the *communication relation*, such that:

- a connection cannot be defined between ports of the same agent;
- procedure ports are either input or output ones i.e. ports defined as procedures are used to transfer signals (values) either to or from a passive agent;
- a connection between an active and a passive agent must be a procedure call;
- a connection between two passive agents must be a procedure call from a non-procedure port.

The start function σ makes it possible to delay activation of some agents.

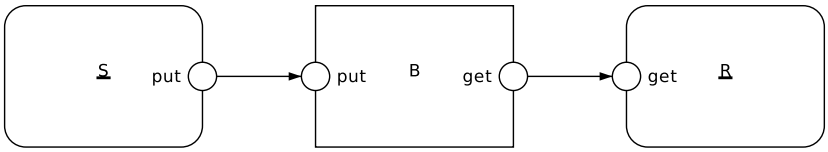
Active agents perform some activities and are similar to tasks in Ada programming language [16]. Each of them can be treated as a thread of control in a concurrent system. By contrast, *passive agents* do not perform any individual activities, and are similar to protected objects (shared variables). Passive agents provide other agents with a set of procedures (services). For more details see [2]. A description of the Alvis syntax can be also found at the Alvis project web site *http://fm.kis.agh.edu.pl*.

An example of Alvis model for a *sender-buffer-receiver* system is given in Fig. 2. Agent S (sender) puts sequentially valueless signals to the buffer (agent B) and agent R (receiver) gets such signals from the buffer. Agent B offers two procedures (services, ports) to connected agents.

States of an Alvis model and transitions among them are represented using a labelled transition system. An LTS graph is an ordered graph with nodes representing states of the considered system and edges representing transitions among states.

Definition 1. *A* Labelled Transition System *is a tuple* $\mathrm{LTS} = (S, A, \rightarrow, s_0)$*, where:*

- S *is the set of* states *and* $s_0 \in S$ *is the* initial state*;*
- A *is the set of* actions*;*
- $\rightarrow \subseteq S \times A \times S$ *is the* transition relation*.*




```
agent Sender {
  loop { out put; }}                          -- 1, 2

agent Buffer {
  i :: Int = 0;
  proc (i == 0) put { in put; i = 1; }        -- 1, 2
  proc (i /= 0) get { out get; i = 0; }}      -- 3, 4

agent Receiver {
  loop { in get; }}                           -- 1, 2
```

Fig. 2. Alvis model for *sender-buffer-receiver* system

The usage of LTS graphs is a universal method of a state space representation and is omnipresent in formal modelling languages. Different languages like Petri nets, time automata, process algebras etc. use different methods of describing nodes and edges in LTS graphs. They also use different names for them e.g. reachability graphs in Petri nets [9], [11], but the general structure of these graphs is still the same. The common feature of these approaches is the encoding of the considered system states using mathematical ideas typical for the chosen formalism. On the other hand they differ from methods used in programming languages significantly. In contrast to this, Alvis syntax is very similar to procedural programming languages and the used method of a model state description is similar to information provided by software debuggers. A state of an Alvis model is represented as a sequence of agents states [4], [2]. To describe the current state of an agent we use the following pieces of information.

- *Agent mode* (am) represents the type of the current agent activity e.g., *Running* (X) means that an agent is performing one of its statements, while *waiting* (W) means that the agent is waiting for an event (for active agents). For passive agents, *waiting* means that the corresponding agent is inactive and waits for another agent to call one of its accessible procedures. On the other hand, *Taken* (T) means that one of the passive agent procedures has been called and the agent is executing it.
- *Program counter* (pc) points out the current statement of an agent.
- *Context information list* (ci) contains additional information about the current state of an agent e.g. if an agent is in the *waiting* mode, ci contains information about events the agent is waiting for.
- *Parameters values list* contains the current values of the corresponding agent parameters.

LTS graph for model from Fig. 2 is shown in Fig. 3. Let us consider state 11:

- $am(B) = \mathsf{T}$, $pc(B) = 1$ – agent B is taken and performs its first step;

- $ci(S) = [proc(B.put, put)]$ – agent S has called procedure $B.put$ via port $S.put$;
- $am(R) = \mathsf{W}$, $pc(R) = 2$, $ci(R) = [in(get)]$ – agent R is waiting after performing step 2 (`out` statement), context information list points out that the agent is waiting for finalisation of the communication that has been initialised via port $R.get$.

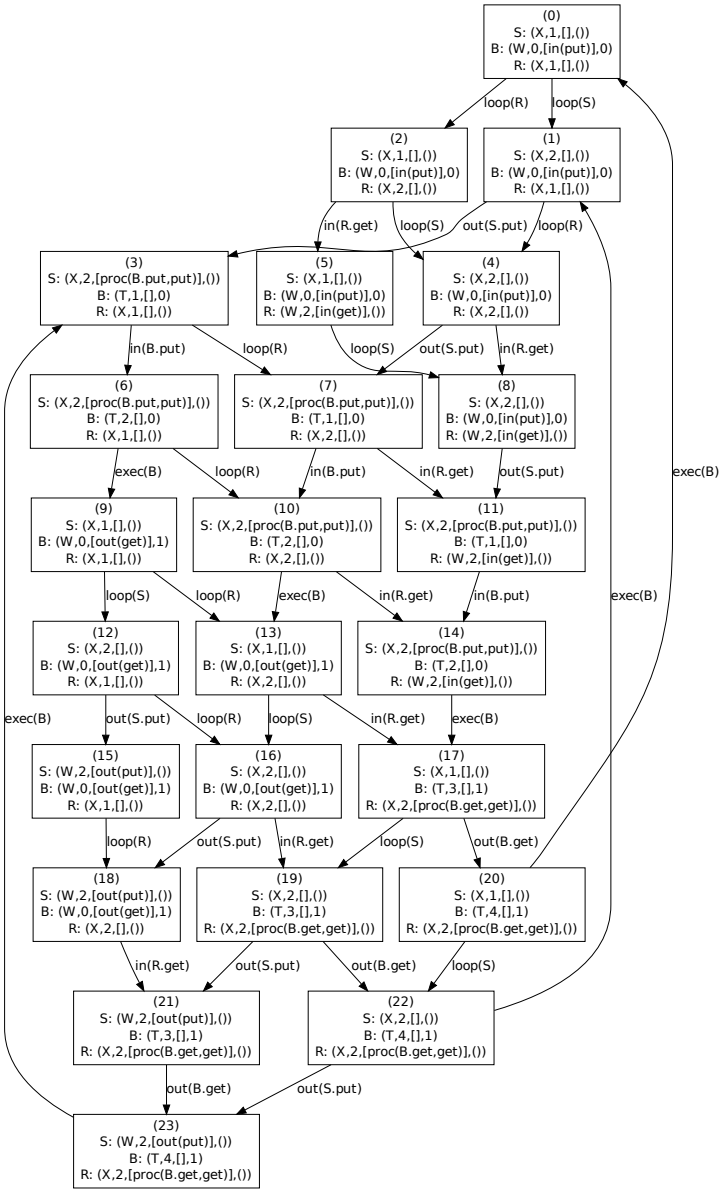


Fig. 3. LTS graph for model from Fig. 2

A state of a model can be changed as a result of executing a step. Most of the Alvis statements e.g. $exec$, $exit$, etc. are *single-step* statements. By contrast, if, $loop$ and $select$ are *multi-step* statements. We use recursion to count the number of steps for multi-step statements. For each of them the first step enters the statement interior. Then we count steps of statements put inside curly brackets. The set of all possible steps for the considered Alvis models contains the following elements:

- $exec$ – performs an evaluation and assignment;
- $exit$ – terminates an agent or a procedure,
- if – enters an if statement,
- in – performs communication (input side),
- $jump$ – jumps to a label,
- $loop$ – enters a loop,
- $null$ – performs an empty statement,
- out – performs communication (output side),
- $select$ – enters a select statement,
- $start$ – starts an inactive agent,
- io – performs communication (both sides).

Results of all these steps execution have been formally defined in [2].

3 Haskell Model Representation

An Alvis model is translated into Haskell source code and this middle-stage representation is used for LTS graph generation and for verification purposes. In order to obtain it following steps has to be performed:

- flattening Alvis hierarchical model,
- constructing agents list,
- generating state tuple for every agent,
- generating system state tuple by combining individual agents states, ordered respectively to agent list generated earlier,
- generating the `enable` function according to Alvis language rules [1], [2],
- generating the `fire` function,
- appending LTS generation code,
- appending LTS export code,
- appending main function.

Some elements common for all Alvis models are defined inside *Alvis* module, which is included into any model source file. This module contains for example enumerated data types for possible steps, entries of context information lists etc. Individual source files generated for models have the following structure:

1. User defined data types (if any).
2. User implemented functions for parameters manipulation (if any).
3. Definition of individual agents state types and the corresponding model state type.
4. Definition of the initial state.
5. Implementation of *enable* and *fire* functions.
6. Implementation of LTS graph generation algorithm.
7. Implementation of export functions into: text, *dot* and *aldebaran* formats.

8. User implemented filtering functions.
9. The *main* function.

The initial part of the Haskell source file for the model from Fig. 2 is shown in Fig. 4. It contains: the list of the model agents, data types for these agents' states and for the model state (type `State`), data type for LTS graph node representation (type `Node`) and the initial model state. The `Node` type contains: the node index, a model state and a list of enabled steps in that state (a step label and the target node number).

```
module Main where
import Alvis

agents = ["S", "B", "R"]

type SState = (Mode, Int, [ContentsInfo], ())
type BState = (Mode, Int, [ContentsInfo], (Int))
type RState = (Mode, Int, [ContentsInfo], ())

type State = (SState, BState, RState)
type Node = (Int, State, [(String, Int)])

s0 :: State
s0 = ((X,1,[],()), (W,0,[CIn "put"],(0)), (X,1,[],()))
```

Fig. 4. Part of Haskell source file for model from Fig. 2

The Haskell representation of an Alvis model behaviour is based on the so-called *enable-fire* approach which takes inspiration from Petri nets. The *enable* function takes a model state and an agent name and provides a list of the agent steps that are enabled (can be performed) in the state. The *fire* function takes an enabled step and a state and gives a new state that is the result of the step execution. A pseudo-code representation of the LTS graphs generation algorithm is shown in Fig. 5. It requires three elements to be given based on selected system layer, model structure and agents code. The first one is an initial state which can be straightforward extracted from system description. The second and third ones are *enable* and *fire* functions mentioned before.

The algorithm runs until it processes all elements form $nodeList$. This list after computation contains the resulting LTS. The $stateList$ is a helper list for quick check if a given state was already computed. For each state (line 5) algorithm computes a list of all enabled transitions (line 7). Afterwards it fires every transition and checks whether the resulting state has already been computed (line 11). Effectiveness of this step is crucial for computation time of the whole algorithm. If the state is present on $stateList$ a new transition is added to the currently checked state (line 14). Otherwise a new state is added to $nodeList$ and $stateList$ (lines 17-18) and a transition from currently investigated state to the new one is appended (line 16). A small part of the *enable* and *fire* functions generated for the considered example is given in Fig. 6.

1: $nodeList \leftarrow [(0, s_0, [])]$
2: $curIdx \leftarrow 0$
3: $stateList \leftarrow [(0, s_0)]$ ▷ for quick access to index of a given state
4: **while** $curIdx < \#nodeList$ **do**
5: $s \leftarrow nodeList[curIdx].state$
6: $tl \leftarrow nodeList[curIdx].transitions$
7: $transList \leftarrow$ transitions enabled in state s
8: **while** $transList$ not empty **do**
9: $trans \leftarrow$get and remove first element from $transList$
10: $state \leftarrow fire(trans, s)$
11: $i \leftarrow$ index of $state$ in $stateList$
12: $last \leftarrow \#nodeList - 1$ ▷ indexing from 0
13: **if** $\exists i$ **then** ▷ $state$ already on list
14: $nodeList[curIdx] \leftarrow (curIdx, s, tl +\!\!+ [t, i])$
15: **else** ▷ $state$ is new
16: $nodeList[curIdx] \leftarrow (curIdx, s, tl +\!\!+ [t, last + 1])$
17: $nodeList$ append $(last + 1, state, [])$
18: $stateList$ append $(last + 1, state)$
19: **end if**
20: **end while**
21: $curIdx \leftarrow curIdx + 1$
22: **end while**

Fig. 5. LTS graph generation algorithm

```
enable :: State -> String -> [TTransition]
enable ((am1,pc1,ci1,()),(am2,pc2,ci2,pv2),(am3,pc3,ci3,())) "S"
  | am1 == X && pc1 == 1 = [TLoop "S" 1]
  | am1 == X && pc1 == 2 && (procfree ci1) && am2 == W
  && elem (CIn "put") ci2  = [TOutAP "S.put" "B.put" 2]
  | am1 == X && pc1 == 2 && (procfree ci1) = [TOut "S.put" 2]
  | otherwise = []

fire :: TTransition -> State -> State
fire (TOutAP "S.put" "B.put" 2)
  ((am1,pc1,ci1,pv1),(am2,pc2,ci2,pv2),(am3,pc3,ci3,pv3))
  = ((am1,pc1,ci1 ++ [CProc "B.put,put"],pv1),(T,1,[],pv2),
    (am3,pc3,ci3,pv3))
```

Fig. 6. Part of the source code for *enable* and *fire* functions for model from Fig. 2

The generation of LTS graphs is the main aim of using the Haskell model representation. However, it should be underlined that the source file contains also functions for exporting LTS graphs into different formats. The most important one is the *aldebaran* format. LTS graphs stored in the *aldebaran* format can be automatically converted into BCG (Binary Coded Graphs) format which is one of the acceptable input formats for the CADP Toolbox. The conversion method is provided by one of CADP tools.

4 Model Verification with Filtering Functions

The Haskell approach to Alvis model verification requires Haskell programming skills, because the so-called *filtering functions* must be user-defined and included into the generated source file. Some of the functions are universal and can be included into any model, so it is possible to import them from an external Haskell module. However, most of these functions are based on the considered model `State` type and must be defined for a model individually.

```
deadState :: Node -> Bool
deadState (n,s,ls) = ls == []
-- filter deadState lts

singleOutState :: Node -> Bool
singleOutState (n,s,ls) = (length ls) == 1
-- filter singleOutState lts
```

Fig. 7. Examples of universal filtering functions

Examples of universal filtering functions are given in Fig. 7. The `deadState` function searches for states without outgoing arcs (dead states), while the `singleOutState` function searches for states with single outgoing arc. Included comments illustrate the usage of these functions.

```
sRunning :: Node -> Bool
sRunning (_,((X,_,_,_),_,_),_) = True
sRunning _ = False

twoWaiting :: Node -> Bool
twoWaiting (_,((W,_,_,_),(W,_,_,_),_),_) = True
twoWaiting (_,((W,_,_,_),_,(W,_,_,_)),_) = True
twoWaiting (_,(_,(W,_,_,_),(W,_,_,_)),_) = True
twoWaiting _ = False

procfree :: [ContentsInfo] -> Bool
procfree [] = True
procfree ((CProc _):_) = False
procfree (_:xs) = procfree xs

noProc :: Node -> Bool
noProc (_, ((_,_,ci1,_),_,(_,_,ci3,_)), _)
  | procfree ci1 && procfree ci3 = True
  | otherwise = False
```

Fig. 8. Examples of special filtering functions

These functions do not use the internal structure of the LTS graph node, thus can be used in any model. However, knowledge of the `State` type details is fundamental for implementing more sophisticated filtering functions. The main disadvantage of such functions is their adaptation to the given model. Examples of special filtering functions implemented for the model from Fig. 2 are shown in Fig. 8. The `sRunning` function searches for states with agent S in the *running* mode. Presented functions use the Haskell pattern matching mechanism. The underscore sign is a wild-card and its role changes depending on the place e.g. the first one replaces the number of a node, the second one – the program counter of agent S and the fifth – the state of agent B. The `twoWaiting` function searches for states with two agents in the *waiting* mode. The last `noProc` function searches for states when no procedure is executed. The auxiliary recursive `procfree` function searches a context information list for *proc* entries.

The functions presented so far are used to search for states which fulfil given filter condition. As shown in Fig. 7, they are used together with the standard `filter` function. More elaborated functions may search an LTS graph oneself. Example of such a function is given in Fig. 9. The `node2node` function returns pairs of nodes connected with an arc with the given label. It uses two auxiliary functions: `iNode` searches for a node with the given number and `endNodeNo` searches for the number of the end node for the given arc.

```
iNode :: Int -> [Node] -> Node
iNode i ((n,s,ls):ns)
  | i == n = (n,s,ls)
  | otherwise = iNode i ns

endNodeNo :: String -> [(String, Int)] -> Int
endNodeNo _ [] = -1
endNodeNo s ((a,i):ls)
  | s == a = i
  | otherwise = endNodeNo s ls

node2node :: String -> [Node] -> [Node] -> [(Node,Node)]
node2node _ _ [] = []
node2node label ltscopy ((n,s,ls):ns) =
  if k /= -1
  then ((n,s,ls),(iNode k ltscopy))
    : (node2node label ltscopy ns)
  else node2node label ltscopy ns
  where k = endNodeNo label ls
```

Fig. 9. Filtering function searching for parts of an LTS graph

5 Summary

Alvis is being developed to provide a simple tool for formal modelling and verification of concurrent systems. Compared to the most popular formalisms like Petri nets, process

algebras etc. its syntax is simple and very similar to procedural programming languages. The knowledge of all of the formal definitions presented in [2] is obsolete for the end user. From user's point of view, the most important are an Alvis model and its LTS graph generated for the model automatically.

The paper deals with the problem of LTS graphs generation for Alvis models. It has been solved using a middle-stage Haskell model representation. This approach has been chosen out of consideration for the usage of Haskell in Alvis models. The Haskell representation of an Alvis model is based on two functions called *enable* and *fire* that provide the list of transitions enabled in the given states and results of these transitions execution. The functions are used in the presented algorithm for LTS graphs generation. It should be emphasized that this *enable-fire* approach can be used for generation states spaces for other formalism. For example, it has been successfully used for XCCS process algebra [17], [18]. In this case, instead of Alvis 4-tuples string values have been used to represent states of individual agents (algebraic equations). Nevertheless, exactly the same Haskell implementation of the LTS generation algorithm has been used to generate LTS graphs. This stresses the flexibility of this approach. It is enough to adapt the *enable* and *fire* functions to a considered formalism and presented approach can be used for verification purposes.

The second advantage of the considered approach is the possibility of model verification using Haskell implemented algorithms (functions). The generated LTS graph is stored as a Haskell list, thus not only is it possible to translate it into the *aldebaran* format and use CADP toolbox to verify its properties, but also user defined Haskell functions can be used to explore an LTS graph. This Haskell based approach is a completion of the CADP based verification. Analysis of Alvis models can be realized using the CADP *evaluator* tool. In such approach, a specification of requirements is given as a set of μ-calculus formulas [6] and the tool is used to check whether the model LTS graph satisfies them. It should be emphasized that this is an action based approach. A μ-calculus formula concerns actions labels while states of considered model are represented using their numbers only. On the other hand, the Haskell approach is rather states oriented. We can use the Haskell pattern matching mechanism to filter states that fulfil given requirements. Moreover, the Haskell approach can be used to implement user defined verification algorithms that search for some specified parts of an LTS graph and are not provided by verification toolbox. For example, this is a good path to test user-defined non-standard verification procedures fast. Moreover, Haskell expressiveness allows to fit even quite complicated algorithms in a few lines of code as compared to imperative languages.

References

1. Szpyrka, M., Matyasik, P., Mrówka, R.: Alvis – modelling language for concurrent systems. In Bouvry, P., Gonzalez-Velez, H., Kołodziej, J., eds.: Intelligent Decision Systems in Large-Scale Distributed Environments. Volume 362 of Studies in Computational Intelligence. Springer-Verlag (2011) 315–341
2. Szpyrka, M., Matyasik, P., Mrówka, R., Kotulski, L.: Formal description of Alvis language with α^0 system layer. Fundamenta Informaticae (2013) (to appear).

3. Szpyrka, M., Matyasik, P., Wypych, M.: Alvis language with time dependence. In: Proceedings of the Federated Conference on Computer Science and Information Systems, Krakow, Poland (2013) 1607–1612
4. Kotulski, L., Szpyrka, M., Sędziwy, A.: Labelled transition system generation from Alvis language. In König, A., et al., eds.: Knowledge-Based and Intelligent Information and Engineering Systems – KES 2011. Volume 6881 of LNCS. Springer-Verlag (2011) 180–189
5. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, London, UK (2008)
6. Emerson, E.A.: Model checking and the Mu-calculus. In Immerman, N., Kolaitis, P.G., eds.: Descriptive Complexity and Finite Models. Volume 31 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society (1997) 185–214
7. Mateescu, R., Sighireanu, M.: Efficient on-the-fly model-checking for regular alternation-free μ-calculus. Technical Report 3899, INRIA (2000)
8. Penczek, W., Półrola, A.: Advances in Verification of Time Petri nets and Timed Automata. A Temporal Logic Approach. Volume 20 of Studies in Computational Intelligence. Springer-Verlag (2006)
9. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
10. Suraj, Z., Fryc, B.: Timed approximate Petri nets. Fundamenta Informaticae **71**(1) (2006) 83–99
11. Szpyrka, M.: Analysis of RTCP-nets with reachability graphs. Fundamenta Informaticae **74**(2–3) (2006) 375–390
12. Bergstra, J.A., Ponse, A., Smolka, S.A., eds.: Handbook of Process Algebra. Elsevier Science, Upper Saddle River, NJ, USA (2001)
13. O'Sullivan, B., Goerzen, J., Stewart, D.: Real World Haskell. O'Reilly Media, Sebastopol, USA (2008)
14. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2006: A toolbox for the construction and analysis of distributed processes. In Damm, W., Hermanns, H., eds.: Computer Aided Verification. Volume 4590 of LNCS., Springer-Verlag (2007) 158–163
15. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science **27**(3) (1983) 333–354
16. Barnes, J.: Programming in Ada 2005. Addison Wesley (2006)
17. Balicki, K., Szpyrka, M.: Tag abstraction for XCCS modelling language. In: Proceedings of the Concurrency Specification and Programming Workshop (CSP 2009). Volume 1., Krakow, Poland (September 28-30 2009) 26–37
18. Balicki, K., Szpyrka, M.: Formal definition of XCCS modelling language. Fundamenta Informaticae **93**(1-3) (2009) 1–15

Bisimulation-Based Concept Learning in Description Logics

Thanh-Luong Tran[1,3], Quang-Thuy Ha[2], Thi-Lan-Giao Hoang[1], Linh Anh Nguyen[3,2], and Hung Son Nguyen[3,2]

[1] Department of Information Technology,
Hue University of Sciences, 77 Nguyen Hue, Hue city, Vietnam
{ttluong,hlgiao}@hueuni.edu.vn
[2] Faculty of Information Technology,
VNU University of Engineering and Technology, 144 Xuan Thuy, Hanoi, Vietnam
thuyhq@vnu.edu.vn
[3] Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
{nguyen,son}@mimuw.edu.pl

Abstract. Concept learning in description logics (DLs) is similar to binary classification in traditional machine learning. The difference is that in DLs objects are described not only by attributes but also by binary relationships between objects. In this paper, we develop the first bisimulation-based method of concept learning in DLs for the following setting: given a knowledge base KB in a DL, a set of objects standing for positive examples and a set of objects standing for negative examples, learn a concept C in that DL such that the positive examples are instances of C w.r.t. KB, while the negative examples are not instances of C w.r.t. KB.

1 Introduction

In this paper we continue our study [12, 15, 7, 4] on concept learning in description logics (DLs). This problem is similar to binary classification in traditional machine learning. The difference is that in DLs objects are described not only by attributes but also by binary relationships between objects. The major settings of concept learning in DLs are as follows:

1. Given a knowledge base KB in a DL L and sets E^+, E^- of individuals, learn a concept C in L such that: (a) $KB \models C(a)$ for all $a \in E^+$, and (b) $KB \models \neg C(a)$ for all $a \in E^-$. The set E^+ (resp. E^-) contains positive (resp. negative) examples of C.
2. The second setting differs from the previous one only in that the condition (b) is replaced by the weaker one: $KB \not\models C(a)$ for all $a \in E^-$.
3. Given an interpretation $\mathcal{I}$ and sets E^+, E^- of individuals, learn a concept C in L such that: (a) $\mathcal{I} \models C(a)$ for all $a \in E^+$, and (b) $\mathcal{I} \models \neg C(a)$ for all $a \in E^-$. Note that $\mathcal{I} \not\models C(a)$ is the same as $\mathcal{I} \models \neg C(a)$.

As an early work on concept learning in DLs, Cohen and Hirsh [3] studied PAC-learnability of the CLASSIC description logic (an early DL formalism) and its sublogic C-CLASSIC. They proposed a concept learning algorithm based on "least common subsumers". In [9] Lambrix and Larocchia proposed a simple concept learning algorithm based on concept normalization.

Badea and Nienhuys-Cheng [1], Iannone et al. [8], Fanizzi et al. [6], Lehmann and Hitzler [10] studied concept learning in DLs by using refinement operators as in inductive logic programming. The works [1, 8] use the first mentioned setting, while the works [6, 10] use the second mentioned setting. Apart from refinement operators, scoring functions and search strategies also play important roles in algorithms proposed in those works. The algorithm DL-Learner [10] exploits genetic programming techniques, while DL-FOIL [6] considers also unlabeled data as in semi-supervised learning.

Nguyen and Szałas [12] applied bisimulation in DLs [5] to model indiscernibility of objects. Their work is pioneering in using bisimulation for concept learning in DLs. It also concerns concept approximation by using bisimulation and Pawlak's rough set theory [13, 14]. In [15] Tran et al. generalized and extended the concept learning method of [12] for DL-based information systems. They took attributes as basic elements of the language. An information system in a DL is a finite interpretation in that logic. Thus, both the works [12, 15] use the third mentioned setting. In [7] Ha et al. developed the first bisimulation-based method, called BBCL, for concept learning in DLs using the first mentioned setting. Their method uses models of KB and bisimulation in those models to guide the search for the concept to be learned. It is formulated for a large class of useful DLs, with well-known DLs like $\mathcal{ALC}$, $\mathcal{SHIQ}$, $\mathcal{SHOIQ}$, $\mathcal{SROIQ}$. The work [7] also introduced dual-BBCL, a variant of BBCL, for concept learning in DLs using the first mentioned setting.

In this paper, we develop the first bisimulation-based method, called BBCL2, for concept learning in DLs using the second mentioned setting, i.e., for learning a concept C such that: $KB \models C(a)$ for all $a \in E^+$, and $KB \not\models C(a)$ for all $a \in E^-$, where KB is a given knowledge base in the considered DL, and E^+, E^- are given sets of examples of C. This method is based on the dual-BBCL method (of concept learning in DLs using the first mentioned setting) from our joint work [7]. We make appropriate changes for dealing with the condition "$KB \not\models C(a)$ for all $a \in E^-$" instead of "$KB \models \neg C(a)$ for all $a \in E^-$".

The rest of this paper is structured as follows. In Section 2, we recall notation and semantics of DLs. We present our BBCL2 method in Section 3 and illustrate it by examples in Section 4. We conclude in Section 5. Due to the lack of space, we will not recall the notion of bisimulation in DLs [5, 7], but just mention the use of the largest auto-bisimulation relations and list the bisimulation-based selectors [7].

2 Notation and Semantics of Description Logics

A *DL-signature* is a finite set $\Sigma = \Sigma_I \cup \Sigma_{dA} \cup \Sigma_{nA} \cup \Sigma_{oR} \cup \Sigma_{dR}$, where Σ_I is a set of *individuals*, Σ_{dA} is a set of *discrete attributes*, Σ_{nA} is a set of *numeric attributes*, Σ_{oR} is a set of *object role names*, and Σ_{dR} is a set of *data roles*.[4] All the sets Σ_I, Σ_{dA}, Σ_{nA}, Σ_{oR}, Σ_{dR} are pairwise disjoint.

Let $\Sigma_A = \Sigma_{dA} \cup \Sigma_{nA}$. Each attribute $A \in \Sigma_A$ has a domain $dom(A)$, which is a non-empty set that is countable if A is discrete, and partially ordered by $\leq$ otherwise.[5] (For simplicity we do not subscript $\leq$ by A.) A discrete attribute is a *Boolean attribute* if $dom(A) = \{\mathsf{true}, \mathsf{false}\}$. We refer to Boolean attributes also as *concept names*. Let $\Sigma_C \subseteq \Sigma_{dA}$ be the set of all concept names of Σ.

An object role name stands for a binary predicate between individuals. A data role σ stands for a binary predicate relating individuals to elements of a set $range(\sigma)$.

We denote individuals by letters like a and b, attributes by letters like A and B, object role names by letters like r and s, data roles by letters like σ and ϱ, and elements of sets of the form $dom(A)$ or $range(\sigma)$ by letters like c and d.

We will consider some (additional) *DL-features* denoted by I (*inverse*), O (*nominal*), F (*functionality*), N (*unquantified number restriction*), Q (*quantified number restriction*), U (*universal role*), Self (*local reflexivity of an object role*). A *set of DL-features* is a set consisting of some or zero of these names.

Let Σ be a DL-signature and Φ be a set of DL-features. Let $\mathcal{L}$ stand for $\mathcal{ALC}$, which is the name of a basic DL. (We treat $\mathcal{L}$ as a language, but not a logic.) The DL language $\mathcal{L}_{\Sigma,\Phi}$ allows *object roles* and *concepts* defined as follows:

- if $r \in \Sigma_{oR}$ then r is an object role of $\mathcal{L}_{\Sigma,\Phi}$
- if $A \in \Sigma_C$ then A is concept of $\mathcal{L}_{\Sigma,\Phi}$
- if $A \in \Sigma_A \setminus \Sigma_C$ and $d \in dom(A)$ then $A = d$ and $A \neq d$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
- if $A \in \Sigma_{nA}$ and $d \in dom(A)$ then $A \leq d$, $A < d$, $A \geq d$ and $A > d$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
- if C and D are concepts of $\mathcal{L}_{\Sigma,\Phi}$, R is an object role of $\mathcal{L}_{\Sigma,\Phi}$, $r \in \Sigma_{oR}$, $\sigma \in \Sigma_{dR}$, $a \in \Sigma_I$, and n is a natural number then
  - $\top$, $\bot$, $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$ and $\exists R.C$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
  - if $d \in range(\sigma)$ then $\exists \sigma.\{d\}$ is a concept of $\mathcal{L}_{\Sigma,\Phi}$
  - if $I \in \Phi$ then r^- is an object role of $\mathcal{L}_{\Sigma,\Phi}$
  - if $O \in \Phi$ then $\{a\}$ is a concept of $\mathcal{L}_{\Sigma,\Phi}$
  - if $F \in \Phi$ then $\leq 1\, r$ is a concept of $\mathcal{L}_{\Sigma,\Phi}$
  - if $\{F, I\} \subseteq \Phi$ then $\leq 1\, r^-$ is a concept of $\mathcal{L}_{\Sigma,\Phi}$
  - if $N \in \Phi$ then $\geq n\, r$ and $\leq n\, r$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
  - if $\{N, I\} \subseteq \Phi$ then $\geq n\, r^-$ and $\leq n\, r^-$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
  - if $Q \in \Phi$ then $\geq n\, r.C$ and $\leq n\, r.C$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
  - if $\{Q, I\} \subseteq \Phi$ then $\geq n\, r^-.C$ and $\leq n\, r^-.C$ are concepts of $\mathcal{L}_{\Sigma,\Phi}$
  - if $U \in \Phi$ then U is an object role of $\mathcal{L}_{\Sigma,\Phi}$

[4] Object role names are atomic object roles.

[5] One can assume that, if A is a numeric attribute, then $dom(A)$ is the set of real numbers and $\leq$ is the usual linear order between real numbers.

$$(r^-)^{\mathcal{I}} = (r^{\mathcal{I}})^{-1} \qquad U^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \qquad \top^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad \bot^{\mathcal{I}} = \emptyset$$
$$(A = d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(x) = d\} \qquad (A \neq d)^{\mathcal{I}} = (\neg(A = d))^{\mathcal{I}}$$
$$(A \leq d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(x) \text{ is defined}, A^{\mathcal{I}}(x) \leq d\}$$
$$(A \geq d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(x) \text{ is defined}, d \leq A^{\mathcal{I}}(x)\}$$
$$(A < d)^{\mathcal{I}} = ((A \leq d) \sqcap (A \neq d))^{\mathcal{I}} \qquad (A > d)^{\mathcal{I}} = ((A \geq d) \sqcap (A \neq d))^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \qquad (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\} \quad (\exists r.\mathsf{Self})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid r^{\mathcal{I}}(x, x)\} \quad (\exists \sigma.\{d\})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sigma^{\mathcal{I}}(x, d)\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\, [R^{\mathcal{I}}(x, y) \Rightarrow C^{\mathcal{I}}(y)]\}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\, [R^{\mathcal{I}}(x, y) \wedge C^{\mathcal{I}}(y)]$$
$$(\geq n\, R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid R^{\mathcal{I}}(x, y) \wedge C^{\mathcal{I}}(y)\} \geq n\}$$
$$(\leq n\, R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid R^{\mathcal{I}}(x, y) \wedge C^{\mathcal{I}}(y)\} \leq n\}$$
$$(\geq n\, R)^{\mathcal{I}} = (\geq n\, R.\top)^{\mathcal{I}} \qquad (\leq n\, R)^{\mathcal{I}} = (\leq n\, R.\top)^{\mathcal{I}}$$

Fig. 1. Interpretation of complex object roles and complex concepts.

- if $\mathsf{Self} \in \Phi$ then $\exists r.\mathsf{Self}$ is a concept of $\mathcal{L}_{\Sigma,\Phi}$.

An *interpretation* in $\mathcal{L}_{\Sigma,\Phi}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain* of $\mathcal{I}$ and $\cdot^{\mathcal{I}}$ is a mapping called the *interpretation function* of $\mathcal{I}$ that associates each individual $a \in \Sigma_I$ with an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each concept name $A \in \Sigma_C$ with a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each attribute $A \in \Sigma_A \setminus \Sigma_C$ with a partial function $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \mathit{dom}(A)$, each object role name $r \in \Sigma_{oR}$ with a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each data role $\sigma \in \Sigma_{dR}$ with a binary relation $\sigma^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathit{range}(\sigma)$. The interpretation function $\cdot^{\mathcal{I}}$ is extended to complex object roles and complex concepts as shown in Figure 1, where $\#\Gamma$ stands for the cardinality of the set Γ.

Given an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ in $\mathcal{L}_{\Sigma,\Phi}$, we say that an object $x \in \Delta^{\mathcal{I}}$ has *depth* k if k is the maximal natural number such that there are pairwise different objects $x_0, \ldots, x_k$ of $\Delta^{\mathcal{I}}$ with the properties that:

- $x_k = x$ and $x_0 = a^{\mathcal{I}}$ for some $a \in \Sigma_I$,
- $x_i \neq b^{\mathcal{I}}$ for all $1 \leq i \leq k$ and all $b \in \Sigma_I$,
- for each $1 \leq i \leq k$, there exists an object role R_i of $\mathcal{L}_{\Sigma,\Phi}$ such that $\langle x_{i-1}, x_i \rangle \in R_i^{\mathcal{I}}$.

By $\mathcal{I}_{|k}$ we denote the interpretation obtained from $\mathcal{I}$ by restricting the domain to the set of objects with depth not greater than k and restricting the interpretation function accordingly.

A *role inclusion axiom* in $\mathcal{L}_{\Sigma,\Phi}$ is an expression of the form $R_1 \circ \ldots \circ R_k \sqsubseteq r$, where $k \geq 1$, $r \in \Sigma_{oR}$ and $R_1, \ldots, R_k$ are object roles of $\mathcal{L}_{\Sigma,\Phi}$ different from U. A *role assertion* in $\mathcal{L}_{\Sigma,\Phi}$ is an expression of the form $\mathtt{Ref}(r)$, $\mathtt{Irr}(r)$, $\mathtt{Sym}(r)$,

$\texttt{Tra}(r)$, or $\texttt{Dis}(R, S)$, where $r \in \Sigma_{oR}$ and R, S are object roles of $\mathcal{L}_{\Sigma,\Phi}$ different from U. Given an interpretation $\mathcal{I}$, define that:

$$\begin{array}{ll}
\mathcal{I} \models R_1 \circ \ldots \circ R_k \sqsubseteq r & \text{if } R_1^{\mathcal{I}} \circ \ldots \circ R_k^{\mathcal{I}} \subseteq r^{\mathcal{I}} \\
\mathcal{I} \models \texttt{Ref}(r) & \text{if } r^{\mathcal{I}} \text{ is reflexive} \\
\mathcal{I} \models \texttt{Irr}(r) & \text{if } r^{\mathcal{I}} \text{ is irreflexive} \\
\mathcal{I} \models \texttt{Sym}(r) & \text{if } r^{\mathcal{I}} \text{ is symmetric} \\
\mathcal{I} \models \texttt{Tra}(r) & \text{if } r^{\mathcal{I}} \text{ is transitive} \\
\mathcal{I} \models \texttt{Dis}(R, S) & \text{if } R^{\mathcal{I}} \text{ and } S^{\mathcal{I}} \text{ are disjoint,}
\end{array}$$

where the operator $\circ$ stands for the composition of binary relations. By a *role axiom* in $\mathcal{L}_{\Sigma,\Phi}$ we mean either a role inclusion axiom or a role assertion in $\mathcal{L}_{\Sigma,\Phi}$. We say that a role axiom φ is *valid* in $\mathcal{I}$ (or $\mathcal{I}$ *validates* φ) if $\mathcal{I} \models \varphi$.

A *terminological axiom* in $\mathcal{L}_{\Sigma,\Phi}$, also called a *general concept inclusion* (GCI) in $\mathcal{L}_{\Sigma,\Phi}$, is an expression of the form $C \sqsubseteq D$, where C and D are concepts in $\mathcal{L}_{\Sigma,\Phi}$. An interpretation $\mathcal{I}$ *validates* an axiom $C \sqsubseteq D$, denoted by $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

An *individual assertion* in $\mathcal{L}_{\Sigma,\Phi}$ is an expression of one of the forms $C(a)$ (*concept assertion*), $r(a, b)$ (*positive role assertion*), $\neg r(a, b)$ (*negative role assertion*), $a = b$, and $a \neq b$, where $r \in \Sigma_{oR}$ and C is a concept of $\mathcal{L}_{\Sigma,\Phi}$. We will write, for example, $A(a) = d$ instead $(A = d)(a)$. Given an interpretation $\mathcal{I}$, define that:

$$\begin{array}{ll}
\mathcal{I} \models a = b & \text{if } a^{\mathcal{I}} = b^{\mathcal{I}} \\
\mathcal{I} \models a \neq b & \text{if } a^{\mathcal{I}} \neq b^{\mathcal{I}} \\
\mathcal{I} \models C(a) & \text{if } a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models r(a, b) & \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}} \\
\mathcal{I} \models \neg r(a, b) & \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin r^{\mathcal{I}}.
\end{array}$$

We say that $\mathcal{I}$ *validates* an individual assertion φ if $\mathcal{I} \models \varphi$.

An *RBox* (resp. *TBox*, *ABox*) in $\mathcal{L}_{\Sigma,\Phi}$ is a finite set of role axioms (resp. terminological axioms, individual assertions) in $\mathcal{L}_{\Sigma,\Phi}$. A *knowledge base* in $\mathcal{L}_{\Sigma,\Phi}$ is a triple $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{R}$ (resp. $\mathcal{T}$, $\mathcal{A}$) is an RBox (resp. a TBox, an ABox) in $\mathcal{L}_{\Sigma,\Phi}$. An interpretation $\mathcal{I}$ is a *model* of a "box" if it validates all the axioms/assertions of that "box". It is a *model* of a knowledge base $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ if it is a model of $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$. A knowledge base is *satisfiable* if it has a model. An individual a is said to be an *instance* of a concept C w.r.t. a knowledge base KB, denoted by $KB \models C(a)$, if, for every model $\mathcal{I}$ of KB, $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

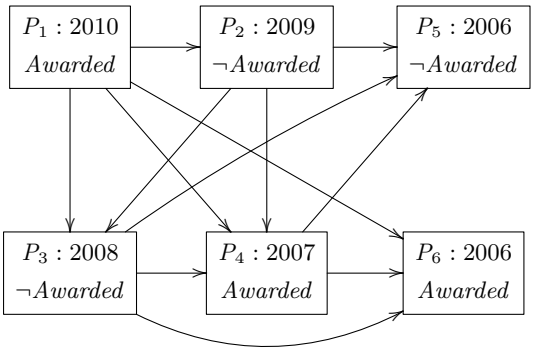


Fig. 2. An illustration for the knowledge base given in Example 2.1

Example 2.1. This example is based on an example of [15, 7]. Let

$$\Phi = \{I, O, N, Q\}, \quad \Sigma_I = \{P_1, P_2, P_3, P_4, P_5, P_6\}, \quad \Sigma_C = \{Pub, Awarded, A_d\},$$
$$\Sigma_{dA} = \Sigma_C, \quad \Sigma_{nA} = \{Year\}, \quad \Sigma_{oR} = \{cites, cited\_by\}, \quad \Sigma_{dR} = \emptyset,$$
$$\mathcal{R} = \{cites^- \sqsubseteq cited\_by, \quad cited\_by^- \sqsubseteq cites\}, \quad \mathcal{T} = \{\top \sqsubseteq Pub\},$$
$$\mathcal{A}_0 = \{Awarded(P_1), \neg Awarded(P_2), \neg Awarded(P_3), Awarded(P_4),$$
$$\neg Awarded(P_5), Awarded(P_6), Year(P_1) = 2010, Year(P_2) = 2009,$$
$$Year(P_3) = 2008, Year(P_4) = 2007, Year(P_5) = 2006, Year(P_6) = 2006,$$
$$cites(P_1, P_2), cites(P_1, P_3), cites(P_1, P_4), cites(P_1, P_6),$$
$$cites(P_2, P_3), cites(P_2, P_4), cites(P_2, P_5), cites(P_3, P_4),$$
$$cites(P_3, P_5), cites(P_3, P_6), cites(P_4, P_5), cites(P_4, P_6)\},$$

where the concept Pub stands for "publication". Then $KB_0 = \langle \mathcal{R}, \mathcal{T}, \mathcal{A}_0 \rangle$ is a knowledge base in $\mathcal{L}_{\Sigma,\Phi}$. The axiom $\top \sqsubseteq Pub$ states that the domain of any model of KB_0 consists of only publications. The knowledge base KB_0 is illustrated in Figure 2 (on page 426). In this figure, nodes denote publications and edges denote citations (i.e., assertions of the role $cites$), and we display only information concerning assertions about $Year$, $Awarded$ and $cites$. ◁

An $\mathcal{L}_{\Sigma,\Phi}$ *logic* is specified by a number of restrictions adopted for the language $\mathcal{L}_{\Sigma,\Phi}$. We say that a logic L is *decidable* if the problem of checking satisfiability of a given knowledge base in L is decidable. A logic L has the *finite model property* if every satisfiable knowledge base in L has a finite model. We say that a logic L has the *semi-finite model property* if every satisfiable knowledge base in L has a model $\mathcal{I}$ such that, for any natural number k, $\mathcal{I}_{|k}$ is finite and constructable.

As the general satisfiability problem of context-free grammar logics is undecidable [2], the most general $\mathcal{L}_{\Sigma,\Phi}$ logics (without restrictions) are also undecidable. The considered class of DLs contains, however, many decidable and useful logics. One of them is $\mathcal{SROIQ}$ - the logical base of the Web Ontology Language OWL 2. This logic has the semi-finite model property.

3 Concept Learning for Knowledge Bases in DLs

Let L be a decidable $\mathcal{L}_{\Sigma,\Phi}$ logic with the semi-finite model property, $A_d \in \Sigma_C$ be a special concept name standing for the "decision attribute", and $KB_0 = \langle \mathcal{R}, \mathcal{T}, \mathcal{A}_0 \rangle$ be a knowledge base in L without using A_d. Let E^+ and E^- be disjoint subsets of Σ_I such that the knowledge base $KB = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ with $\mathcal{A} = \mathcal{A}_0 \cup \{A_d(a) \mid a \in E^+\} \cup \{\neg A_d(a) \mid a \in E^-\}$ is satisfiable. The set E^+ (resp. E^-) is called the set of *positive* (resp. *negative*) *examples* of A_d. Let $E = \langle E^+, E^- \rangle$.

The problem is to learn a concept C as a definition of A_d in the logic L restricted to a given sublanguage $\mathcal{L}_{\Sigma^\dagger,\Phi^\dagger}$ with $\Sigma^\dagger \subseteq \Sigma \setminus \{A_d\}$ and $\Phi^\dagger \subseteq \Phi$ such that: $KB \models C(a)$ for all $a \in E^+$, and $KB \not\models C(a)$ for all $a \in E^-$.

Given an interpretation $\mathcal{I}$ in $\mathcal{L}_{\Sigma,\Phi}$, by $\equiv_{\Sigma^\dagger,\Phi^\dagger,\mathcal{I}}$ we denote the equivalence relation on $\Delta^\mathcal{I}$ with the property that $x \equiv_{\Sigma^\dagger,\Phi^\dagger,\mathcal{I}} x'$ iff x is $\mathcal{L}_{\Sigma^\dagger,\Phi^\dagger}$-equivalent to x' (i.e., for every concept D of $\mathcal{L}_{\Sigma^\dagger,\Phi^\dagger}$, $x \in D^\mathcal{I}$ iff $x' \in D^\mathcal{I}$). By [7, Theorem 3], this equivalence relation coincides with the largest $\mathcal{L}_{\Sigma^\dagger,\Phi^\dagger}$-auto-bisimulation $\sim_{\Sigma^\dagger,\Phi^\dagger,\mathcal{I}}$ of $\mathcal{I}$ (see [7] for the definition of this notion).

Let $\mathcal{I}$ be an interpretation. We say that a set $Y \subseteq \Delta^\mathcal{I}$ is *divided* by E if there exist $a \in E^+$ and $b \in E^-$ such that $\{a^\mathcal{I}, b^\mathcal{I}\} \subseteq Y$. A partition $P = \{Y_1, \ldots, Y_k\}$ of $\Delta^\mathcal{I}$ is said to be *consistent* with E if, for every $1 \leq i \leq n$, Y_i is not divided by E. Observe that if $\mathcal{I}$ is a model of KB then:

- since C is a concept of $\mathcal{L}_{\Sigma^\dagger,\Phi^\dagger}$, by [7, Theorems 2 and 3], $C^\mathcal{I}$ should be the union of a number of equivalence classes of $\Delta^\mathcal{I}$ w.r.t. $\equiv_{\Sigma^\dagger,\Phi^\dagger,\mathcal{I}}$
- we should have that $a^\mathcal{I} \in C^\mathcal{I}$ for all $a \in E^+$, and $a^\mathcal{I} \notin C^\mathcal{I}$ for all $a \in E^-$.

The idea is to use models of KB and bisimulation in those models to guide the search for C. We now describe our method *BBCL2* (*Bisimulation-Based Concept Learning* for knowledge bases in DLs using the *second* setting). It constructs a set of E_0^- of individuals and sets of concepts $\mathbb{C}$, $\mathbb{C}_0$. E_0^- will cover more and more individuals from E^-. The meaning of $\mathbb{C}$ is to collect concepts D such that $KB \models D(a)$ for all $a \in E^+$. The set $\mathbb{C}_0$ is auxiliary for the construction of $\mathbb{C}$. When a concept D does not satisfy the mentioned condition but is a "good" candidate for that, we put it into $\mathbb{C}_0$. Later, when necessary, we take disjunctions of some concepts from $\mathbb{C}_0$ and check whether they are good for adding to $\mathbb{C}$. During the learning process, we will always have that:

- $KB \models (\bigsqcap \mathbb{C})(a)$ for all $a \in E^+$,
- $KB \not\models (\bigsqcap \mathbb{C})(a)$ for all $a \in E_0^-$,

where $\bigsqcap\{D_1, \ldots, D_n\} = D_1 \sqcap \ldots \sqcap D_n$ and $\bigsqcap \emptyset = \top$. We try to extend $\mathbb{C}$ to satisfy $KB \not\models (\bigsqcap \mathbb{C})(a)$ for more and more $a \in E^-$. Extending $\mathbb{C}$ enables extension of E_0^-. When E_0^- reaches E^-, we return the concept $\bigsqcap \mathbb{C}$ after normalization and simplification. Our method is not a detailed algorithm, as we leave some steps at an abstract level, open to implementation heuristics. In particular, we assume that it is known whether L has the finite model property, how to construct models of KB, and how to do instance checking $KB \models D(a)$ for arbitrary D and a. The steps of our method are as follows.

1. Initialize $E_0^- := \emptyset$, $\mathbb{C} := \emptyset$, $\mathbb{C}_0 := \emptyset$.
2. (This is the beginning of a loop controlled by "go to" at Step 6.) If L has the finite model property then construct a (next) finite model $\mathcal{I}$ of KB. Otherwise, construct a (next) interpretation $\mathcal{I}$ such that either $\mathcal{I}$ is a finite model of KB or $\mathcal{I} = \mathcal{I}'_{|K}$, where $\mathcal{I}'$ is an infinite model of KB and K is a parameter of the learning method (e.g., with value 5).
3. Starting from the partition $\{\Delta^{\mathcal{I}}\}$, make subsequent granulations to reach the partition $\{Y_{i_1}, \ldots, Y_{i_k}\}$ corresponding to $\equiv_{\Sigma^\dagger, \Phi^\dagger, \mathcal{I}}$, where each Y_{i_j} is characterized by an appropriate concept C_{i_j} (such that $Y_{i_j} = C_{i_j}^{\mathcal{I}}$).
4. For each $1 \leq j \leq k$, if Y_{i_j} contains some $a^{\mathcal{I}}$ with $a \in E^-$ and no $a^{\mathcal{I}}$ with $a \in E^+$ then:
   - if $KB \models \neg C_{i_j}(a)$ for all $a \in E^+$ then
     if $\bigsqcap \mathbb{C}$ is not subsumed by $\neg C_{i_j}$ w.r.t. KB (i.e. $KB \not\models (\bigsqcap \mathbb{C} \sqsubseteq \neg C_{i_j})$) then add $\neg C_{i_j}$ to $\mathbb{C}$ and add to E_0^- all $a \in E^-$ such that $a^{\mathcal{I}} \in Y_{i_j}$
   - else add $\neg C_{i_j}$ to $\mathbb{C}_0$.
5. If $E_0^- = E^-$ then go to Step 8.
6. If it was hard to extend $\mathbb{C}$ during a considerable number of iterations of the loop (with different interpretations $\mathcal{I}$) then go to Step 7, else go to Step 2.
7. Repeat the following:
   (a) Randomly select some concepts $D_1, \ldots, D_l$ from $\mathbb{C}_0$ and let $D = (D_1 \sqcup \ldots \sqcup D_l)$.
   (b) If $KB \models D(a)$ for all $a \in E^+$, $\bigsqcap \mathbb{C}$ is not subsumed by D w.r.t. KB (i.e., $KB \not\models (\bigsqcap \mathbb{C}) \sqsubseteq D$), and $E^- \setminus E_0^-$ contains some a such that $KB \not\models (\bigsqcap \mathbb{C})(a)$, then:
      i. add D to $\mathbb{C}$,
      ii. add to E_0^- all $a \in E^- \setminus E_0^-$ such that $KB \not\models (\bigsqcap \mathbb{C})(a)$,
      iii. if $E_0^- = E^-$ then go to Step 8.
   (c) If it was still too hard to extend $\mathbb{C}$ during a considerable number of iterations of the current loop, or $\mathbb{C}$ is already too big, then stop the process with failure.
8. For each $D \in \mathbb{C}$, if $KB \not\models \bigsqcap(\mathbb{C} \setminus \{D\})(a)$ for all $a \in E^-$ then delete D from $\mathbb{C}$.
9. Let C be a normalized form of $\bigsqcap \mathbb{C}$.[6] Observe that $KB \models C(a)$ for all $a \in E^+$, and $KB \not\models C(a)$ for all $a \in E^-$. Try to simplify C while preserving this property, and then return it.

For Step 2, if L is one of the well known DLs, then $\mathcal{I}$ can be constructed by using a tableau algorithm (see [7] for references). During the construction, randomization is used to a certain extent to make $\mathcal{I}$ different from the interpretations generated in previous iterations of the loop.

We describe Step 3 in more details:

- In the granulation process, we denote the blocks created so far in all steps by $Y_1, \ldots, Y_n$, where the current partition $\{Y_{i_1}, \ldots, Y_{i_k}\}$ consists of only some of them. We do not use the same subscript to denote blocks of different

[6] Normalizing concepts can be done, e.g., as in [11].

- A, where $A \in \Sigma_C^\dagger$
- $A = d$, where $A \in \Sigma_A^\dagger \setminus \Sigma_C^\dagger$ and $d \in dom(A)$
- $A \leq d$ and $A < d$, where $A \in \Sigma_{nA}^\dagger$, $d \in dom(A)$ and d is not a minimal element of $dom(A)$
- $A \geq d$ and $A > d$, where $A \in \Sigma_{nA}^\dagger$, $d \in dom(A)$ and d is not a maximal element of $dom(A)$
- $\exists\sigma.\{d\}$, where $\sigma \in \Sigma_{dR}^\dagger$ and $d \in range(\sigma)$
- $\exists r.C_i$, $\exists r.\top$ and $\forall r.C_i$, where $r \in \Sigma_{oR}^\dagger$ and $1 \leq i \leq n$
- $\exists r^-.C_i$, $\exists r^-.\top$ and $\forall r^-.C_i$, if $I \in \Phi^\dagger$, $r \in \Sigma_{oR}^\dagger$ and $1 \leq i \leq n$
- $\{a\}$, if $O \in \Phi^\dagger$ and $a \in \Sigma_I^\dagger$
- $\leq 1\, r$, if $F \in \Phi^\dagger$ and $r \in \Sigma_{oR}^\dagger$
- $\leq 1\, r^-$, if $\{F, I\} \subseteq \Phi^\dagger$ and $r \in \Sigma_{oR}^\dagger$
- $\geq l\, r$ and $\leq m\, r$, if $N \in \Phi^\dagger$, $r \in \Sigma_{oR}^\dagger$, $0 < l \leq \#\Delta^\mathcal{I}$ and $0 \leq m < \#\Delta^\mathcal{I}$
- $\geq l\, r^-$ and $\leq m\, r^-$, if $\{N, I\} \subseteq \Phi^\dagger$, $r \in \Sigma_{oR}^\dagger$, $0 < l \leq \#\Delta^\mathcal{I}$ and $0 \leq m < \#\Delta^\mathcal{I}$
- $\geq l\, r.C_i$ and $\leq m\, r.C_i$, if $Q \in \Phi^\dagger$, $r \in \Sigma_{oR}^\dagger$, $1 \leq i \leq n$, $0 < l \leq \#C_i$ and $0 \leq m < \#C_i$
- $\geq l\, r^-.C_i$ and $\leq m\, r^-.C_i$, if $\{Q, I\} \subseteq \Phi^\dagger$, $r \in \Sigma_{oR}^\dagger$, $1 \leq i \leq n$, $0 < l \leq \#C_i$ and $0 \leq m < \#C_i$
- $\exists r.\mathsf{Self}$, if $\mathsf{Self} \in \Phi^\dagger$ and $r \in \Sigma_{oR}^\dagger$

Fig. 3. Selectors. Here, n is the number of blocks created so far when granulating $\Delta^\mathcal{I}$, and C_i is the concept characterizing the block Y_i. It was proved in [15] that using these selectors is sufficient to granulate $\Delta^\mathcal{I}$ to obtain the partition corresponding to $\equiv_{\Sigma^\dagger,\Phi^\dagger,\mathcal{I}}$.

contents (i.e. we always use new subscripts obtained by increasing n for new blocks). We take care that, for each $1 \leq i \leq n$, Y_i is characterized by an appropriate concept C_i (such that $Y_i = C_i^\mathcal{I}$).

- Following [12, 15] we use the concepts listed in Figure 3 (on page 429) as *selectors* for the granulation process. If a block Y_{i_j} $(1 \leq j \leq k)$ is divided by $D^\mathcal{I}$, where D is a selector, then partitioning Y_{i_j} by D is done as follows:
  - $s := n + 1,\ \ t := n + 2,\ \ n := n + 2,$
  - $Y_s := Y_{i_j} \cap D^\mathcal{I},\ \ C_s := C_{i_j} \sqcap D,$
  - $Y_t := Y_{i_j} \cap (\neg D)^\mathcal{I},\ \ C_t := C_{i_j} \sqcap \neg D,$
  - the new partition of $\Delta^\mathcal{I}$ becomes $\{Y_{i_1}, \ldots, Y_{i_k}\} \setminus \{Y_{i_j}\} \cup \{Y_s, Y_t\}$.
- Which block from the current partition should be partitioned first and which selector should be used to partition it are left open for heuristics. For example, one can apply some gain function like the entropy gain measure, while taking into account also simplicity of selectors and the concepts characterizing the blocks. Once again, randomization is used to a certain extent. For example, if some selectors give the same gain and are the best then randomly choose any one of them.

As a modification for Step 3, the granulation process can be stopped as soon as the current partition is consistent with E (or when some criteria are met).

But, if it is hard to extend $\mathbb{C}$ during a considerable number of iterations of the loop (with different interpretations $\mathcal{I}$), then this loosening should be discarded.

Observe that, when $\neg C_{i_j}$ is added to $\mathbb{C}$, we have that $a^{\mathcal{I}} \in (\neg C_{i_j})^{\mathcal{I}}$ for all $a \in E^+$. This is a good point for hoping that $KB \models \neg C_{i_j}(a)$ for all $a \in E^+$. We check it, for example, by using some appropriate tableau decision procedure, and if it holds then we add $\neg C_{i_j}$ to the set $\mathbb{C}$. Otherwise, we add $\neg C_{i_j}$ to $\mathbb{C}_0$. To increase the chance to have C_{i_j} satisfying the mentioned condition and being added to $\mathbb{C}$, we tend to make C_{i_j} strong enough. For this reason, we do not use the technique with *LargestContainer* introduced in [12], and when necessary, we do not apply the above mentioned loosening for Step 3.

Note that any single concept D from $\mathbb{C}_0$ does not satisfy the condition $KB \models D(a)$ for all $a \in E^+$, but when we take a number of concepts $D_1, \ldots, D_l$ from $\mathbb{C}_0$ we may have that $KB \models (D_1 \sqcup \ldots \sqcup D_l)(a)$ for all $a \in E^+$. So, when it is really hard to extend $\mathbb{C}$ by directly using concepts $\neg C_{i_j}$ (where C_{i_j} are the concepts used for characterizing blocks of partitions of the domains of models of KB), we change to using disjunctions $D_1 \sqcup \ldots \sqcup D_l$ of concepts from $\mathbb{C}_0$ as candidates for adding to $\mathbb{C}$.

4 Illustrative Examples

Example 4.1. Let $KB_0 = \langle \mathcal{R}, \mathcal{T}, \mathcal{A}_0 \rangle$ be the knowledge base given in Example 2.1. Let $E^+ = \{P_4, P_6\}$, $E^- = \{P_1, P_2, P_3, P_5\}$, $\Sigma^\dagger = \{Awarded, cited\_by\}$ and $\Phi^\dagger = \emptyset$. As usual, let $KB = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A} = \mathcal{A}_0 \cup \{A_d(a) \mid a \in E^+\} \cup \{\neg A_d(a) \mid a \in E^-\}$. Execution of our BBCL2 method on this example is as follows.

1. $E_0^- := \emptyset$, $\mathbb{C} := \emptyset$, $\mathbb{C}_0 := \emptyset$.
2. KB has infinitely many models, but the most natural one is $\mathcal{I}$ specified below, which will be used first:

$$\Delta^{\mathcal{I}} = \{P_1, P_2, P_3, P_4, P_5, P_6\}, \quad x^{\mathcal{I}} = x \text{ for } x \in \{P_1, P_2, P_3, P_4, P_5, P_6\},$$
$$Pub^{\mathcal{I}} = \Delta^{\mathcal{I}}, \quad Awarded^{\mathcal{I}} = \{P_1, P_4, P_6\},$$
$$cites^{\mathcal{I}} = \{\langle P_1, P_2 \rangle, \langle P_1, P_3 \rangle, \langle P_1, P_4 \rangle, \langle P_1, P_6 \rangle, \langle P_2, P_3 \rangle, \langle P_2, P_4 \rangle,$$
$$\langle P_2, P_5 \rangle, \langle P_3, P_4 \rangle, \langle P_3, P_5 \rangle, \langle P_3, P_6 \rangle, \langle P_4, P_5 \rangle, \langle P_4, P_6 \rangle\},$$
$$cited\_by^{\mathcal{I}} = (cites^{\mathcal{I}})^{-1}, \text{ the function } Year^{\mathcal{I}} \text{ is specified as usual.}$$

3. $Y_1 := \Delta^{\mathcal{I}}$, $partition := \{Y_1\}$
4. Partitioning Y_1 by $Awarded$:
   - $Y_2 := \{P_1, P_4, P_6\}$, $C_2 := Awarded$,
   - $Y_3 := \{P_2, P_3, P_5\}$, $C_3 := \neg Awarded$,
   - $partition := \{Y_2, Y_3\}$.
5. Partitioning Y_2:
   - All the selectors $\exists cited\_by.\top$, $\exists cited\_by.C_2$ and $\exists cited\_by.C_3$ partition Y_2 in the same way. We choose $\exists cited\_by.\top$, as it is the simplest one.
   - $Y_4 := \{P_4, P_6\}$, $C_4 := C_2 \sqcap \exists cited\_by.\top$,

- $Y_5 := \{P_1\}$, $C_5 := C_2 \sqcap \neg\exists cited\_by.\top$,
- $partition := \{Y_3, Y_4, Y_5\}$.

6. The obtained partition is consistent with E, having $Y_3 = \{P_2, P_3, P_5\} \subset E^-$, $Y_4 = \{P_4, P_6\} = E^+$ and $Y_5 = \{P_1\} \subset E^-$. (It is not yet the partition corresponding to $\equiv_{\Sigma^\dagger, \Phi^\dagger, \mathcal{I}}$.)
7. Since $Y_3 \subset E^-$ and $KB \models \neg C_3(a)$ for all $a \in E^+$, we add $\neg C_3$ to $\mathbb{C}$ and add the elements of Y_3 to E_0^-. Thus, $\mathbb{C} = \{\neg C_3\}$ and $E_0^- = \{P_2, P_3, P_5\}$.
8. Since $Y_5 \subset E^-$ and $KB \models \neg C_5(a)$ for all $a \in E^+$ and $\bigsqcap\mathbb{C}$ is not subsumed by $\neg C_5$ w.r.t. KB, we add $\neg C_5$ to $\mathbb{C}$ and add the elements of Y_5 to E_0^-. Thus, $\mathbb{C} = \{\neg C_3, \neg C_5\}$, $\bigsqcap\mathbb{C} = \neg\neg Awarded \sqcap \neg(Awarded \sqcap \neg\exists cited\_by.\top)$ and $E_0^- = \{P_1, P_2, P_3, P_5\}$.
9. Since $E_0^- = E^-$, we normalize $\bigsqcap\mathbb{C}$ to $Awarded \sqcap \exists cited\_by.\top$ and return it as the result. (This concept denotes the set of publications which were awarded and cited.) ◁

Example 4.2. Let KB_0, E^+, E^-, KB and $\Phi^\dagger$ be as in Example 4.1, but let $\Sigma^\dagger = \{cited\_by, Year\}$. Execution of the BBCL2 method on this new example has the same first two steps as in Example 4.1, and then continues as follows.

1. Granulating $\{\Delta^\mathcal{I}\}$ as in [15, Example 11] we reach the partition $\{Y_4, Y_6, Y_7, Y_8, Y_9\}$ consistent with E and have that:
   - $Y_4 = \{P_4\}$, $Y_6 = \{P_1\}$, $Y_7 = \{P_2, P_3\}$, $Y_8 = \{P_6\}$, $Y_9 = \{P_5\}$,
   - $C_2 = (Year \geq 2008)$, $C_3 = (Year < 2008)$,
     $C_5 = C_3 \sqcap (Year < 2007)$, $C_6 = C_2 \sqcap (Year \geq 2010)$,
     $C_7 = C_2 \sqcap (Year < 2010)$, $C_9 = C_5 \sqcap \neg\exists cited\_by.C_6$.
2. We have $C_6 = (Year \geq 2008) \sqcap (Year \geq 2010)$. Since $Y_6 \subset E^-$ and $KB \models \neg C_6(a)$ for all $a \in E^+$, we add $\neg C_6$ to $\mathbb{C}$ and add the elements of Y_6 to E_0^-. Thus, $\mathbb{C} = \{\neg C_6\}$ and $E_0^- = \{P_1\}$.
3. We have $C_7 := (Year \geq 2008) \sqcap (Year < 2010)$. Since $Y_7 \subset E^-$ and $KB \models \neg C_7(a)$ for all $a \in E^+$ and $\bigsqcap\mathbb{C}$ is not subsumed by $\neg C_7$ w.r.t. KB, we add $\neg C_7$ to $\mathbb{C}$ and add the elements of Y_7 to E_0^-. Thus, $\mathbb{C} = \{\neg C_6, \neg C_7\}$ and $E_0^- = \{P_1, P_2, P_3\}$.
4. We have $C_9 := (Year < 2008) \sqcap (Year < 2007) \sqcap \neg\exists cited\_by.((Year \geq 2008) \sqcap (Year \geq 2010))$. Since $Y_9 \subset E^-$ and $KB \models \neg C_9(a)$ for all $a \in E^+$ and $\bigsqcap\mathbb{C}$ is not subsumed by $\neg C_9$ w.r.t. KB, we add $\neg C_9$ to $\mathbb{C}$ and add the elements of Y_9 to E_0^-. Thus, $\mathbb{C} = \{\neg C_6, \neg C_7, \neg C_9\}$ and $E_0^- = \{P_1, P_2, P_3, P_5\}$.
5. Since $E_0^- = E^-$, we normalize and simplify $\bigsqcap\mathbb{C}$ before returning it as the result. Without exploiting the fact that publication years are integers, $\bigsqcap\mathbb{C}$ can be normalized to

$$(Year < 2008) \sqcap [(Year \geq 2007) \sqcup \exists cited\_by.(Year \geq 2010)].$$

   $C = (Year < 2008) \sqcap \exists cited\_by.(Year \geq 2010)$ is a simplified form of the above concept, which still satisfies that $KB \models C(a)$ for all $a \in E^+$ and $KB \not\models C(a)$ for all $a \in E^-$. Thus, we return it as the result. (The returned concept denotes the set of publications released before 2008 that are cited by some publications released since 2010.) ◁

5 Discussion and Conclusion

We first compare the BBCL2 method with the BBCL and dual-BBCL methods from our joint work [7]. First of all, BBCL2 is used for the second setting of concept learning in DLs, while BBCL and dual-BBCL are used for the first setting. BBCL2 is derived from dual-BBCL, but it contains substantial modifications needed for the change of setting. BBCL2 differs from BBCL at Steps 1, 4, 5, 7, 8, 9, and differs from dual-BBCL by the use of E_0^- at Steps 1, 4, 5 and 7.

Comparing the examples given in this paper and in [7], apart from detailed technical differences in concept learning, it can be seen that the first setting requires more knowledge[7] in order to obtain similar effects as the second setting. In other words, the second setting has effects of a kind of closed world assumption, while the first setting does not. The overall impression is that the second setting is more convenient than the first one.

Recall that our BBCL2 method is the *first bisimulation-based* method for concept learning in DLs using the second setting. As for the case of BBCL and dual-BBCL, it is formulated for the class of decidable $\mathcal{ALC}_{\Sigma,\Phi}$ DLs that have the finite or semi-finite model property, where $\Phi \subseteq \{I, O, F, N, Q, U, \mathsf{Self}\}$. This class contains many useful DLs. For example, $\mathcal{SROIQ}$ (the logical base of OWL 2) belongs to this class. Our method is applicable also to other decidable DLs with the finite or semi-finite model property. The only additional requirement is that those DLs have a good set of selectors (in the sense of [15, Theorem 10]).

Like BBCL and dual-BBCL, the idea of BBCL2 is to use models of the considered knowledge base and bisimulation in those models to guide the search for the concept. Thus, it is completely different from the previous works [6, 10] on concept learning in DLs using the second setting. As bisimulation is the notion for characterizing indiscernibility of objects in DLs, our BBCL2 method is natural and very promising. We intend to implement BBCL2 in the near future.

Acknowledgments

This paper was written during the first author's visit at Warsaw Center of Mathematics and Computer Science (WCMCS). He would like to thank WCMCS for the support. This work was also supported by Polish National Science Centre (NCN) under Grant No. 2011/01/B/ST6/02759 as well as by Polish National Center for Research and Development (NCBiR) under Grant No. SP/I/1/77065/10 by the strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".

References

1. L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In *Proceedings of ILP'2000*, volume 1866 of *LNCS*, pages 40–59. Springer, 2000.

[7] like the assertions $(\neg\exists \mathit{cited\_by}.\top)(P_1)$ and $(\forall \mathit{cited\_by}.\{P_2, P_3, P_4\})(P_5)$, which state that P_1 is not cited by any publication and P_5 is only cited by P_2, P_3 and P_4

2. M. Baldoni, L. Giordano, and A. Martelli. A tableau for multimodal logics and some (un)decidability results. In *Proceedings of TABLEAUX'1998*, volume 1397 of *LNCS*, pages 44–59. Springer, 1998.
3. W.W. Cohen and H. Hirsh. Learning the Classic description logic: Theoretical and experimental results. In *Proceedings of KR'1994*, pages 121–133.
4. A.R. Divroodi, Q.-T. Ha, L.A. Nguyen, and H.S. Nguyen. On C-learnability in description logics. In *Proceedings of ICCCI'2012 (1)*, volume 7653 of *LNCS*, pages 230–238. Springer, 2012.
5. A.R. Divroodi and L.A. Nguyen. On bisimulations for description logics. In *Proceedings of CS&P'2011*, pages 99–110 (see also arXiv:1104.1964).
6. N. Fanizzi, C. d'Amato, and F. Esposito. DL-FOIL concept learning in description logics. In *Proc. of ILP'2008*, volume 5194 of *LNCS*, pages 107–121. Springer, 2008.
7. Q.-T. Ha, T.-L.-G. Hoang, L.A. Nguyen, H.S. Nguyen, A. Szałas, and T.-L. Tran. A bisimulation-based method of concept learning for knowledge bases in description logics. In *Proceedings of SoICT'2012*, pages 241–249. ACM, 2012.
8. L. Iannone, I. Palmisano, and N. Fanizzi. An algorithm based on counterfactuals for concept learning in the Semantic Web. *Appl. Intell.*, 26(2):139–159, 2007.
9. P. Lambrix and P. Larocchia. Learning composite concepts. In *Proc. of DL'1998*.
10. J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203–250, 2010.
11. L.A. Nguyen. An efficient tableau prover using global caching for the description logic $\mathcal{ALC}$. *Fundamenta Informaticae*, 93(1-3):273–288, 2009.
12. L.A. Nguyen and A. Szałas. Logic-based roughification. In A. Skowron and Z. Suraj, editors, *Rough Sets and Intelligent Systems (To the Memory of Professor Zdzisław Pawlak), Vol. 1*, pages 529–556. Springer, 2012.
13. Z. Pawlak. *Rough Sets. Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Dordrecht, 1991.
14. Z. Pawlak and A. Skowron. Rudiments of rough sets. *Inf. Sci.*, 177(1):3–27, 2007.
15. T.-L. Tran, Q.-T. Ha, T.-L.-G. Hoang, L.A. Nguyen, H.S. Nguyen, and A. Szałas. Concept learning for description logic-based information systems. In *Proceedings of KSE'2012*, pages 65–73. IEEE Computer Society, 2012.

Preprocessing for Network Reconstruction: Feasibility Test and Handling Infeasibility

Annegret K. Wagler and Jan-Thierry Wegener*

Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
Université Blaise Pascal (Clermont-Ferrand II)
BP 10125, 63173 Aubière Cedex, France
Annegret.Wagler@univ-bpclermont.fr wegener@isima.fr

Abstract. The context of this work is the reconstruction of Petri net models for biological systems from experimental data. Such methods aim at generating all network alternatives fitting the given data. For a successful reconstruction, the data need to satisfy two properties: reproducibility and monotonicity. In this paper, we focus on a necessary preprocessing step for a recent reconstruction approach. We test the data for reproducibility, provide a feasibility test to detect cases where the reconstruction from the given data may fail, and provide a strategy to cope with the infeasible cases. After having performed the preprocessing step, it is guaranteed that the (given or modified) data are appropriate as input for the main reconstruction algorithm.

1 Introduction

The aim of systems biology is to analyze and understand different phenomena as, e.g., responses of cells to environmental changes, host-pathogen interactions, or effects of gene defects. To gain the required insight into the underlying biological systems, experiments are performed and the resulting experimental data have to be interpreted in terms of models that reflect the observed phenomena. Depending on the biological aim and the type and quality of the available data, different types of mathematical models are used and corresponding methods for their reconstruction have been developed. We focus on Petri nets, a framework which turned out to coherently model both static interactions in terms of networks and dynamic processes in terms of state changes [1–4].

In fact, a (standard) network $\mathcal{P} = (P, T, A, w)$ reflects the involved system components by places $p \in P$ and their interactions by transitions $t \in T$, the arcs in $A \subset (P \times T) \cup (T \times P)$ link places and transitions, and the arc weights $w : A \to \mathbb{N}$ reflect stoichiometric coefficients of the corresponding reactions. Moreover, each place $p \in P$ can be marked with an integral number x_p of tokens defining a system state $\boldsymbol{x} \in \mathbb{Z}_+^{|P|}$. If a capacity $\mathrm{cap}(p)$ is given for the places,

* This work was founded by the French National Research Agency, the European Commission (Feder funds) and the Région Auvergne in the Framework of the LabEx IMobS^3.

then $x_p \leq \mathrm{cap}(p)$ follows and we obtain $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{N}^{|P|} : x_p \leq \mathrm{cap}(p)\}$ as set of potential states. A transition $t \in T$ is *enabled* in a state $\boldsymbol{x}$ if $x_p \geq w(p,t)$ for all p with $(p,t) \in A$ (and $x_p + w(t,p) \leq \mathrm{cap}(p)$ for all $(t,p) \in A$), switching or firing t yields a successor state $\mathrm{succ}(\boldsymbol{x}) = \boldsymbol{x}'$ with $x'_p = x_p - w(p,t)$ for all $(p,t) \in A$ and $x'_p = x_p + w(t,p)$ for all $(t,p) \in A$. Dynamic processes are represented by sequences of such state changes.

Petri net models can be reconstructed from experimental time-series data by means of exact, exclusively data-driven reconstruction approaches [5–10]. These approaches take as input a set P of places and discrete time-series data $\mathcal{X}'$ given by sequences $(\boldsymbol{x}^0; \boldsymbol{x}^1, \ldots, \boldsymbol{x}^m)$ of experimentally observed system states. The goal is to determine all Petri nets (P, T, A, w) that are able to reproduce the data, i.e., that perform for each $\boldsymbol{x}^j \in \mathcal{X}'$ the experimentally observed state change to $\boldsymbol{x}^{j+1} \in \mathcal{X}'$ in a simulation.

In general, there can be more than one transition enabled at a state. The decision which transition switches is typically taken randomly (and the dynamic behavior is analyzed in terms of reachability, starting from a certain initial state). To properly predict the dynamic behavior, (standard) Petri nets have to be equipped with additional activation rules to force the switching or firing of special transitions, and to prevent all others from switching.

This can be done by using priority relations and control-arcs and leads to the notion of $\mathcal{X}'$-deterministic Petri nets [11], which show a prescribed behavior on the experimentally observed subset $\mathcal{X}'$ of states: the reconstructed Petri nets do not only contain enough transitions to reach the experimentally observed successors $\boldsymbol{x}^{j+1}$ from $\boldsymbol{x}^j$, but exactly this transition will be selected among all enabled ones in $\boldsymbol{x}^j$ which is necessary to reach $\boldsymbol{x}^{j+1}$ (see Section 2.2 for details).

For a successful reconstruction, the data $\mathcal{X}'$ need to satisfy two properties: reproducibility (for each $\boldsymbol{x}^j \in \mathcal{X}'$ there is a unique observed successor state $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^{j+1} \in \mathcal{X}'$) and monotonicity (meaning that all essential responses are indeed reported in the experiments), see Section 2.1. Having reproducible data is clearly evident for a successful reconstruction; the necessity of monotone data is shown in [12].

In this paper, we focus on a necessary preprocessing step for the reconstruction approach described in [8]. We test the data for reproducibility, provide a feasibility test (based on previous works in [7]) to detect cases where the reconstruction from the given data may fail (see Section 3.1), and provide a strategy (based on previous works in [7,9]) to cope with infeasible cases (see Section 3.2). We close with some concluding remarks.

2 Reconstructing Petri Nets from Experimental Data

In this section we describe the input and the desired output of the reconstruction method from [8], whereas we refer the reader for details on the reconstruction approach itself to [8].

2.1 Input: Experimental Time-Series Data

First, a set of components P (later represented by the set of places) is chosen which is expected to be crucial for the studied phenomenon and which can be treated in terms of measurements[1].

To perform an experiment, the system is stimulated in a state $\boldsymbol{x}^0$ (by external stimuli like the change of nutrient concentrations or the exposition to some pathogens) to generate an initial state $\boldsymbol{x}^1 \in \mathcal{X}$. Then the system's response to the stimulation is observed and the resulting state changes are measured at certain time points. This yields a sequence $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)$ of states $\boldsymbol{x}^i \in \mathcal{X}$ reflecting the time-dependent response of the system to the stimulation, denoted by

$$\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k) = (\boldsymbol{x}^0; \boldsymbol{x}^1, \ldots, \boldsymbol{x}^k).$$

Note that we also provide the state $\boldsymbol{x}^0$ as the starting point for the stimulation, which will be needed later (see Section 3.2). Every sequence has an observed *terminal state* $\boldsymbol{x}^k \in \mathcal{X}$, without further changes of the system. The set of all terminal states in $\mathcal{X}'$ is denoted by $\mathcal{X}'_{term}$.

For technical reasons, we interpret a terminal state $\boldsymbol{x}^k \in \mathcal{X}'_{term}$ as a state which has itself as observed successor state, i.e., $\boldsymbol{x}^k = \mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^k)$.

Typically, several experiments starting from different initial states in a set $\mathcal{X}'_{ini} \subseteq \mathcal{X}$ are necessary to describe the whole phenomenon, and we obtain *experimental time-series data* of the form

$$\mathcal{X}' = \{\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k) : \boldsymbol{x}^1 \in \mathcal{X}'_{ini}, \boldsymbol{x}^k \in \mathcal{X}'_{term}\}.$$

We write $\boldsymbol{x} \in \mathcal{X}'$ to indicate that $\boldsymbol{x}$ is an element of a sequence $\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k) \in \mathcal{X}'$.

Example 1. As running example, we consider the *light-induced sporulation of Physarum polycephalum* [10]. The developmental decision of *P. polycephalum* plasmodia to enter the sporulation pathway is controlled by environmental factors like visible light [13]. A phytochrome-like photoreversible photoreceptor protein is involved in the control of sporulation *Spo* which occurs in two stages P_{FR} and P_R. If the dark-adapted form P_{FR} absorbs far-red light FR, the receptor is converted into its red-absorbing form P_R, which causes sporulation [14]. If P_R is exposed to red light R, it is photo-converted back to the initial stage P_{FR}, which can prevent sporulation in an early stage, but does not prevent sporulation in a later stage. Figure 1 gives an example of experimental time-series data reflecting this behavior, containing three time-series: $\mathcal{X}(\boldsymbol{x}^1, \boldsymbol{x}^4) = (\boldsymbol{x}^0; \boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3, \boldsymbol{x}^4)$, $\mathcal{X}(\boldsymbol{x}^5, \boldsymbol{x}^0) = (\boldsymbol{x}^2; \boldsymbol{x}^5, \boldsymbol{x}^0)$ and $\mathcal{X}(\boldsymbol{x}^6, \boldsymbol{x}^8) = (\boldsymbol{x}^3; \boldsymbol{x}^6, \boldsymbol{x}^7, \boldsymbol{x}^8)$.

In the best case, two consecutively measured states $\boldsymbol{x}^j, \boldsymbol{x}^{j+1} \in \mathcal{X}'$ are also consecutive system states, i.e., $\boldsymbol{x}^{j+1}$ can be obtained from $\boldsymbol{x}^j$ by switching a single transition. This is, however, in general not the case (and depends on the chosen time points to measure the states in $\mathcal{X}'$), but $\boldsymbol{x}^{j+1}$ is obtained from $\boldsymbol{x}^j$

[1] Possibly, it is known that a certain component plays a crucial role, but it is not possible to measure the values of that component experimentally.

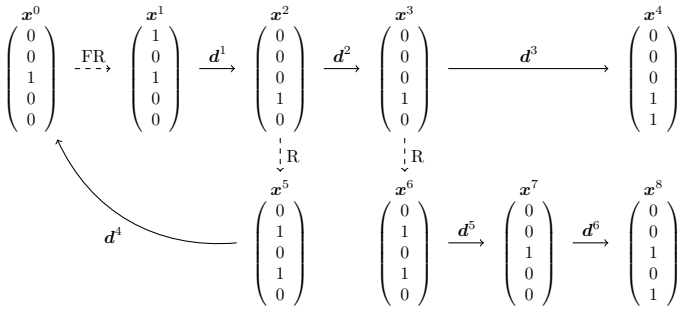


Fig. 1. This figure shows experimental time-series data $\mathcal{X}'$ for the light-induced sporulation of Physarum polycephalum. The experimental setting uses the set $P = \{FR, R, P_{fr}, P_r, Sp\}$ of studied components, observed states are represented by vectors of the form $\boldsymbol{x} = (x_{FR}, x_R, x_{P_{fr}}, x_{P_r}, x_{Sp})^T$ having 0/1-entries only. Dashed arrows represent stimulations to the system and solid arrows represent the observed responses.

by a switching sequence of some length, where the intermediate states are not reported in $\mathcal{X}'$.

For a successful reconstruction, the data $\mathcal{X}'$ need to satisfy two properties: reproducibility and monotonicity. The data $\mathcal{X}'$ are *reproducible* if for each $\boldsymbol{x}^j \in \mathcal{X}'$ there is a unique observed successor state $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^{j+1} \in \mathcal{X}'$. Moreover, the data $\mathcal{X}'$ are *monotone* if for each such pair $(\boldsymbol{x}^j, \boldsymbol{x}^{j+1}) \in \mathcal{X}'$, the possible intermediate states $\boldsymbol{x}^j = \boldsymbol{y}^1, \boldsymbol{y}^2, ..., \boldsymbol{y}^{m+1} = \boldsymbol{x}^{j+1}$ satisfy

$$y_p^1 \leq y_p^2 \leq \ldots \leq y_p^m \leq y_p^{m+1} \text{ for all } p \in P \text{ with } x_p^j \leq x_p^{j+1} \text{ and}$$
$$y_p^1 \geq y_p^2 \geq \ldots \geq y_p^m \geq y_p^{m+1} \text{ for all } p \in P \text{ with } x_p^j \geq x_p^{j+1}.$$

Whereas reproducibility is obviously necessary, it was shown in [12] that monotonicity has to be required or, equivalently, that all essential responses are indeed reported in the experiments.

Remark 1. When continuous data is discretized for the reconstruction approach, all local minima and maxima of the measured values have to be kept for each $p \in P$ to ensure monotonicity.

2.2 Output: $\mathcal{X}'$-Deterministic Extended Petri Nets

A standard Petri net $\mathcal{P} = (P, T, A, w)$ fits the given data $\mathcal{X}'$ when it is able to perform every observed state change from $\boldsymbol{x}^j \in \mathcal{X}'$ to $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^{j+1} \in \mathcal{X}'$. This can be interpreted as follows. With $\mathcal{P}$, an *incidence matrix* $M \in \mathbb{Z}^{|P| \times |T|}$ is associated, where each row corresponds to a place $p \in P$ of the network, and each column $M_{\cdot t}$ to the *update vector* $\boldsymbol{r}^t$ of a transition $t \in T$:

$$r_p^t = M_{pt} := \begin{cases} -w(p,t) & \text{if } (p,t) \in A, \\ +w(t,p) & \text{if } (t,p) \in A, \\ 0 & \text{otherwise.} \end{cases}$$

Reaching $\boldsymbol{x}^{j+1}$ from $\boldsymbol{x}^j$ by a switching sequence using the transitions from a subset $T' \subseteq T$ is equivalent to obtain the state vector $\boldsymbol{x}^{j+1}$ from $\boldsymbol{x}^j$ by adding the corresponding columns $M_{\cdot t}$ of M for all $t \in T'$:

$$\boldsymbol{x}^j + \sum_{t \in T'} M_{\cdot t} = \boldsymbol{x}^{j+1}. \tag{1}$$

Hence, T has to contain enough transitions to perform all experimentally observed switching sequences. The network $\mathcal{P} = (P, T, A, w)$ is *conformal* with $\mathcal{X}'$ if, for any two consecutive states $\boldsymbol{x}^j, \mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^{j+1} \in \mathcal{X}'$, the linear equation system $\boldsymbol{x}^{j+1} - \boldsymbol{x}^j = M\lambda$ has an integral solution $\lambda \in \mathbb{N}^{|T|}$ such that λ is the incidence vector of a sequence $(t^1, ..., t^m)$ of transition switches, i.e., there are intermediate states $\boldsymbol{x}^j = \boldsymbol{y}^1, \boldsymbol{y}^2, ..., \boldsymbol{y}^{m+1} = \boldsymbol{x}^{j+1}$ with $\boldsymbol{y}^l + M_{\cdot t^l} = \boldsymbol{y}^{l+1}$ for $1 \leq l \leq m$. Hereby, monotonicity avoids unnecessary solutions, since no homogeneous solutions of equation (1) have to be considered, see [10, 12].

To also force that the networks exhibit the experimentally observed dynamic behavior in a simulation, we equip standard networks with additional activation rules to further control the switching of enabled transitions, see [5, 6, 8, 11].

On the one hand, the concept of control-arcs can be used to represent catalytic or inhibitory dependencies. An *extended Petri net* $\mathcal{P} = (P, T, (A \cup A_R \cup A_I), w)$ is a Petri net which has, besides the (standard) arcs in A, two additional sets of so-called control-arcs: the set of read-arcs $A_R \subset P \times T$ and the set of inhibitor-arcs $A_I \subset P \times T$. We denote the set of all arcs by $\mathcal{A} = A \cup A_R \cup A_I$. Here, an enabled transition $t \in T$ coupled with a read-arc (resp. an inhibitor-arc) to a place $p \in P$ can switch in a state $\boldsymbol{x}$ only if a token (resp. no token) is present in p; we denote by $T_{\mathcal{A}}(\boldsymbol{x})$ the set of all such transitions.

On the other hand, in [9, 10, 15] the concept of priority relations among the transitions of a network was introduced in order to allow the modeling of deterministic systems. In Marwan et al. [9] it is proposed to model such priorities with the help of partial orders $\mathcal{O}$ on the transitions in order to reflect the rates of the corresponding reactions where the fastest reaction has highest priority and, thus, is taken. For each state $\boldsymbol{x}$, only a transition is allowed to switch if it is enabled and there is no other enabled transition with higher priority according to $\mathcal{O}$; we denote by $T_{\mathcal{A},\mathcal{O}}(\boldsymbol{x})$ the set of all such transitions. We call $(\mathcal{P}, \mathcal{O})$ a *Petri net with priorities* if $\mathcal{P} = (P, T, \mathcal{A}, w)$ is a (standard or extended) Petri net and $\mathcal{O}$ a priority relation on T.

For a deterministic behavior, $T_{\mathcal{A},\mathcal{O}}(\boldsymbol{x})$ must contain at most one element for each state $\boldsymbol{x}$ to enforce that $\boldsymbol{x}$ has a unique successor state $\mathrm{succ}_{\mathcal{X}}(\boldsymbol{x})$, see [15] for more details. For our purpose we consider a relaxed condition, namely that $T_{\mathcal{A},\mathcal{O}}(\boldsymbol{x})$ contains at most one element for each experimentally observed state $\boldsymbol{x} \in \mathcal{X}'$, but $T_{\mathcal{A},\mathcal{O}}(\boldsymbol{x})$ may contain several elements for non-observed states $\boldsymbol{x} \in \mathcal{X} \setminus \mathcal{X}'$. We call such Petri nets $\mathcal{X}'$-*deterministic* (see [11]).

The extended Petri net $\mathcal{P} = (P, T, \mathcal{A}, w)$ is *catalytically conformal* with $\mathcal{X}'$ if $t^l \in T_{\mathcal{A}}(\boldsymbol{y}^l)$ for each intermediate state $\boldsymbol{y}^l$ of any pair $(\boldsymbol{x}^j, \boldsymbol{x}^{j+1}) \in \mathcal{X}'$, and the extended Petri net with priorities $(\mathcal{P}, \mathcal{O})$ is $\mathcal{X}'$-*deterministic* if $\{t^l\} = T_{\mathcal{A},\mathcal{O}}(\boldsymbol{y}^l)$ holds for all $\boldsymbol{y}^l$.

The desired output of the reconstruction approach consists of the set of all $\mathcal{X}'$-deterministic extended Petri nets $(\mathcal{P}, \mathrm{cap}, \mathcal{O})$ (all having the same set P of places and the same capacities cap deduced from $\mathcal{X}'$ by $\mathrm{cap}(p) = \max\{x_p : \boldsymbol{x} \in \mathcal{X}'\}$).

Figure 2 shows an $\mathcal{X}'$-deterministic extended Petri net fitting the experimental data from Example 1.

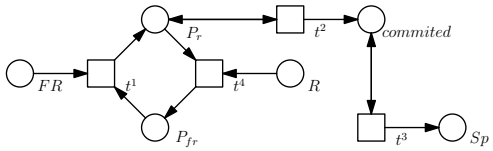


Fig. 2. This figure shows an $\mathcal{X}'$-deterministic extended Petri net fitting the experimental data from Example 1. The set of components is $P = \{FR, R, P_{fr}, P_r, Sp, \mathit{committed}\}$, where FR, R, P_{fr}, P_r and Sp have been measured directly in the experiment. The here added component *committed* cannot be measured directly, but only indirectly by the behavior of Physarum polycephalum observed in the experiment. The here shown network corresponds to solution (a) from Figure 4. In this $\mathcal{X}'$-deterministic extended Petri net there is a read-arc from P_r to t^2 and one from *committed* to t^3. Furthermore, we have the set of priorities $\mathcal{O} = \{t^2 < t^4, t^3 < t^4\}$. The control-arcs and priorities ensure $|T_{\mathcal{A},\mathcal{O}}(\boldsymbol{x})| = 1$ for every state $\boldsymbol{x} \in \mathcal{X}'$.

3 Feasibility Test and Handling Infeasibility

Before the reconstruction is started, a preprocessing step is necessary in order to verify or falsify whether the experimental time-series data $\mathcal{X}'$ is suitable for reconstructing $\mathcal{X}'$-deterministic extended Petri nets (see Section 3.1). If the test is successful, the reconstruction algorithm can be applied. For the case that the given data are not suitable for the reconstruction, we provide a method to handle the infeasible cases (see Section 3.2).

For that, we interpret (as in [7]) the experimental time-series data $\mathcal{X}'$ as a directed graph $\mathcal{D}(\mathcal{X}') = (V_{\mathcal{X}'}, A_D \cup A_S)$ having the measured states $\boldsymbol{x} \in \mathcal{X}'$ as nodes and two kinds of arcs:

- $A_D := \{(\boldsymbol{x}^j, \boldsymbol{x}^{j+1}) : \boldsymbol{x}^{j+1} = \mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j)\}$ for the observed responses,
- $A_S := \{(\boldsymbol{x}^0, \boldsymbol{x}^1) : \mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k) = (\boldsymbol{x}^0; \boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)\}$ for the stimulations.

We call $\mathcal{D}(\mathcal{X}')$ the *experiment graph* of $\mathcal{X}'$. It can be interpreted as a minor of the reachability graph, where observed responses may correspond to directed paths with intermediate states.

Our main objective is to test the given experimental time-series data $\mathcal{X}'$ for reproducibility, i.e., whether each state $\boldsymbol{x} \in \mathcal{X}'$ has a unique successor state $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}) \in \mathcal{X}'$. We provide a feasibility test to ensure this property (based on previous tests for standard Petri nets [7] and extended Petri nets [5], see

Section 3.1). If this test fails, we have a state $\boldsymbol{x} \in \mathcal{X}'$ with at least two successors in $\mathcal{X}'$, and it is not possible to reconstruct an $\mathcal{X}'$-deterministic extended Petri net from $\mathcal{X}'$ in its current form. As proposed in [7, 9, 10], this situation can be resolved by adding further components[2] to P with the goal to split any state $\boldsymbol{x} \in \mathcal{X}'$ with two successors into different states each having a unique successor. We present in Section 3.2 an approach for this step (based on previous works for standard Petri nets [7, 9]).

3.1 $\mathcal{X}'$-Determinism Conflicts and Feasibility Test

Definition 1. *Let $\mathcal{X}'$ be experimental time-series data. We say that two time-series $\mathcal{X}_i = \mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})$ and $\mathcal{X}_\ell = \mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})$ are in* $\mathcal{X}'$-determinism conflict*, when there exists a state $\boldsymbol{x} \in \mathcal{X}'$ with $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}) \neq \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x})$ and call $\boldsymbol{x}$ the corresponding* $\mathcal{X}'$-determinism conflict state*. We have*

- *a* strong $\mathcal{X}'$*-determinism conflict if $\boldsymbol{x}^{i_k} \neq \boldsymbol{x}^{\ell_m}$ or $\mathcal{X}_i = \mathcal{X}_\ell$;*
- *a* weak $\mathcal{X}'$*-determinism conflict if $\boldsymbol{x}^{i_k} = \boldsymbol{x}^{\ell_m}$ and $\mathcal{X}_i \neq \mathcal{X}_\ell$.*

The definition of strong $\mathcal{X}'$-determinism conflicts includes the case discussed in [5, 7] that there must not exist a terminal state $\boldsymbol{x}^j \in \mathcal{X}'_{term}$ that occurs as intermediate state in an experiment. Furthermore, it includes the case that a state $\boldsymbol{x}^j \in \mathcal{X}' \setminus \mathcal{X}'_{term}$ has itself as successor, i.e., $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^j$, which would result in $\boldsymbol{d}^j = 0$ (see Example 2).

Example 2. In the experimental time-series data $\mathcal{X}'$ shown in Figure 1 we have no weak but two strong $\mathcal{X}'$-determinism conflicts:

- in the sequence $\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^4)$ the states $\boldsymbol{x}^2$ and $\boldsymbol{x}^3$ are equal but have different successor states,
- the sequences $\mathcal{X}'(\boldsymbol{x}^5, \boldsymbol{x}^0)$ and $\mathcal{X}'(\boldsymbol{x}^6, \boldsymbol{x}^8)$ have equal initial state $\boldsymbol{x}^5 = \boldsymbol{x}^6$, but different terminal states. Besides the initial states, the states $\boldsymbol{x}^0$ and $\boldsymbol{x}^7$ are $\mathcal{X}'$-determinism conflict states.

Obviously, every $\mathcal{X}'$-determinism conflict violates the condition of the data being reproducible, and the reconstruction of $\mathcal{X}'$-deterministic extended Petri nets from $\mathcal{X}'$ is not possible. However, the converse is true:

Lemma 1. *Let $\mathcal{X}'$ be experimental time-series data. If every state $\boldsymbol{x} \in \mathcal{X}'$ has a unique successor state $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}) \in \mathcal{X}'$ then there exists an $\mathcal{X}'$-deterministic extended Petri net.*

Sketch of the proof. The pre-condition that every state has a unique successor in $\mathcal{X}'$ includes the cases that no non-terminal state has itself as successor and that no terminal state is an intermediate state of any experiment. This guarantees

[2] Since P is only a projection from the real world, it is possible that some components of the system, crucial for the studied phenomenon, were not taken into account or could not be experimentally measured.

the existence of a standard network $\mathcal{P} = (P, T, \cdot, \cdot)$ being conformal with $\mathcal{X}'$. Since every state has a unique successor state it follows for all states $\boldsymbol{x}^j, \boldsymbol{x}^l \in \mathcal{X}'$ with $\mathrm{succ}(\boldsymbol{x}^j) \neq \mathrm{succ}(\boldsymbol{x}^l)$ that there exists a non-empty subset $P' \subseteq P$ so that $\boldsymbol{x}_p^j \neq \boldsymbol{x}_p^l$ holds for every $p \in P'$. Therefore, $\mathcal{P}$ can be made $\mathcal{X}'$-deterministic by adding appropriate control-arcs (p, t), where $p \in P'$ and $t \in T$, in a way that exactly the transition is enabled which was observed in the experiments. □

Two time-series $\mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})$ and $\mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})$ with $\boldsymbol{x}^{i_k} = \boldsymbol{x}^{\ell_m}$ may be in weak $\mathcal{X}'$-determinism conflict, due to differently chosen time points of the measurements. We test the data for such a situation and try to resolve the conflict by linearizing these sequences, respecting monotonicity.

A *linear order* $\mathcal{L}$ (or *total order*) on a set S is a partial order where additionally $(a \leq b) \in \mathcal{L}$ or $(b \leq a) \in \mathcal{L}$ holds for all $a, b \in S$. In this case, we say that the set S is *totally ordered* (w.r.t. $\mathcal{L}$). A totally ordered subset $U \subseteq S$ of a partially ordered set S is called a *chain* of S.

On a time-series $\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k) = (\boldsymbol{x}^0; \boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)$, a linear order is induced by the successor relation: $\boldsymbol{x}^j \leq \boldsymbol{x}^{j+1}$ iff $\boldsymbol{x}^{j+1} = \mathrm{succ}_{\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k)}(\boldsymbol{x}^j)$, hence $\mathcal{X}'$ can be considered as a partially ordered set (ordered by the successor relation), where each time-series $\mathcal{X}'(\boldsymbol{x}^1, \boldsymbol{x}^k)$ is a chain of $\mathcal{X}'$. Let $\mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x}^j) = \boldsymbol{x}^{j+1}$ and

$$\mathrm{Box}(\boldsymbol{x}^j, \boldsymbol{x}^{j+1}) := \left\{ \boldsymbol{y} \in \mathcal{X} : \begin{array}{ll} x_p^j \leq y_p \leq x_p^{j+1} & \text{if } x_p^j \leq x_p^{j+1} \\ x_p^j \geq y_p \geq x_p^{j+1} & \text{if } x_p^j \geq x_p^{j+1} \end{array} \right\}.$$

Note that due to monotonicity, all intermediate states $\boldsymbol{y}$ of any refined sequence from $\boldsymbol{x}^j$ to $\boldsymbol{x}^{j+1}$ lie in $\mathrm{Box}(\boldsymbol{x}^j, \boldsymbol{x}^{j+1})$. Consequently, if two time-series $\mathcal{X}_i = \mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})$ and $\mathcal{X}_\ell = \mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})$ with $\boldsymbol{x}^{i_k} = \boldsymbol{x}^{\ell_m}$ are in weak $\mathcal{X}'$-determinism conflict, and $\boldsymbol{x}$ is a determinism conflict state then we have to test whether

(i) $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}) \in \mathrm{Box}(\boldsymbol{x}, \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x}))$ or
(ii) $\mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x}) \in \mathrm{Box}(\boldsymbol{x}, \mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}))$,

see Figure 3 for an illustration. If one of the two conditions holds, we conclude $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}) < \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x})$ (resp. $\mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x}) < \mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x})$); otherwise we cannot find a $\mathcal{X}'$-deterministic linear order. Therefore, $\boldsymbol{x}$ is no longer a $\mathcal{X}'$-determinism conflict state, but a new $\mathcal{X}'$-determinism conflict state $\boldsymbol{x}'$ is detected since either

(i) $\boldsymbol{x}' = \mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x})$ has two successor states: $\mathrm{succ}_{\mathcal{X}_i}(\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}))$, $\mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x})$ or
(ii) $\boldsymbol{x}' = \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x})$ has two successor states: $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x})$ and $\mathrm{succ}_{\mathcal{X}_\ell}(\mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x}))$.

Hence, the procedure has to be repeated for $\boldsymbol{x}'$ until $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}') = \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x}')$ holds or the test fails (see Algorithm 1). This works since in case of a weak $\mathcal{X}'$-determinism conflict at least the terminal states $\boldsymbol{x}^{i_k}$ and $\boldsymbol{x}^{\ell_m}$ are equal.

Whenever the test described above is successful for $\boldsymbol{x}$ and all subsequent $\mathcal{X}'$-determinism conflict states $\boldsymbol{x}'$, we say that it is *resolvable*, otherwise we say it is an *unresolvable* weak $\mathcal{X}'$-determinism conflict. We further obtain:

Theorem 1. *Let $\mathcal{X}'$ be experimental time-series data. There exists an $\mathcal{X}'$-deterministic extended Petri net if and only if there are neither strong $\mathcal{X}'$-determinism conflicts nor unresolvable weak $\mathcal{X}'$-determinism conflicts.*

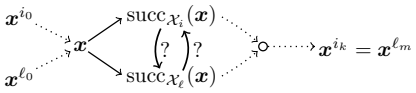


Fig. 3. This figure shows a weak $\mathcal{X}'$-determinism conflict. To resolve this conflict we can test if the two different successor states (resulting from two different experiments) of the $\mathcal{X}'$-determinism conflict state $\boldsymbol{x}$ can be ordered in such a way that the monotonicity constraint is not violated. In other words, we test if one of these successor states is an unmeasured intermediate state of $\boldsymbol{x}$ and the other successor state.

Sketch of the proof. If neither strong $\mathcal{X}'$-determinism conflicts nor unresolvable weak $\mathcal{X}'$-determinism conflicts exists, the statement follows from the procedure described above and from Lemma 1.

Conversely, suppose that an unresolvable weak (or strong) $\mathcal{X}'$-determinism conflict state exists. In the case that $\boldsymbol{x} = \mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x})$ holds for at least one strong $\mathcal{X}'$-determinism conflict state $\boldsymbol{x}$, then there does not exist any standard network being conformal with $\mathcal{X}'$. Otherwise, there exist conformal standard networks, but none of them can be made $\mathcal{X}'$-deterministic.

Let $\boldsymbol{x}$ be an unresolvable weak (or strong) $\mathcal{X}'$-determinism conflict state for two time-series $\mathcal{X}'_i = \mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})$ and $\mathcal{X}'_\ell = \mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})$. First note that $\boldsymbol{x}$ remains an unresolvable weak (or strong) $\mathcal{X}'$-determinism conflict state for every refined sequence (respecting the monotonicity constraint) of $\mathcal{X}'_i$ and $\mathcal{X}'_\ell$. Thus, w.l.o.g. we can assume that $\mathrm{succ}_{\mathcal{X}'_i}(\boldsymbol{x}) \neq \mathrm{succ}_{\mathcal{X}'_\ell}(\boldsymbol{x})$ and denote the respective transitions by t^i and t^ℓ. Since both transitions t^i and t^ℓ are (and need to stay) enabled at $\boldsymbol{x}$, there is no way to add priorities and/or control-arcs to force the network to deterministically show the observed behavior of $\mathcal{X}'_i$ and of $\mathcal{X}'_\ell$ simultaneously. □

3.2 Handling Infeasibility

Due to Theorem 1, it is impossible to reconstruct $\mathcal{X}'$-deterministic extended Petri nets from experimental time-series data $\mathcal{X}'$ containing a strong $\mathcal{X}'$-determinism conflict or an unresolvable weak $\mathcal{X}'$-determinism conflict. In this section we show how these conflicts can be resolved by using additional components.

For that we extend, as proposed in [7,9], all the n-dimensional state vectors $\boldsymbol{x} \in \mathcal{X}'$ to suitable $(n+a)$-dimensional vectors

$$\overline{\boldsymbol{x}}^j := \begin{pmatrix} \boldsymbol{x}^j \\ \boldsymbol{z}^j \end{pmatrix} \in \overline{\mathcal{X}'} = \left\{ \overline{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{pmatrix} \in \mathbb{Z}^{n+a} : \ \mathbf{0} \leq \boldsymbol{z} \leq \mathbf{1}, \ \boldsymbol{x} \in \mathcal{X}' \right\}.$$

The studied extensions $\overline{\boldsymbol{x}}^j \in \mathbb{N}^{n+a}$ of the states $\boldsymbol{x}^j \in \mathcal{X}'$ correspond to suitable *labelings* of the experiment graph $\mathcal{D}(\mathcal{X}')$:

Algorithm 1 Resolving weak $\mathcal{X}'$-determinism conflicts by linearization

Input: time-series $\mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})$, $\mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})$ in weak $\mathcal{X}'$-determinism conflict
Output: adjusted time-series if resolvable weak $\mathcal{X}'$-determinism conflict or *false* otherwise

for all conflict states $\boldsymbol{x}$ **do**
  $\boldsymbol{x}^i \leftarrow \mathrm{succ}_{\mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})}(\boldsymbol{x})$, $\boldsymbol{x}^\ell \leftarrow \mathrm{succ}_{\mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})}(\boldsymbol{x})$
  $\mathcal{L} \leftarrow \emptyset$ ▷ stores the linear order
  while $\boldsymbol{x}^i \neq \boldsymbol{x}^\ell$ **do**
    if $\boldsymbol{x}^i \in \mathrm{Box}(\boldsymbol{x}, \boldsymbol{x}^\ell)$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\boldsymbol{x}^i < \boldsymbol{x}^\ell\}$
      $\boldsymbol{x} \leftarrow \boldsymbol{x}^i$
      $\boldsymbol{x}^i \leftarrow \mathrm{succ}_{\mathcal{X}'(\boldsymbol{x}^{i_0}, \boldsymbol{x}^{i_k})}(\boldsymbol{x}^i)$
    else if $\boldsymbol{x}^l \in \mathrm{Box}(\boldsymbol{x}, \boldsymbol{x}^i)$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\boldsymbol{x}^\ell < \boldsymbol{x}^i\}$
      $\boldsymbol{x} \leftarrow \boldsymbol{x}^\ell$
      $\boldsymbol{x}^\ell \leftarrow \mathrm{succ}_{\mathcal{X}'(\boldsymbol{x}^{\ell_0}, \boldsymbol{x}^{\ell_m})}(\boldsymbol{x}^\ell)$
    else
      return *false*
return adjusted time-series according to $\mathcal{L}$

- if $a = 1$, to $(0,1)$-labelings, where label i is assigned to node $\boldsymbol{x}^j$ if $\overline{x}^j_{n+1} = \boldsymbol{z}^j = i$ is selected for $i \in \{0,1\}$;
- if $a = 2$, to $(0,1,2,3)$-labelings, where the labels are assigned to the four different states $(0,0)^T, (1,0)^T, (0,1)^T$ and $(1,1)^T$;
- if $a \geq 3$ we use similar encodings for all 2^a different 0/1-vectors.

By using appropriate additional components, states that appear equal in experimental time-series data $\mathcal{X}'$ become different in $\overline{\mathcal{X}'}$ (see Figure 4 for an illustration). It is already stressed in [7] that not every labeling for the experiment graph $\mathcal{D}(\mathcal{X}')$ is reasonable, as a state $\overline{\boldsymbol{x}}^k \in \overline{\mathcal{X}'}$ with $\boldsymbol{x}^k \in \mathcal{X}'_{term}$ might have a successor state, a state $\overline{\boldsymbol{x}}^j$ might have multiple successor states, or some stimulation changes more than the target input component(s). To obtain suitable labelings for $\mathcal{X}'$-deterministic extended Petri nets, we adjust Definition 15 from [7]:

Definition 2. *A labeling L of $\mathcal{X}'$ is* valid *if it satisfies the following conditions:*

(i) every state $\overline{\boldsymbol{x}}$ has a unique successor state $\mathrm{succ}(\overline{\boldsymbol{x}})$*,*
(ii) any stimulation preserves the values on the additional component(s),
(iii) for every $\boldsymbol{d} = \mathrm{succ}(\boldsymbol{x}) - \boldsymbol{x}$ and $\boldsymbol{d}' = \mathrm{succ}(\boldsymbol{x}') - \boldsymbol{x}'$ with $\boldsymbol{d} = \boldsymbol{d}'$ follows $\overline{\boldsymbol{d}} = \overline{\boldsymbol{d}}'$.

From Condition (i) we can conclude that we have $\boldsymbol{x} = \mathrm{succ}_{\mathcal{X}'}(\boldsymbol{x})$ if and only if $\boldsymbol{x} \in \mathcal{X}'_{term}$. Condition (ii) ensures that a stimulation does not change more than the target input component(s), and finally, Condition (iii) ensures a minimal number of label switches, while keeping the data as close as possible to the original measurements. Furthermore, due to symmetry reasons, we can choose a label for one state, e.g., a conflict state.

Example 3. Besides symmetric solutions, there are two possible valid labelings with $a = 1$ for the experimental time-series data from Figure 1. These two solutions are shown in Figure 4. The solutions are obtained by applying the conditions of Definition 2 as follows. We start by selecting an $\mathcal{X}'$-determinism conflict state, here $\boldsymbol{x}^2$, and choose its label as $\boldsymbol{x}^2_z = 0$. Due to Condition (ii), $\boldsymbol{x}^5_z = 0$ follows. Condition (i) implies that $\boldsymbol{x}^3$ (resp. $\boldsymbol{x}^6$) must be different from $\boldsymbol{x}^2$ (resp. $\boldsymbol{x}^5$). Therefore, $\boldsymbol{x}^3_z = 1$ and $\boldsymbol{x}^6_z = 1$ follows. Since we have $\boldsymbol{d}^4 = \boldsymbol{d}^5$, Condition (iii) implies that the only valid labels for $\boldsymbol{x}^0$ and $\boldsymbol{x}^7$ are 0 and 1, respectively. Condition (ii) shows $\boldsymbol{x}^1_z = 0$. Finally, we can choose a label for $\boldsymbol{x}^4$ and $\boldsymbol{x}^8$, respectively. However, since $\boldsymbol{d}^3 = \boldsymbol{d}^6$, if follows from (iii) that both labels must be equal.

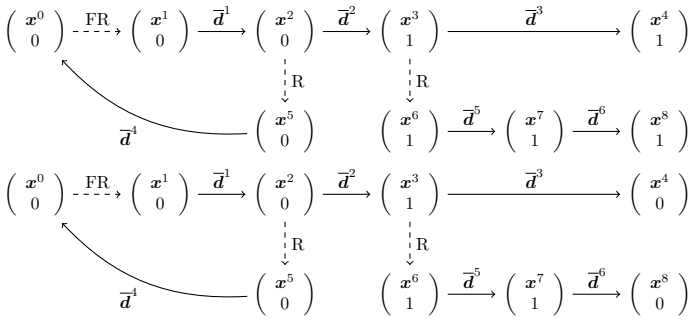


Fig. 4. This figure shows values for additional components resolving the strong $\mathcal{X}'$-determinism conflicts from Example 2 in Figure 1.

In order to find all valid labelings of a general experiment graph $\mathcal{D}(\mathcal{X}') = (V_{\mathcal{X}'}, A_D \cup A_S)$ we set up an optimization problem encoding the conditions for valid labelings and having as objective the minimization of the number a of additional components. For that we introduce decision variables y_{ji} to determine whether label i is assigned to $\boldsymbol{x}^j$.

We are interested in finding $\min\{a \in \mathbb{N} : \mathcal{P}(a) \neq \emptyset\}$, where $\mathcal{P}(a)$ is given by

$$\sum_{i=1}^{a} |y_{ji} - y_{li} - (y_{pi} - y_{qi})| \geq 1 \qquad \text{for all}(\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in A_D, \text{ with } \boldsymbol{x}^j = \boldsymbol{x}^p, \boldsymbol{x}^l \neq \boldsymbol{x}^q \tag{2a}$$

$$y_{ji} - y_{li} = 0 \qquad \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l) \in A_S \tag{2b}$$

$$y_{ji} - y_{li} = y_{pi} - y_{qi} \qquad \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in A_D, \text{ with } \boldsymbol{x}^l - \boldsymbol{x}^j = \boldsymbol{x}^p - \boldsymbol{x}^q \tag{2c}$$

$$y_{j1}, \ldots, y_{j2^a} \in \{0, 1\} \qquad \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l) \in A_D, i = 1, \ldots, 2^a, \tag{2d}$$

where equations (2a) ensure that every state has a unique successor state (Condition (i) from Definition 2), equations (2b) that no stimulation changes the state

of additional components (Condition (ii)), and equations (2c) preserve equal difference vectors (Condition (iii)). The conditions (2d) ensure that we have binary decision variables y_{ij}. Each valid labeling corresponds to a vector in $\mathcal{P}(a)$.

Note, due to inequalities (2a) the optimization problem is non-linear and has a non-convex set of feasible solutions. However, it is only necessary to find the minimal a so that $\mathcal{P}(a) \neq \emptyset$. We can consider the set $\mathcal{P}(a)$ as the union of 2^a convex sets (see Figure 5 for an illustration). Therefore, we can split the problem into 2^a linear subproblems, each having a convex (=polyhedral) feasible region. For that, we define two sets for each subproblem $1 \leq k \leq 2^a$, namely $P^+(k)$ and $P^-(k)$, so that $P^+(k) \cup P^-(k) = \{1, \ldots, a\}$ and $P^+(k) \cap P^-(k) = \emptyset$ and $P^+(p) \neq P^+(q)$, $P^-(p) \neq P^-(q)$ for all $p \neq q$. The sets induce the indices i so that $y_{ji} - y_{li} - (y_{pi} - y_{qi}) \geq 0$ and $y_{ji} - y_{li} - (y_{pi} - y_{qi}) \leq 0$, respectively. Hereby, we have all possible combinations. For the sake of readability let $z_{jlpqi} = y_{ji} - y_{li} - (y_{pi} - y_{qi})$. Then we replace inequalities (2a) by the following constraints

$$\sum_{i^+ \in P^+(k)} z_{jlpqi^+} - \sum_{i^- \in P^-(k)} z_{jlpqi^-} \geq 1 \quad \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in \mathcal{A}_D, \tag{3a}$$

$$z_{jlpqi^+} \geq 0 \quad \begin{array}{l} \text{for all } i^+ \in P^+(k), \\ \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in \mathcal{A}_D, \end{array} \tag{3b}$$

$$z_{jlpqi^-} \leq 0 \quad \begin{array}{l} \text{for all } i^+ \in P^+(k), \\ \text{for all } (\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in \mathcal{A}_D, \end{array} \tag{3c}$$

where $\mathcal{A}_D := \{(\boldsymbol{x}^j, \boldsymbol{x}^l), (\boldsymbol{x}^p, \boldsymbol{x}^q) \in A_D \text{ with } \boldsymbol{x}^j = \boldsymbol{x}^p, \boldsymbol{x}^l \neq \boldsymbol{x}^q\}$. These linear subproblems can be solved by standard solvers, and the optimal solution a of the original problem is obtained if one subproblem turns out to be feasible. All (minimal) valid labelings are then in $\mathcal{P}(a)$.

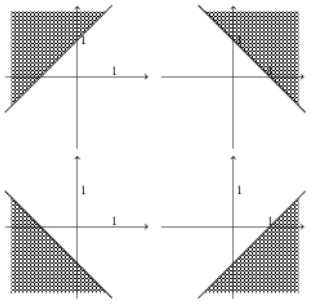


Fig. 5. In this figure the division of (2a) into 2^a subproblems is illustrated within the 2-dimensional space (i.e., $a = 2$). Each of the resulting 4 subproblems has a convex feasible region (highlighted by the dotted regions) whose union corresponds to the feasible region of the original problem.

4 Conclusion

In this work, we give a preprocessing step for a reconstruction algorithm from [8] that reconstructs extended Petri nets with priorities from experimental time-series data $\mathcal{X}'$, so-called $\mathcal{X}'$-deterministic extended Petri nets. For a successful reconstruction the data must be reproducible and monotone. While reproducibility is clearly evident, the necessity of monotone data is shown in [12]. In this paper we give a feasibility test for the data and a strategy for handling infeasible cases.

Firstly, the preprocessing step examines the given experimental time-series data for reproducibility, i.e., it tests if all measured states $\boldsymbol{x} \in \mathcal{X}'$ have a unique successor state (see Section 3.1). If this test is successful we can reconstruct an $\mathcal{X}'$-deterministic extended Petri net (Lemma 1).

Whenever two time-series $\mathcal{X}_i$ and $\mathcal{X}_\ell$ have a common state $\boldsymbol{x}$ but different successor states in each of these sequences (i.e., $\mathrm{succ}_{\mathcal{X}_i}(\boldsymbol{x}) \neq \mathrm{succ}_{\mathcal{X}_\ell}(\boldsymbol{x})$) we have an $\mathcal{X}'$-determinism conflict. Depending on whether the terminal states of these conflicts are equal or not, we have a weak or a strong $\mathcal{X}'$-determinism conflict.

When we encounter a weak $\mathcal{X}'$-determinism conflict we try to linearize the two sequences by the induced order of the successor relation. This is done in the second step of the preprocessing (see Section 3.1).

If linearizing the time-series is not possible or when there are strong $\mathcal{X}'$-determinism conflicts, we cannot reproduce $\mathcal{X}'$-deterministic extended Petri nets (Theorem 1). In this case we extend the data by adding additional components to every state of $\mathcal{X}'$ (see Section 3.2). Finally, in order to compute valid vectors of additional components, we solve an optimization problem.

After having performed the preprocessing step, the reproducibility of the (given or modified) data $\mathcal{X}'$ can be guaranteed such that $\mathcal{X}'$ can serve as appropriate input for the main reconstruction algorithm.

References

1. Chen, M., Hofestädt, W.: Quantitative Petri net model fo gene regulated metabolic networks in the cell. In Silico Biology **3** (2003) 347–365
2. Koch, I., Heiner, M.: Petri nets. In Junker, B.H., Schreiber, F., eds.: Biological Network Analysis. Wiley Book Series on Bioinformatics (2007) 139–179
3. Marwan, W., Wagler, A.K., Weismantel, R.: Petri nets as a framework for the reconstruction and analysis of signal transduction pathways and regulatory networks. Natural Computing **10** (2011) 639–654
4. Pinney, J.W., Westhead, R.D., McConkey, G.A.: Petri net representations in systems biology. Biochem. Soc. Tarns. **31** (2003) 1513–1515
5. Durzinsky, M., Marwan, W., Wagler, A.K.: Reconstruction of extended Petri nets from time-series data by using logical control functions. Journal of Mathematical Biology **66** (2013) 203–223
6. Durzinsky, M., Marwan, W., Wagler, A.K.: Reconstruction of extended Petri nets from time series data and its application to signal transduction and to gene regulatory networks. BMC Systems Biology **5** (2011)

7. Durzinsky, M., Wagler, A.K., Weismantel, R.: An algorithmic framework for network reconstruction. Journal of Theoretical Computer Science **412**(26) (2011) 2800–2815
8. Favre, M., Wagler, A.K.: Reconstructing $\mathcal{X}'$-deterministic extended Petri nets from experimental time-series data $\mathcal{X}'$. In: Preceedings of the 4th International Workshop on Biological Processes & Petri Nets. (2013) 45–59
9. Marwan, W., Wagler, A.K., Weismantel, R.: A mathematical approach to solve the network reconstruction problem. Math. Methods of Operations Research **67**(1) (2008) 117–132
10. Wagler, A.K.: Prediction of network structure. In Koch, I., Schreiber, F., Reisig, W., eds.: Modeling in Systems Biology. Volume 16 of Computational Biology., Springer London (2010) 309–338
11. Wagler, A.K., Wegener, J.T.: On minimality and equivalence of Petri nets. Proceedings of Concurrency, Specification and Programming CS&P'2012 Workshop **2** (2012) 382–393
12. Durzinsky, M., Wagler, A.K., Weismantel, R.: A combinatorial approach to reconstruct Petri nets from experimental data. In Heiner, M., Uhrmacher, A.M., eds.: CMSB. Volume 5307 of Lecture Notes in Computer Science., Springer (2008) 328–346
13. Starostzik, C., Marwan, W.: Functional mapping of the branched signal transduction pathway that controls sporulation in Physarum polycephalum. Photochem Photobiol **62**(5) (1995) 930–933
14. Lamparter, T., Marwan, W.: Spectroscopic detection of a phytochrome-like photoreceptor in the myxomycete Physarum polycephalum and the kinetic mechanism for the photocontrol of sporulation by pfr. Photochem Photobiol **73**(6) (2001) 697–702
15. Torres, L.M., Wagler, A.K.: Encoding the dynamics of deterministic systems. Math. Methods of Operations Research **73** (2011) 281–300

An Holistic State Equation for Timed Petri Nets

Matthias Werner[1], Louchka Popova-Zeugmann[2], Mario Haustein[1], and E. Pelz[3]

[1] Professur Betriebssysteme, Technische Universität Chemnitz
[2] Institut für Informatik, Humboldt-Universität zu Berlin
[3] LACL, UPEC, France

1 Introduction

Timed Petri nets (TPN) or Duration Petri nets (DPN) is a well-know approach to extend "classic" Petri nets in order to allow the modeling of time [1].

In [2], a state equation for TPN was provided that describes the net's marking in an algebraic manner, but not its transitions clocks. Hence, proofing the nonreachability of a marking is mainly done by symbolic manipulation of the state equation, which is impractical for the automated generation of such proofs. Here, we introduce a holistic state equation that allows for modeling the clocks algebraically in addition to markings and thus provides a more automatical way to show the nonreachablity of specific markings.

2 Timed Petri Nets

2.1 Notation

This section introduces the basic notations we use in our paper. $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ denotes the set of natural numbers without 0, and $\mathbb{Q}_0^+$ denotes the set of nonnegative rational numbers. Let S be a finite set. $|S|$ is the number of elements of S. Multisets can contain an element multiple times and are designated by Fraktur letters. The $\uplus$-operator denotes the union of multisets. The number of occurrences of each element in the result of the $\uplus$-operation is given by the sum of the occurrences of this element in both operands. $|\mathfrak{S}|_e$ denotes the multiplicity of e in the multiset $\mathfrak{S}$. $\mathbb{1}_c$ is the indicator function which yields 1 iff the condition c holds or 0 otherwise.

A matrix $\mathbf{A} \in \mathcal{M}(m,n)$ is a matrix with m rows and n columns. A superindex in parentheses distinguishes different matrices or vectors, and their elements, respectively. $\mathbf{Z}_{m\times n} = (z_{i,j}) \in \mathcal{M}(m,n)$ is the zero matrix with $z_{i,j} = 0$ and $\mathbf{E}_n = (e_{i,j}) \in \mathcal{M}(n,n)$ denotes the identity matrix with:

$$e_{i,j} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

The relation $\boldsymbol{r}^{(1)} \leq \boldsymbol{r}^{(2)}$ of the two vectors $\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)} \in \mathcal{M}(m, 1)$ means, that all elements of $\boldsymbol{r}_i^{(1)}$ are less or equal than the corresponding elements of $\boldsymbol{r}_i^{(2)}$. The relation $\boldsymbol{r}^{(1)} \not\leq \boldsymbol{r}^{(2)}$ means, that the above relation does not hold, i.e. there exists at least one $i \in \{1, \ldots, m\}$ with $\boldsymbol{r}_i^{(1)} > \boldsymbol{r}_i^{(2)}$. The relations $<$ and $\not<$ are defined analogously.

2.2 Timed Petri Nets

Definition 1 (Petri net).
The structure $\mathcal{N} = (P, T, F, V, m^{(0)})$ is called a Petri net (PN), iff

1. *P, T are finite sets with $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$,*
2. *$F \subseteq (P \times T) \cup (T \times P)$ (relation between places and transitions),*
3. *$V : F \to \mathbb{N}^+$ (weight of the arcs),*
4. *$m^{(0)} : P \to \mathbb{N}$ (initial marking)*

A marking of a Petri net is a function $m : P \to \mathbb{N}$, such that $m(p)$ denotes the number of tokens at the place p. The pre- and post-sets of a transition t are given by $\bullet t = \{p : (p, t) \in F\}$ and $t\bullet = \{p : (t, p) \in F\}$, respectively. Each transition $t \in T$ induces the marking change t^- and t^+, defined as follows:

$$t^-(p) = \begin{cases} V(p,t) & (p,t) \in F \\ 0 & \text{else} \end{cases} \qquad t^+(p) = \begin{cases} V(t,p) & (t,p) \in F \\ 0 & \text{else} \end{cases}$$

A transition $t \in T$ is enabled (may fire) at a marking m, iff $t^-(p) \leq m(p)$ for every place $p \in P$. When an enabled transition t at a marking m fires, this yields a new marking m' given by $m'(p) := m(p) - t^-(p) + t^+(p)$. The firing is denoted by $m \xrightarrow{t} m'$.

Definition 2 (Timed Petri net).
The structure $\mathcal{Z} = (\mathcal{N}, D)$ is called a Timed Petri net (TPN) iff:

1. *$\mathcal{N}$ (called Skeleton of $\mathcal{Z}$) is a Petri net,*
2. *$D : T \to \mathbb{Q}_0^+$.*

$D(t)$ is the duration of the firing transition t and denotes the delay of t. It is easy to see, that considering TPNs with $D : T \to \mathbb{N}$ will not result in a loss of generality. Therefore, only such time functions D will be considered subsequently.

An active transition t passes through three phases. First it *consumes* tokens from $\bullet t$ which leads to a new marking $m'(p) := m(p) - t^-(p)$. This change takes no time. Then time $D(t)$ passes. During this time, the marking m' may change to m'' by the firing of other transitions. Finally t *delivers* tokens to $t\bullet$ which leads to the marking $m'''(p) := m''(p) + t^+(p)$.

Definition 3 (Maximal step).
Let $z = (m, u)$ be a state in the Timed Petri net $\mathcal{Z} = (P, T, F, V, m_0, D)$. Then $M \subseteq T$ is a maximal step in z, if

1. $\forall t \in M : u(t) = 0$,
2. $\sum\limits_{t \in M} t^- \leq m$,
3. $\forall \hat{t} \in T \setminus M : \hat{t}^- \leq m \wedge u(\hat{t}) = 0 \Longrightarrow \hat{t}^- \not\leq m - \sum\limits_{t \in M} t^-$.

In a Timed Petri net, an enabled transition must fire immediately. In case of non-self-concurrent transitions, a transition can only be enabled, if it is not active at the moment. Please note, immediate transitions are implicitly self-concurrent. In the following we will consider all delayed transitions as not self-concurrent.

Definition 4 (Firing).
Let $z_1 = (m_1, u_1)$ be a state in the Timed Petri net $\mathcal{Z}$ and $M \subseteq T$. Then M can fire in z_1 (notation: $z_1 \xrightarrow{M}$), if M is a maximal step in z_1. After the firing of M the net $\mathcal{Z}$ changes into the state $z_2 = (m_2, u_2)$ (notation: $z_1 \xrightarrow{M} z_2$) with:

$$m_2 := m_1 - \sum_{t \in M} t^- + \sum_{\substack{t \in M \\ D(t)=0}} t^+ \qquad \textit{and} \qquad u_2(t) := \begin{cases} D(t) & t \in M \\ u_1(t) & \textit{else} \end{cases}$$

In a Timed Petri net it is possible that after firing of a maximal step containing transitions with zero delay some transitions are still enabled. For that purpose, we define a global step:

Definition 5 (Global step).
Let z be a state in the Timed Petri net $\mathcal{Z}$. The multiset $\mathfrak{G}$ over T is called a global step in z, that is computed by the following procedure:

1. $\mathfrak{G} := \emptyset$;
2. *Let M be a maximal step in z;*
3. ***if*** $M \neq \emptyset$ ***then*** $\mathfrak{G} := \mathfrak{G} \uplus M$ ***else*** *stop;*
4. *Let $z \xrightarrow{M} z'$; Set $z := z'$;* **goto** *2;*

To ensure finite global steps, we do not allow Timed Petri nets with time deadlocks, i.e., with a closed directed path that contains immediate transitions only.

Definition 6 (Elapsing of time).
Let $z_1 = (m_1, u_1)$ be a state in the Timed Petri net $\mathcal{Z}$. Then, the elapsing of one time unit is possible in $\mathcal{Z}$ (notation: $z_1 \underset{1}{\rightarrow}$), if

$$\forall t \in T : u_1(t) = 0 \Longrightarrow t^- \not\leq m_1$$

After the elapsing of one time unit the Timed Petri net $\mathcal{Z}$ is in the state $z_2 = (m_2, u_2)$ (notation: $z_1 \underset{1}{\rightarrow} z_2$) with:

$$m_2 := m_1 + \sum_{\substack{t \in T \\ u_1(t)=1}} t^+ \qquad \textit{and} \qquad u_2(t) := \begin{cases} u_1(t) - 1 & u_1(t) \geq 1 \\ 0 & \textit{else} \end{cases}$$

3 State Equation

3.1 Structure

Like in classical Petri nets, we describe the structure of a Timed Petri net by two matrices $\mathbf{C}^+, \mathbf{C}^- \in \mathcal{M}(|P|, |T|)$ over the base set $\mathbb{N}$, with $c^+_{i,j} = t^+_j(p_i)$ and $c^-_{i,j} = t^-_j(p_i)$. Furthermore the delay of each transition is encoded in a delay matrix $\mathbf{\Gamma} \in \mathcal{M}(|T|, d)$, where $d = \max\{D(t) : t \in T\} + 1$ specifies the maximum delay. The matrix $\mathbf{\Gamma}$ is given by the following definition:

$$\gamma_{i,j} = \begin{cases} 1 & j = D(t_i) + 1 \\ 0 & \text{else} \end{cases}$$

3.2 Dynamics

The dynamic of a Petri net at each point of time, is unambiguously decribed by its state. In our model, the state consists of two parts, the place marking $\boldsymbol{m}$ and the clock matrix $\mathbf{U}$. The place marking is given by a vector $\boldsymbol{m} \in \mathcal{M}(|P|, 1)$ over $\mathbb{N}$, which specifies how many tokens are on each place. In contrast to classical Petri nets not only the marking is part of the state. The clock matrix $\mathbf{U} \in \mathcal{M}(|T|, d)$ accounts for all active transitions. The element $u_{i,j}$ specifies how often a transition t_i has consumed $D(t_i) - j + 1$ time steps ago, and therefore how often it will deliver in $j - 1$ time steps from now. Thus, the first row of $\mathbf{U}$ states how many times each transition will finish right now. Because a zero delay transition t_k is intrinsically self-concurrent, the value of $u_{k,1}$ may have any value from $\mathbb{N}$. But the row sum of delayed transitions is at most 1, due to the lack of self-concurrency. It is easy to see, that all values $u_{i,j}$ for $j > D(t_i) + 1$ are zero.

To calculate the state reached by a given firing sequence, we have to represent the sequence inside our equation. In classical Petri nets this is done by the Parikh vector. We extend the Parikh vector to the Parikh matrix. The Parikh matrix $\mathbf{\Psi} \in \mathcal{M}(|T|, |T|)$ of a global step $\mathfrak{G}$ is defined by

$$\mathbf{\Psi} = \operatorname{diag}(\psi_1, \ldots, \psi_{|T|}) \qquad \text{with} \qquad \psi_i = |\mathfrak{G}|_{t_i}$$

To use $\boldsymbol{m}$, $\mathbf{U}$ and $\mathbf{\Psi}$ together, three operators $\mathbf{A} \in \mathcal{M}(d, 1)$ (adoption operator), $\mathbf{R} \in \mathcal{M}(d, d)$ (progress operator) and $\mathbf{\Lambda} \in \mathcal{M}(|T|, d)$ (selection operator) are necessary, specified by the corresponding matrices:

$$\mathbf{A} = \begin{pmatrix} 1\ 0 \ldots 0 \end{pmatrix}^{\mathrm{T}} \qquad r_{i,j} = \begin{cases} 1 & i - j = 1 \\ 0 & \text{else} \end{cases} \qquad \lambda_{i,j} = \begin{cases} 1 & j = 1 \\ 0 & \text{else} \end{cases}$$

$\dot{\mathbf{\Psi}} = \mathbf{\Psi} \cdot \mathbf{\Lambda} \cdot \mathbf{A}$ gains the "classical" Parikh vector. The term $\overline{\mathbf{\Psi}}^{(i)}$ later used is an abbreviation of $\sum_{j=1}^{i} \dot{\mathbf{\Psi}}^{(j)}$ and specifies how often each transition has fired until time step i.

3.3 Algebraical Representation of State Changes

Using the definitions of the section above, we can derive an algebraic description for a single state change:

$$\mathbf{U}' = \mathbf{U} + \mathbf{\Psi}\,\mathbf{\Gamma} \qquad \boldsymbol{m}' = \boldsymbol{m} - \mathbf{C}^- \mathbf{\Psi}\,\mathbf{\Lambda}\,\mathbf{A} + \mathbf{C}^+ \mathbf{\Psi}\,\mathbf{\Gamma}\,\mathbf{A} \quad \text{(Firing)} \tag{1}$$

$$\mathbf{U}'' = \mathbf{U}'\,\mathbf{R} \qquad \boldsymbol{m}'' = \boldsymbol{m}' + \mathbf{C}^+ \mathbf{U}'\,\mathbf{R}\,\mathbf{A} \quad \text{(Time elapsing)} \tag{2}$$

To apply Equations (1) and (2) to arbitrary sequences, we introduce time indices to $\boldsymbol{m}$ and $\mathbf{U}$ and mark the results of Equation (1) by a hat and the results of Equation (2) by a tilde. Firing steps and timing steps are dennoted by

$$(\tilde{\boldsymbol{m}}^{(i)}, \tilde{\mathbf{U}}^{(i)}) \xrightarrow{\mathfrak{G}^{(i)}} (\hat{\boldsymbol{m}}^{(i)}, \hat{\mathbf{U}}^{(i)}) \quad \text{and} \quad (\hat{\boldsymbol{m}}^{(i)}, \hat{\mathbf{U}}^{(i)}) \underset{1}{\longrightarrow} (\tilde{\boldsymbol{m}}^{(i+1)}, \tilde{\mathbf{U}}^{(i+1)})$$

respectively. Every firing sequence σ can then be represented by alternating firing and time elapsing steps, where some of the $\mathfrak{G}$ can be empty sets of course:

$$\sigma : (\tilde{\boldsymbol{m}}^{(1)}, \tilde{\mathbf{U}}^{(1)}) \xrightarrow{\mathfrak{G}^{(1)}} (\hat{\boldsymbol{m}}^{(1)}, \hat{\mathbf{U}}^{(1)}) \underset{1}{\longrightarrow} \cdot \xrightarrow{\mathfrak{G}^{(2)}} \cdots \underset{1}{\longrightarrow} \cdot \xrightarrow{\mathfrak{G}^{(i)}} (\hat{\boldsymbol{m}}^{(i)}, \hat{\mathbf{U}}^{(i)})$$

The term $\tilde{\boldsymbol{m}}^{(1)}$ is equal to the initial marking $\boldsymbol{m}^{(0)}$ and $\tilde{\mathbf{U}}^{(1)}$ is the zero matrix $\mathbf{Z}_{|T|\times d}$, because no transition is active in the initial state.

3.4 State equation

We now consider the actual state equation. From Equations (1) and (2) we can derrive the following equations on $\hat{\boldsymbol{m}}^{(i)}$ and $\hat{\mathbf{U}}^{(i)}$ for the sequence σ.

$$\hat{\mathbf{U}}^{(i)} = \sum_{j=1}^{i} \mathbf{\Psi}^{(j)}\,\mathbf{\Gamma}\,\mathbf{R}^{i-j} \tag{3}$$

$$\begin{aligned}\hat{\boldsymbol{m}}^{(i)} &= \boldsymbol{m}^{(0)} + \mathbf{C}^+ \left(\sum_{j=1}^{i} \mathbf{\Psi}^{(j)}\,\mathbf{\Gamma} \sum_{k=0}^{i-j} \mathbf{R}^k \right) \mathbf{A} - \mathbf{C}^- \overline{\boldsymbol{\Psi}}^{(i)} \\ &= \boldsymbol{m}^{(0)} + \mathbf{C}^+ (\psi_k^{(i-D(t_k))} + \cdots + \psi_k^{(1)})_k - \mathbf{C}^- \overline{\boldsymbol{\Psi}}^{(i)}\end{aligned} \tag{4}$$

With defining $\underline{\boldsymbol{\Psi}}^{(i)} := (\psi_k^{(i-D(t_k))} + \cdots + \psi_k^{(1)})_k$, Equation (4) can be expressed in a more compact way:

$$\hat{\boldsymbol{m}}^{(i)} = \boldsymbol{m}^{(0)} + \mathbf{C}^+ \underline{\boldsymbol{\Psi}}^{(i)} - \mathbf{C}^- \overline{\boldsymbol{\Psi}}^{(i)} \tag{5}$$

Let $\hat{\boldsymbol{\Upsilon}}^{(i)} := \hat{\mathbf{U}}^{(i)} \cdot \begin{pmatrix} 0 & 1 & \cdots & 1 \end{pmatrix}^{\mathrm{T}}$ dennote all transition which are active immediately after the zero delay transitions in firing step i have delivered. Since each delayed transition can only be active once at each point of time, only clock matrices $\hat{\mathbf{U}}^{(i)}$ are valid, which fulfill the following constraint:

$$\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^{\mathrm{T}} \overset{!}{\geq} \hat{\mathbf{U}}^{(i)} \cdot \begin{pmatrix} 0 & 1 & \cdots & 1 \end{pmatrix}^{\mathrm{T}} = \hat{\boldsymbol{\Upsilon}}^{(i)}$$

By substituting $\hat{\mathbf{U}}^{(i)}$ in the former equation with the right side of Equation (3), we get:

$$\begin{pmatrix}1 \cdots 1\end{pmatrix}^{\mathrm{T}} \overset{!}{\geq} (\psi_k^{(i-D(t_k)+1)} + \cdots + \psi_k^{(i)})_k = \hat{\boldsymbol{\Upsilon}}^{(i)} \tag{6}$$

Because the inner term of the former equation is zero for immediate transitions, it follows that $\hat{\boldsymbol{\Upsilon}}^{(i)}$ must be an element of $B_1 \times \cdots \times B_{|T|}$, with

$$B_k = \begin{cases} \{0,1\} & D(t_k) > 0 \\ \{0\} & \text{else} \end{cases}$$

The right part of Equation (6) helps reformulating Equation (4) into

$$\hat{\boldsymbol{m}}^{(i)} = \boldsymbol{m}^{(0)} + (\mathbf{C}^+ - \mathbf{C}^-) \cdot \underline{\boldsymbol{\Psi}}^{(i)} - \mathbf{C}^- \, \hat{\boldsymbol{\Upsilon}}^{(i)} \tag{7}$$

The vector $\underline{\boldsymbol{\Psi}}^{(i)}$ sums up all transition which have completed until time step i. It is easy to see that $\underline{\boldsymbol{\Psi}}^{(i)} + \hat{\boldsymbol{\Upsilon}}^{(i)}$ yields the Parikh vector $\overline{\boldsymbol{\Psi}}^{(i)}$. An equation for $\tilde{\boldsymbol{m}}^{(i)}$ can be obtained by applying Equations (2) to Equation (7):

$$\begin{aligned}
\tilde{\boldsymbol{m}}^{(i)} &= \hat{\boldsymbol{m}}^{(i-1)} + \mathbf{C}^+ \, \hat{\mathbf{U}}^{(i-1)} \, \mathbf{R} \, \mathbf{A} \\
&= \hat{\boldsymbol{m}}^{(i-1)} + \mathbf{C}^+ \, \hat{\mathbf{U}}^{(i-1)} \cdot \begin{pmatrix}0\,1\,0 \cdots 0\end{pmatrix}^{\mathrm{T}} \\
&= \hat{\boldsymbol{m}}^{(i-1)} + \mathbf{C}^+ \, \hat{\mathbf{U}}^{(i-1)} \cdot \begin{pmatrix}0\,1\,1 \cdots 1\end{pmatrix}^{\mathrm{T}} - \mathbf{C}^+ \, \hat{\mathbf{U}}^{(i-1)} \cdot \begin{pmatrix}0\,0\,1 \cdots 1\end{pmatrix}^{\mathrm{T}} \\
&= \hat{\boldsymbol{m}}^{(i-1)} + \mathbf{C}^+ \, \hat{\mathbf{U}}^{(i-1)} \cdot \begin{pmatrix}0\,1 \cdots 1\end{pmatrix}^{\mathrm{T}} - \mathbf{C}^+ \, \tilde{\mathbf{U}}^{(i)} \cdot \begin{pmatrix}0\,1 \cdots 1\end{pmatrix}^{\mathrm{T}} \\
&= \hat{\boldsymbol{m}}^{(i-1)} + \mathbf{C}^+ \, \hat{\boldsymbol{\Upsilon}}^{(i-1)} - \mathbf{C}^+ \, \tilde{\boldsymbol{\Upsilon}}^{(i)} \qquad (8) \\
&= \boldsymbol{m}^{(0)} + (\mathbf{C}^+ - \mathbf{C}^-) \cdot (\underline{\boldsymbol{\Psi}}^{(i-1)} + \hat{\boldsymbol{\Upsilon}}^{(i-1)}) - \mathbf{C}^+ \, \tilde{\boldsymbol{\Upsilon}}^{(i)} \qquad (9)
\end{aligned}$$

The vector $\tilde{\boldsymbol{\Upsilon}}^{(i)} := \tilde{\mathbf{U}}^{(i)} \cdot \begin{pmatrix}0\,1 \cdots 1\end{pmatrix}^{\mathrm{T}}$ specifies all transition which are in progress after the i-th time step has finished. This vector is elementwise less or equal than $\hat{\boldsymbol{\Upsilon}}^{(i-1)}$, because we have to subtract the elements of the second column of $\hat{\mathbf{U}}^{(i-1)}$ to yield $\tilde{\boldsymbol{\Upsilon}}^{(i)}$. Furthermore $\tilde{\boldsymbol{\Upsilon}}^{(i)}$ is elementwise less or equal than $\hat{\boldsymbol{\Upsilon}}^{(i)}$, which follows directly from Equation (1). Obviously no element of the $\boldsymbol{\Upsilon}$-vectors can be negative. In case of a transition t_k with delay less than 2, the k-th element of $\tilde{\boldsymbol{\Upsilon}}^{(i)}$ is zero in any case, because due to the construction of $\boldsymbol{\Gamma}$ only the first or the second element of the k-th row of $\hat{\boldsymbol{U}}^{(i-1)}$ can be non-zero. Consequently it holds:

$$(0)_k \overset{!}{\leq} \tilde{\boldsymbol{\Upsilon}}^{(i)} \overset{!}{\leq} \begin{Bmatrix} \hat{\boldsymbol{\Upsilon}}^{(i-1)} \\ \hat{\boldsymbol{\Upsilon}}^{(i)} \end{Bmatrix} \overset{!}{\leq} (\mathbb{1}_{D(t_k)>0})_k \qquad \text{and} \qquad \tilde{\boldsymbol{\Upsilon}}^{(i)} \overset{!}{\leq} (\mathbb{1}_{D(t_k)>1})_k \tag{10}$$

4 Non-Reachability and Application Example

State equations provide a criterion to decide whether a given marking is not reachable in a specified Petri net. When the state equation does not have a solution, the marking is not reachable. In case of Timed Petri nets, Equations (7)

and (9) as well es the constraint Equations (10) form a system of diophantene (in-)equalities. If this system does not have a valid solution, a given $\hat{\boldsymbol{m}}$-marking and $\tilde{\boldsymbol{m}}$-marking, resp., is not reachable. A solution is valid if and only if the Parikh vector candidate does not contain a negative element. Furthermore we can draft on predecessor markings and maximal step conditions to further discard some valid solutions of the state equation.

In this section we show how the state equation in the recent section can be used for a more systematical nonreachability proof of the example given in [2]. Consider the following Timed Petri net $\mathcal{Z}$:

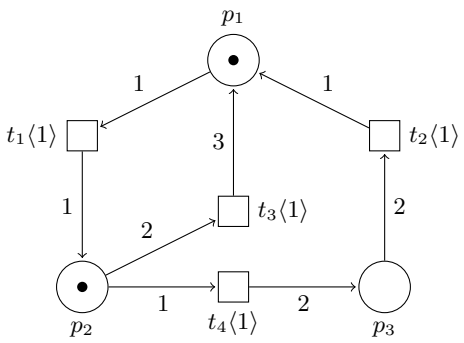


Fig. 1. Petri net from [2]; transition times are given in angle bracket

We want to show that the marking $\boldsymbol{m}^* = \begin{pmatrix}0\ 2\ 0\end{pmatrix}^{\mathrm{T}}$ is not reachable. To do so, one has to show that both $\tilde{\boldsymbol{m}}^{(i)} = \boldsymbol{m}^*$ and $\hat{\boldsymbol{m}}^{(i)} = \boldsymbol{m}^*$ are not reachable in $\mathcal{Z}$. Lets consider the $\tilde{\boldsymbol{m}}^{(i)}$-case first. First we have to determine all $\boldsymbol{\Psi}^{(i-1)} + \hat{\boldsymbol{\Upsilon}}^{(i-1)}$ which solve Equation (9). The solution space $\mathcal{S}_t$ depends on $\tilde{\boldsymbol{\Upsilon}}^{(i)}$ and can be calculated by the Gauss-Algorithm:

$$\mathcal{S}_t(\tilde{\boldsymbol{\Upsilon}}^{(i)}) = \left\{ \begin{pmatrix}-1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}3&2&6&2\\0&0&0&-1\\1&1&3&1\\0&0&0&0\end{pmatrix} \cdot \tilde{\boldsymbol{\Upsilon}}^{(i)} + k \cdot \begin{pmatrix}1\\1\\0\\1\end{pmatrix} : k \in \mathbb{Z} \right\}$$

In this special case it follows from Equation (10) that $\tilde{\boldsymbol{\Upsilon}}^{(i)}$ must be the zero vector, so only one solution space has to be considered during the further calculation. As stated, a valid solution cannot contain a negative component. Thus, we can rule out all $\hat{\boldsymbol{\Upsilon}}^{(i-1)}$ for which the set $\mathbb{N}^{|T|} \cap \{\boldsymbol{S} - \hat{\boldsymbol{\Upsilon}}^{(i-1)} : \boldsymbol{S} \in \mathcal{S}_t\}$ is empty. Algorithmically this problem can be decided by integer linear programming techniques. In our example at least $\tilde{v}_3^{(i-1)}$ must be zero, which narrows the set of possible $\hat{\boldsymbol{\Upsilon}}^{(i-1)}$ down to eight distinct cases, shown in the following table. With the aid of Equation (8), we can calculate the corresponding $\hat{\boldsymbol{m}}^{(i-1)}$:

candidates for $\hat{\boldsymbol{\Upsilon}}^{(i-1)}$	$\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\\0\\1\end{pmatrix}$
corresp. $\hat{\boldsymbol{m}}^{(i-1)}$	$\begin{pmatrix}0\\2\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\end{pmatrix}$	$\begin{pmatrix}-1\\2\\0\end{pmatrix}$	$\begin{pmatrix}-1\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\2\\-2\end{pmatrix}$	$\begin{pmatrix}0\\1\\-2\end{pmatrix}$	$\begin{pmatrix}-1\\2\\-2\end{pmatrix}$	$\begin{pmatrix}-1\\1\\-2\end{pmatrix}$

Because places cannot contain a negative amount of tokens, we can rule out the last six cases. The remaining two cases can be discarded by aid of Definition 3. According to the definition of a maximal step for all active transitions t_k, i.e., all transitions t_k for which $\hat{v}_k = 0$, it must hold $\hat{\boldsymbol{m}} \not\geq t_k^-$. Now we consider t_4 which is active in both cases. The maximal step condition $\hat{\boldsymbol{m}}^{(i-1)} \not\geq \begin{pmatrix}0\,1\,0\end{pmatrix}^{\mathrm{T}}$ resulting from t_4 is not fulfilled. Thus, the two remaining solutions are not valid in $\mathcal{Z}$. Consequently $\boldsymbol{m}^*$ cannot be reached as a $\tilde{\boldsymbol{m}}$-marking.

Now, we show $\boldsymbol{m}^*$ is not reachable as a $\hat{\boldsymbol{m}}$-marking. First, we calculate the set $\mathcal{Y}$ of possible $\hat{\boldsymbol{\Upsilon}}^{(i)}$ by using the definition of the maximal step:

$$\mathcal{Y}(\hat{\boldsymbol{m}}^{(i)}) = \{\hat{\boldsymbol{\Upsilon}} : (\forall k : \hat{v}_k = 0 \Longrightarrow \hat{\boldsymbol{m}}^{(i)} \not\geq t_k^-)\}$$

In our example, at least $\hat{v}_3^{(i)} = 1$ and $\hat{v}_4^{(i)} = 1$, otherwise the maximal step condition for t_3 and t_4, resp., would not be fulfilled. Then we calculate the solution set $\mathcal{S}_f$ of Equation (7).

$$\mathcal{S}_f(\hat{\boldsymbol{\Upsilon}}^{(i)}) = \left\{ \begin{pmatrix}-1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}2&2&6&3\\0&-1&0&0\\1&1&2&1\\0&0&0&0\end{pmatrix} \cdot \tilde{\boldsymbol{\Upsilon}}^{(i)} + k \cdot \begin{pmatrix}1\\1\\0\\1\end{pmatrix} : k \in \mathbb{Z} \right\}$$

In the example every possible $\hat{\boldsymbol{\Upsilon}}^{(i)}$ yields at least one Parikh vector $\underline{\boldsymbol{\Psi}}^{(i)}$. From $\hat{\boldsymbol{m}}^{(i)}$ we can calculate $\tilde{\boldsymbol{m}}^{(i)}$ by:

$$\tilde{\boldsymbol{m}}^{(i)} = \hat{\boldsymbol{m}}^{(i)} + \mathbf{C}^+ \tilde{\boldsymbol{\Upsilon}}^{(i)} - \mathbf{C}^+ \hat{\boldsymbol{\Upsilon}}^{(i)}$$

Due to Equation (10) the vector $\tilde{\boldsymbol{\Upsilon}}^{(i)}$ must be the zero vector in this example. Consequently we can calculate $\tilde{\boldsymbol{m}}^{(i)}$ without further case discriminations on $\tilde{\boldsymbol{\Upsilon}}^{(i)}$:

$\hat{\boldsymbol{\Upsilon}}^{(i)}$	$\begin{pmatrix}0\\0\\1\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\\1\\1\end{pmatrix}$	$\begin{pmatrix}0\\1\\1\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$
corresp. $\tilde{\boldsymbol{m}}^{(i)}$	$\begin{pmatrix}-3\\2\\-2\end{pmatrix}$	$\begin{pmatrix}-3\\1\\-2\end{pmatrix}$	$\begin{pmatrix}-4\\2\\-2\end{pmatrix}$	$\begin{pmatrix}-4\\1\\-2\end{pmatrix}$

This leads to an invalid solution for the marking in each case. Thus, $\boldsymbol{m}^*$ is not reachable as a $\hat{\boldsymbol{m}}$-marking, too.

5 Conclusion

We have presented a new state equation for Timed Petri nets. In contrast to [2] the new equation is a holistic one: It describes the marking as well as the clocks, whereas [2] has dealt with the net's marking only.

Also, we have demonstrated in an example how the new state equation can be used to prove non-reachability.

References

1. Ramchandani, C.: Analysis of asynchronous concurrent systems by Timed Petri Nets. Project MAC-TR 120, MIT (February 1974)
2. Popova-Zeugmann, L., Werner, M., Richling, J.: Using state equation to prove non-reachability in timed petrinets. Fundam. Inf. **55**(2) (August 2002) 187–202
3. Popova-Zeugmann, L., Pelz, E.: Algebraical characterisation of interval-timed petri nets with discrete delays. Fundamenta Informaticae **120**(3–4) (2012) 341–357

Query Rewriting Based on Meta-Granular Aggregation

Piotr Wiśniewski[1] and Krzysztof Stencel[2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Toruń, Poland
pikonrad@mat.uni.torun.pl
[2] Institute of Informatics
The University of Warsaw
Warsaw, Poland
stencel@mimuw.edu.pl

Abstract. Analytical database queries are exceptionally time consuming. Decision support systems employ various execution techniques in order to accelerate such queries and reduce their resource consumption. Probably the most important of them consists in materialization of partial results. However, any introduction of additional derived objects into the database schema increases the cost of software development, since programmers must take care of their usage and synchronization. In this paper we propose novel query rewriting methods that build queries using partial aggregations materialized in additional tables. These methods are based on the concept of meta-granules that represent the information on grouping and used aggregations. Meta-granules have a natural partial order that guides the optimisation process. We also present an experimental evaluation of the proposed rewriting method.

1 Introduction

In the article [1] we presented the idea of materializing partial aggregations in order to accelerate analytical queries. The inherent cost of this idea is attributed to the need of database triggers that keep the materializations up to date in real time. In particular we showed an application programming interface that facilitates defining and using partial aggregations. We designed and implemented appropriate mechanisms that automatically create necessary triggers.

The solutions presented in [1] suffer from a noteworthy deficiency. The application programmer is obliged to cater for additional database objects that store materialized data. He/she not only has to create them, but also has to address them through API methods in order to use them in analytical queries. In particular, it is impossible to use these facilities through queries formulated in HQL, i.e. the standard query language of Hibernate ORM [2].

In this paper we address the abovementioned deficiency. We present an algorithm to rewrite HQL queries so that the usage of materialized aggregations is transparent to application programmers.

This algorithm is based on analyses of the grouping granularity and opportunities to reconstruct necessary data from partial aggregations that are persisted in the database. In order to control the complexity of the space of aggregations that can possibly be materialized, we introduce the notion of a *meta-granule*. It represents a potentially interesting aggregation level. The set of meta-granules is partially ordered. The rewrite method efficiently analyses and traverses the graph of this partial order.

Our solution is based on the idea of materialized views. A recent example of an implementation of such views are *FlexViews* [3] within MySQL based on the results described in [4, 5]. FlexViews rely on applying changes that have been written to the change log.

This article is organized as follows. In Section 2 we recall the idea of partial aggregations and introduce the running example used thorough the paper. We also discuss the integration of the prototype with Hibernate, a major object-relational mapping system. In Section 3 we formalize meta-granules and their partial order. In Section 4 we introduce the query rewrite algorithm that utilizes meta-granules. Section 5 summarizes the results of our experimental evaluation of possible gains triggered by the proposed optimization algorithm. Section 6 concludes.

2 Partial aggregation

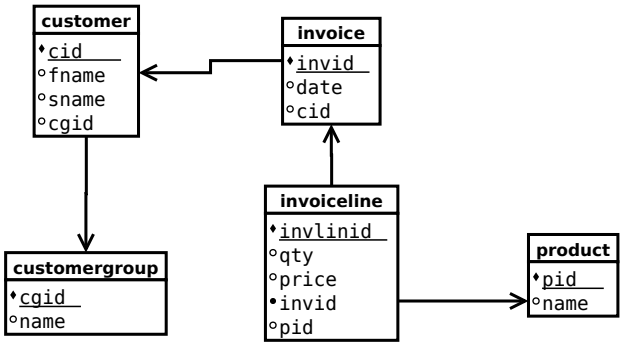


Fig. 1. The original schema of the database on customers and invoices.

Example 1. Let us consider a database schema on customers and their invoices as show on Figure 1. Assume that the company database often has to answer queries that follow the pattern of the SELECT statement presented below.

```
SELECT invoiceline.pid, invoice.date,
       sum(invoiceline.qty)
  FROM invoice JOIN invoiceline USING (invid)
```

```
 GROUP BY invoice.date, invoiceline.pid
HAVING date BETWEEN '2011-07-16' AND '2011-07-22'
```

In order to serve such queries efficiently we extend the database schema from Figure 1 by adding the derived table `dw_invline_value_by_customer_date` as shown on Figure 2. These tables will store partial sums that are needed to quickly answer the query shown above.

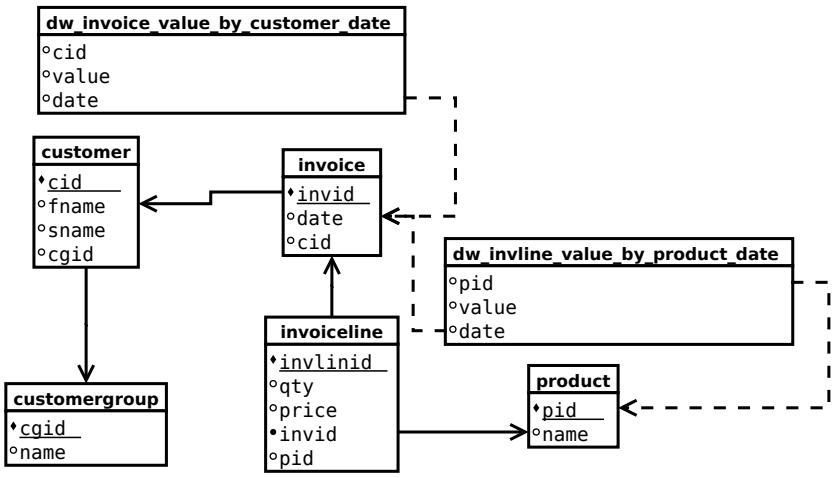


Fig. 2. The schema of the database on customers and invoices extended with derived tables that store materialized aggregations.

In our research we exploit the possibility to hide internals of optimization algorithms in the layers of the object-relational mapping [6]. In our opinion this additional layer of abstraction is a perfect place to put disparate peculiarities of optimisation algorithms. The database server is not the only place to implement query rewriting. Moreover, this ORM option is sometimes the only one, e.g. when the database system is not open-source. It also facilitates writing reusable code that does not depend on the SQL dialect of a particular DBMS. We applied this approach successfully to recursive querying [7] and index selection [8].

Thanks to the generators hidden in the object-relational mapping layer, the extra tables from Figure 2 will be created automatically. For the application programmer it is enough to augment the declaration of the Java entity class `InvoiceLine` with the annotation shown on Listing 1.1.

Listing 1.1. Java class `InvoiceLine` with annotations that cause generation of materialized aggregations

```
@Entity
public class Invoiceline {
    ...
```

```
@DWDim(Dim = "date")
private Invoice invoice;
private Long invlinid;
@DWDim
private Product product;
@DWAgr(function="SUM")
private Integer qty;
...
```

When the materialized aggregations are computed and stored in the database, DBMS can execute the following query using derived storage objects instead of the original user query. The modified query will be served significantly faster since it addresses pre-aggregated data.

```
SELECT pid, date, value
 FROM dw_invline_value_by_customer_date
 WHERE date BETWEEN '2011-07-16' AND '2011-07-22'
```

If we add an appropriate annotation to the entity class `Invoice`, the materialized aggregation `dw_invoice_value_by_customer_date` can be automatically generated. Figure 2 shows this derived table as well. It allows for a notable acceleration of preparation of reports on sales partitioned by dates and customers.

3 Granularity and meta-granules

In this paper *granularity* is the partitioning implied by the grouping clause, while *meta-granules* are schema items that unambiguously define this partition. Basic meta-granules are tables with data we want to summarize, e.g. `invoice_line, unit` is a meta-granule that represents individual rows of the table `invoice_line`. The meta-granule `invoice_line, product, date` stands for the partitioning formalized in our example query. Let us assume that we want to get the total sales for a particular day. For an application programmer it is obvious that instead of base data we can use existing materialized aggregations.

A similar meta-granule `invoice_line, customer, date` describes grouping by the customer and the date of sale. For this meta-granule we created a materialized aggregation as shown of Figure 2. If a user now poses a query for the total sales on a given day, we can choose among two meta-granules that can accelerate his/her query.

Let us introduce a partial order of meta-granules. A meta-granule g_1 is smaller or equal than a meta-granule g_2, if and only if each row in g_2 can be computed by aggregating some rows of g_1.

In the analysis of our running examples we will use the following symbols of meta-granules:

$$g_{il} = \texttt{invoice\_line: unit}$$
$$g_{pd} = \texttt{invoice\_line: product, date}$$

$$g_{cd} = \texttt{invoice\_line: customer, date}$$
$$g_{d} = \texttt{invoice\_line: date}$$

Then, the following inequalities are satisfied:

$$g_{il} \leq g_{pd} \leq g_{d}$$
$$g_{il} \leq g_{cd} \leq g_{d}$$

On the other hand, the meta-granules g_{pd} and g_{cd} are incomparable. Thus, the partial order of meta-granules is not linear.

Let us analyze another example queries that allow identifying other meta-granules.

```
SELECT invoice.date, cg.name
       sum(invoiceline.qty)
  FROM customergroup cg JOIN customer USING(cgid)
    JOIN invoice USING(cid)
    JOIN invoiceline USING (invid)
  GROUP BY invoice.date, cg.name
  HAVING date = '2011-07-16'
```

This query induces the following meta-granule:

$$g_{cgm} = \texttt{invoice\_line: customer\_group, month(date)}$$

```
SELECT month(invoice.date), product.pid
       sum(invoiceline.qty)
  FROM invoice JOIN invoiceline USING (invid)
    JOIN product USING (pid)
  GROUP BY month(invoice.date), product.pid
```

This query induces the following meta-granule:

$$g_{pm} = \texttt{invoice\_line: product, month(date)}$$

```
SELECT invid, sum(invoiceline.qty)
  FROM invoice JOIN invoiceline USING (invid)
  GROUP BY invid
  HAVING date = '2011-07-16'
```

This query induces the meta-granule:

$$g_{i} = \texttt{invoice\_line: invoice}$$

Our extended schema presented on Figure 2 does not contain these meta-granules. If a meta-granule is associated with a materialized aggregation stored in the database, this meta-granule will be called *proper*. Otherwise, the meta-granule is called *virtual*.

Figure 3 shows the partial order of all meta-granules enumerated in presented examples. Virtual meta-granules are depicted as rectangles, while proper metgranules are portrayed as ovals. Observe that each meta-granule is bigger or equal to the proper meta-granule of basic facts, i.e. g_{il}. Data in all meta-granules is derived from g_{il} by some aggregation.

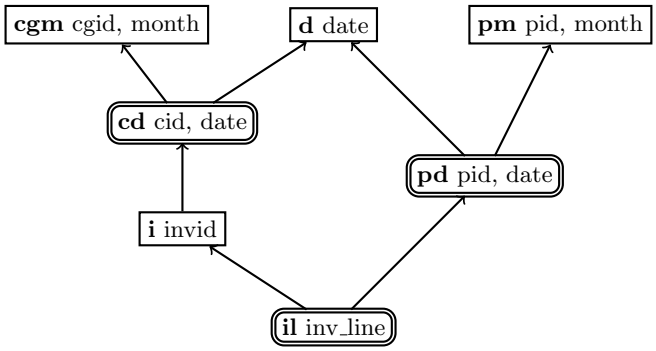


Fig. 3. The partial order of meta-granules

4 The rewriting algorithm

The optimization method based on meta-granules is composed of the following steps. Assume that a query has been posed.

1. We identify the needed meta-granule. We check if the query has internal WHERE clause. Next, we analyze the grouping used and we compute the meta-granule required to compute the answer to the query.
2. If this meta-granule is proper, the query will get rewritten so that it uses the materialization associated with the meta-granule instead of the base fact table.
3. If this meta-granule is virtual, we will find the maximal proper meta-granule not greater than the meta-granule of the query. This maximal meta-granule will be used in the rewriting of the original query.

Since the set of all meta-granules is finite, the set of meta-granules smaller that the given virtual meta-granule always contains maximal elements. This set is never empty, since there exists basic fact meta-granule that is smaller than any mete-granule. However, there can be more than one maximal meta-granule in this set. Let us consider the following example query:

```
SELECT invoice.date, sum(invoiceline.qty)
  FROM invoice JOIN invoiceline USING (invid)
  GROUP BY invoice.date, product.pid
```

This query induces the metagranule:

$$g_d = \texttt{invoice\_line: date}$$

For this virtual meta-granule we have two maximal proper meta-granules. They are g_{cd} and g_{pd}. The usage of any of them means a significant acceleration of the execution of this query.

5 Experimental evaluation

In this Section we show the potential gains of using the optimization algorithm proposed in this paper. We used a computer with Pentium G2120 3.1 GHz (Ivy Brigde), 8 GB RAM. The disk was 120 GB SDD SATA III for the system and Raid 0 on 2x Caviar Black 1 TB 7400rpm for the database storage. We used plain PostgreSQL 9.1 installed on Ubuntu 13.04. No upfront optimization or tuning has been performed. The tested database have the schema shown on Figure 1. The volumes of data are summarized by Table 1.

Table 1. Row counts of tables from the example schema

Table name	Row count
customer	4 999 000
customergroup	50 000
invoice	99 973 000
invoiceline	1 049 614 234
product	9 000

We use three databases instances with different sets of proper meta-granules from Figure 3. The database *plain* contains only the basic meta-granule with invoice lines. The database *med* additionally has proper meta-granules **cd** and **pd**. The database *full* materializes all meta-granules from Figure 3. Table 2 recaps proper meta-granules of all these three databases. It also shows their sizes in gigabytes.

Table 2. Proper meta-granules and volumes of tested database instances

Database name	Proper meta-granules	Volume
plain	`il`	101 GB
med	`cd`, `il`, `pd`	110 GB
full	`cd`, `cgm`, `d`, `i`, `il`, `pd`, `pm`	120 GB

5.1 Rewriting queries

Let us start from a simple query that returns 10 customer groups with biggest sales in March 2006. This query can be formulated as follows assuming the schema from Figure 2.

```
SELECT cgid, sum(price * qty) as sum_val,
       extract(year FROM date) as year,
       extract(month FROM date) as month
```

```
  FROM invline JOIN inv USING (invid)
               JOIN cust USING (cid)
  GROUP BY cgid, year, month
  HAVING extract(year FROM date) = 2006
     AND extract(month FROM date) = 3
  ORDER BY sum_val DESC
  LIMIT 10;
```

Since in the database *plain* no non-trivial proper meta-granule is available, out rewriting algorithm cannot modify this query. When run in this form, it finishes in 1 067.5 seconds.

However, in the database *med* we have the meta-granule **cd** at our disposal. It contains data pre-aggregated by `cid` and `date`. Therefore, our algorithm rewrites the query to use this proper meta-granule:

```
SELECT cgid, sum(sum_val) as cgmsum_val,
       extract(year FROM date) as year,
       extract(month FROM date) as month
  FROM aggr_cd JOIN cust USING (cid)
  GROUP BY cgid, year, month
  HAVING extract(year FROM date) = 2006
     AND extract(month FROM date) = 3
  ORDER BY cgmsum_val DESC
  LIMIT 10;
```

Now the query will run 41.6 seconds. We have accelerated this query 25 times. Although, the database *med* does not contain all possible meta-granules, the running time has been significantly reduced. Of course further increase of efficiency is possible if we have even more proper meta-granules as in the database *full*. In this case, the algorithm will choose using the meta-granule `cgm` to get the following query that runs in half a second.

```
SELECT cgid, sum_val, year, month
  FROM aggr_cgm
  WHERE year = 2006
    AND month = 3
  ORDER BY sum_val DESC
  LIMIT 10
```

In the second example query we ask for 15 best sale days in 2005. This query can be formulated as shown below assuming the schema from Figure 2. Only in this form it can be run against the database *plain* that has no proper meta-granules. It takes 3 475.1 seconds to complete its execution.

```
SELECT date, sum(price * qty) as sum_val
  FROM invline JOIN inv USING (invid)
  GROUP BY date
```

```
  ORDER BY sum_val DESC
  LIMIT 15
```

With the database *med* the optimiser has two options. It can use either the meta-granule **cd** or **pd** as presented below. It takes 56.2 sec to completion with **cd** and 19.9 sec with **pd**. Since both queries are plain SQL, the cost model of the underlying database should be used to determine the query to be run. In this case we have two maximal meta-granules to choose.

```
SELECT date, sum(sum_val) as dsum_val
  FROM aggr_cd
  GROUP BY date
  ORDER BY dsum_val DESC
  LIMIT 15;

  SELECT date, sum(sum_val) as dsum_val
  FROM aggr_pd
  GROUP BY date
  ORDER BY dsum_val DESC
  LIMIT 15;
```

When we have all possible proper meta-granules and in the database *full* we can use the meta-granules **d** and run the following query in 41 milliseconds.

```
SELECT date, sum_val
  FROM aggr_d
  ORDER BY sum_val DESC
  LIMIT 15;
```

The third query is to list 10 best selling products together with the sold volume from September to December 2006. In absence of proper meta-granules it can be formulated as follows. In this form for the database *plain* this query runs for 1 074 seconds.

```
SELECT pid, sum(qty) as sum_qty,
       sum (price * qty) as sum_val
  FROM invline JOIN inv USING (invid)
  WHERE extract(year FROM date) = 2006
     AND extract(month FROM date) between 9 and 12
  GROUP BY pid
  ORDER BY sum_val DESC
  LIMIT 10;
```

In the database *med* we can employ the proper meta-granule **pd** and get the following query that completes in 38.8 seconds.

```
SELECT pid, sum(sum_qty) as pmsum_qty,
       sum (sum_val) as pmsum_val
  FROM aggr_pd
  WHERE extract(year FROM date) = 2006
     AND extract(month FROM date) between 9 and 12
  GROUP BY pid
  ORDER BY pmsum_val DESC
  LIMIT 10;
```

The abundant meta-granules of the database *full* allow executing the following query instead. It finishes in 1282 miliseconds.

```
SELECT pid, sum(sum_qty) as pmsum_qty,
       sum (sum_val) as pmsum_val
  FROM aggr_pm
  WHERE year = 2006
     AND month between 9 and 12
  GROUP BY pid
  ORDER BY pmsum_val DESC
  LIMIT 10;
```

The results of the tests are summarized in Table 3. Obviously, the usage appropriate proper meta-granules significantly accelerates the queries. However, even when only limited subset of meta-granules is proper, our rewriting algorithm can notably boost the query execution. This is the case of the database *med*. Although the optimization algorithm did not have the optimal meta-granule, it could successfully employ the meta-granules that were at its disposal.

Table 3. Summary of query execution times for tested database instances

Database	Query 1	Query 2		Query 3
plain	1 067.5 s	3 475.1 s		1 074.2 s
med	41.6 s	56.2 s	19.9 s	38.8 s
full	0.5 s	0.04 s		0.001 s

5.2 Preparing meta-granules

As usual, keeping derived data structures in sync with the base data induces a significant overhead. In this Section we show experiments that assess this overhead. We performed inserting a number of invoices into each database instance. On average each invoice contained 10 lines. We performed two subsequent runs with 10 000 invoices and one run with 15 000 invoices and one with 20 000 invoices. Before each run (but the second with 10 000 invoices) the database management system was shutdown in order to make the database buffer cold

initially. Table 4 summarizes the run times. As we can see, avoiding creation of some proper meta-granules (compare *med* with *full*) spares a noteworthy amount of time.

Table 4. Time spent on inserting new invoices and synchronizing proper meta-granules

Invoices	Buffer	*plain*	*med*	*full*
10 000	cold	2m 58.335s	29m 06.670s	37m 56.663s
10 000	hot	3m 03.308s	9m 05.212s	13m 01.924s
15 000	cold	4m 30.261s	20m 59.395s	36m 30.021s
20 000	cold	6m 32.502s	29m 59.064s	44m 38.321s

6 Conclusions

In this paper we presented a novel method to select materialized data in analytical query execution. A fact table can be pre-aggregated for numerous sets of its dimensions. We call this sets of dimensions meta-granules and introduce their partial order. "Bigger" meta-granules are more aggregated, i.e., contain less specific data. Whenever an *ad-hoc* query is posed, the database system can choose using some of the stored meta-granules as a means to accelerate the query. Sometimes, DBMS can find a perfect meta-granule. If such meta-granule does not exists, DBMS will not give up. According to the presented rewriting algorithm, DBMS will use the maximal suitable meta-granule, i.e. the one that is least coarse but still fits the query. Our solution has two benefits. First, the database administrator does not have to create all imaginable materializations. Second, even if some reasonable materialization has been forgotten, the database system can still use exiting imperfect meta-granules to boost the query.

We have also shown results of the experimental evaluation of out method. It proves that even if some ideal meta-granules lack, the database system can still offer satisfactory performance. The experiments also attest that the overhead caused by the need to keep materialized data in sync is acceptable.

References

1. Gawarkiewicz, M., Wiśniewski, P.: Partial aggregation using Hibernate. In: FGIT. Volume 7105 of LNCS. (2011) 90–99
2. O'Neil, E.J.: Object/relational mapping 2008: Hibernate and the Entity Data Model (EDM). In Wang, J.T.L., ed.: SIGMOD Conference, ACM (2008) 1351–1356
3. Flexviews: Incrementally refreshable materialized views for MySQL (2012)
4. Mumick, I.S., Quass, D., Mumick, B.S.: Maintenance of data cubes and summary tables in a warehouse. In Peckham, J., ed.: SIGMOD Conference, ACM Press (1997) 100–111

5. Salem, K., Beyer, K., Lindsay, B., Cochrane, R.: How to roll a join: asynchronous incremental view maintenance. SIGMOD Rec. **29** (2000) 129–140
6. Melnik, S., Adya, A., Bernstein, P.A.: Compiling mappings to bridge applications and databases. ACM Trans. Database Syst. **33** (2008)
7. Szumowska, A., Burzańska, M., Wiśniewski, P., Stencel, K.: Efficient implementation of recursive queries in major object relational mapping systems. In: FGIT. (2011) 78–89
8. Boniewicz, A., Gawarkiewicz, M., Wiśniewski, P.: Automatic selection of functional indexes for object relational mappings system. International Journal of Software Engineering and Its Applications **7** (2013)

Checking MTL Properties of Discrete Timed Automata via Bounded Model Checking*

Extended Abstract

Bożena Woźna-Szcześniak and Andrzej Zbrzezny

IMCS, Jan Długosz University.
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
{b.wozna,a.zbrzezny}@ajd.czest.pl

Abstract. We investigate a SAT-based bounded model checking (BMC) method for MTL (metric temporal logic) that is interpreted over linear discrete infinite time models generated by discrete timed automata. In particular, we translate the existential model checking problem for MTL to the existential model checking problem for a variant of linear temporal logic (called HLTL), and we provide a SAT-based BMC technique for HLTL. We show how to implement the BMC technique for HLTL and discrete timed automata, and as a case study we apply the technique in the analysis of TGPP, a Timed Generic Pipeline Paradigm modelled by a network of discrete timed automata.

1 Introduction

Nowadays the interest in model checking [5] is focused not only on standard concurrent systems, but also on soft real-time systems, i.e., systems the goal of which is to ensure a certain subset of deadlines in order to optimize some application specific criteria. A number of formalisms, which use a discrete time domain, have been proposed in the literature to model the behaviour of these systems, e.g. discrete timed automata [2] and discrete timed Petri nets [7]. To express the requirements of the systems mostly standard temporal logics are used: *computation tree logic* (CTL) [4], *the soft real-time CTL* (RTCTL) [6], *linear temporal logic* (LTL) [13], and *metric temporal logic* (MTL) [8, 10, 14].

Bounded model checking (BMC) [3, 11, 12] is a symbolic verification method that uses only a portion of the considered model that is truncated up to some specific depth. It exploits the observation that we can infer some properties of the model using only its fragments. This approach can be combined with symbolic techniques based on decision diagrams or with techniques which involve translation of the verification problem either to the boolean satisfiability problem (SAT) or to the satisfiability modulo theories (SMT) problem.

The original contributions of the paper are as follows. First, we define a SAT-based BMC for soft real-time systems, which are modelled by discrete

* Partly supported by National Science Center under the grant No. 2011/01/B/ST6/05317.

timed automata, and for properties expressible in MTL. Next, we report on the implementation of the proposed BMC method as a new module of VerICS [9]. Finally, we evaluate the BMC method experimentally by means of a *timed generic pipeline paradigm* (TGPP), which we model by a network of discrete timed automata.

The rest of the paper is structured as follows. In Section 2 we brief the basic notion used through the paper. In Section 3 we define the BMC method for HLTL. In Section 4 we discuss our experimental results. In Section 5 we conclude the paper.

2 Preliminaries

We assume familiarity with the notion of *discrete timed automaton* (DTA) and their semantics in terms of the Kripke structure (called *model*). We refer the reader to the body of the paper [15] for details; note that a discrete timed automaton is basically a timed automaton with the restriction that clocks are positive integer variables. Further, we assume the following syntax of MTL. Let $p \in \mathcal{PV}$, and I be an interval in $\mathbb{N} = \{0, 1, 2, \ldots\}$ of the form: $[a, b)$ or $[a, \infty)$, for $a, b \in \mathbb{N}$ and $a \neq b$; note that the remaining forms of intervals can be defined by means of $[a, b)$ and $[a, \infty)$. The MTL formulae in the negation normal form are defined by the following grammar:

$$\alpha := \top \mid \bot \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \mathrm{U}_I \alpha \mid \alpha \mathrm{R}_I \alpha$$

We refer the reader to the body of the paper [15] for the semantics of MTL; note that to get the discrete time semantics for MTL from the dense time semantics for MITL, in all the definitions presented in [15] at page 5, it is enough to replace the set of positive real numbers with the set of positive integer numbers, and to drop the assumption about single intervals.

Determining whether an MTL formula φ is existentially (resp. universally) valid in a given model is called an *existential* (resp. *universal*) *model checking problem*.

In order to define a SAT-based BMC method for MTL, we first translate the existential model checking problem for MTL that is interpreted over the region graph (i.e., a standard finite model defined for (discrete) timed automata) to the existential model checking problem for HLTL that is also interpreted over the region graph. For the details on this translation and the semantics of the HLTL language we refer the reader to [15]; note that the single intervals do not affect this translation. Here we only provide the syntax of HLTL and the translation scheme.

Let φ be an MTL formula, n the number of intervals in φ, $p \in \mathcal{PV}$ a propositional variables, and $h = 0, \ldots, n-1$. The HLTL formulae in release positive normal form are given by the following grammar:

$$\alpha := \top \mid \bot \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \mathrm{H}_h \alpha \mid \alpha \mathrm{U} \alpha \mid \alpha \mathrm{R} \alpha$$

where the symbols U and R denote the *until* and *release* modalities, respectively. The indexed symbol H_h denotes the *reset* modality representing the reset of the clock number h. In addition, we introduce some useful derived temporal modalities: $\mathrm{G}\alpha \stackrel{def}{=} \bot \mathrm{R} \alpha$ (*always*), $\mathrm{F}\alpha \stackrel{def}{=} \top \mathrm{U} \alpha$ (*eventually*).

Let φ be a MTL formula. We translate the formula φ inductively into the HLTL formula $\mathcal{H}(\varphi)$ in the following way:

$$\mathcal{H}(\top) = \top, \quad \mathcal{H}(\bot) = \bot, \quad \mathcal{H}(p) = p, \quad \mathcal{H}(\neg p) = \neg p, \text{ for } p \in \mathcal{PV},$$
$$\mathcal{H}(\alpha \vee \beta) = \mathcal{H}(\alpha) \vee \mathcal{H}(\beta), \quad \mathcal{H}(\alpha \wedge \beta) = \mathcal{H}(\alpha) \wedge \mathcal{H}(\beta),$$
$$\mathcal{H}(\alpha \mathrm{U}_{I_h} \beta) = \mathrm{H}_h(\mathcal{H}(\alpha) \mathrm{U} (\mathcal{H}(\beta) \wedge p_{y_h \in I_h} \wedge (p_{nf} \vee \mathcal{H}(\alpha)))),$$
$$\mathcal{H}(\alpha \mathrm{R}_{I_h} \beta)) = \mathrm{H}_h(\mathcal{H}(\alpha) \mathrm{R} (\neg p_{y_h \in I_h} \vee \mathcal{H}(\beta))).$$

Observe that the translation of literals as well as logical connectives is straightforward. The translation of the U_{I_h} operator ensures that: (1) the translation of β holds in the interval I – this is expressed by the requirement $\mathcal{H}(\beta) \wedge p_{y_h \in I_h}$; (2) the translation of α holds always before the translation of β; and (3) if the value of the clock y_h belong to the final zone, i.e. the values of all the clocks are bigger then some maximal value (in this case the proposition p_{nf} is not true), then the translation of $\mathcal{H}(\alpha)$ is taken into account as well. The translation of the R_{I_h} operator makes use of the fact $\alpha \mathrm{R}_{I_h} \beta = \beta \mathrm{U}_{I_h} \alpha \wedge \beta \vee \mathrm{G}_{I_h} \beta$.

This translation preserves the existential model checking problem, i.e., the existential model checking of an MTL formula φ over the discrete model can be reduced to the existential model checking of $\mathcal{H}(\varphi)$ over the region graph.

The next step in defining a SAT-based BMC method for MTL relies on introducing a discretisation scheme for the region graph (defined for a given discrete timed automaton) that will represent zones (i.e. sets of equivalent clock valuations) of the region graph by exactly one specially chosen representative, and proving that a discretised model based on this scheme preserves the validity of the HLTL formulae - the discretised model constitutes the base for an implementation of our BMC method. We do not report on this step here in detail, since it requires introducing the huge mathematical machinery, but in fact it can be done in a way similar to the one presented in [16]. However, this will be provided in the full version of the paper.

The final step in defining a SAT-based BMC method for MTL relies on defining the BMC method for HLTL. This is described in the next section.

3 Bounded model checking for HLTL

Bounded semantics of a logic in question with existential interpretation is always used as the theoretical basis for the SAT-based bounded model checking. In the paper we have decided not to provide this semantics and not to show its equivalence to the unbounded semantics. This will be presented in the full version of the paper. Here, we only focus on the core of the BMC method, i.e. on the translation to SAT.

Let $\mathcal{A}$ be a discrete timed automaton, φ an MTL formula, $\psi = \mathcal{H}(\varphi)$ the corresponding HLTL formula, $\mathcal{M}$ a discretized model for $\mathcal{A}_\varphi$ (this an extension

of $\mathcal{A}$ – for the exact construction we refer to [15]), and $k \geq 0$ a bound. We propose a BMC method for HLTL, which is based on the BMC technique presented in [17]. More precisely, we construct a propositional formula

$$[\mathcal{M}, \psi]_k := [\mathcal{M}^{\psi,\iota}]_k \ \wedge \ [\psi]_{\mathcal{M},k} \tag{1}$$

that is satisfiable if and only if the underlying model $\mathcal{M}$ is a genuine model for ψ. The constructed Formula (1) is given to a satisfiability solving program (a SAT-solver), and if a satisfying assignment is found, that assignment is a witness for the checked property. If a witness cannot be found at a given depth, k, then the search is continued for larger k.

The definition of the formula $[\mathcal{M}, \psi]_k$ requires, among other, states of the model $\mathcal{M}$ to be encoded in a symbolic way. This encoding is possible, since the set of states of $\mathcal{M}$ is finite. In particular, we represent each state s by a vector $w = (\mathtt{w}_1, \ldots, \mathtt{w}_r)$ (called a *symbolic state*) of propositional variables (called *state variables*), whose length r depends on the number of locations and clocks in $\mathcal{A}_\varphi$. Further, we need to represent finite prefixes of paths in a symbolic way. We call this representation a *j-th symbolic k-path* $\boldsymbol{\pi}_j$ and define it as a pair $((w_{0,j}, \ldots, w_{k,j}), u_j)$, where $w_{i,j}$ are symbolic states for $0 \leq j < f_k(\psi)$ and $0 \leq i \leq k$, and u_j is a *symbolic number* for $0 \leq j < f_k(\psi)$. The *symbolic number* u_j is a vector $u_j = (\mathtt{u}_{1,j}, \ldots, \mathtt{u}_{t,j})$ of propositional variables (called *natural variables*), whose length t equals to $max(1, \lceil log_2(k+1) \rceil)$, and it is used to encode the looping conditions. Next, we need an auxiliary function $\widehat{f_k}$: HLTL $\to \mathbb{N}$ that gives a bound on the number of k-paths sufficient for validating a given HLTL formula. The function is defined as $\widehat{f_k}(\psi) = f_k(\psi) + 1$, where $f_k(\top) = f_k(\bot) = f_k(p) = f_k(\neg p) = 0$ for $p \in \mathcal{PV}$; $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$; $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$; $f_k(\mathrm{H}_h\alpha) = f_k(\alpha)+1$; $f_k(\alpha \mathrm{U} \beta) = k \cdot f_k(\alpha) + f_k(\beta)$; $f_k(\alpha \mathrm{R} \beta) = (k+1) \cdot f_k(\beta) + f_k(\alpha)$.

The formula $[\mathcal{M}^{\psi,\iota}]_k$ – the 1st conjunct of Formula (1) – encodes $\widehat{f_k}(\psi)$-times unrolled transition relation, and it is defined in the following way:

$$[\mathcal{M}^{\psi,\iota}]_k := I_\iota(w_{0,0}) \wedge \bigvee_{j=1}^{\widehat{f_k}(\varphi)} H(w_{0,0}, w_{0,j}) \wedge \bigwedge_{j=1}^{\widehat{f_k}(\psi)} \bigwedge_{i=0}^{k-1} \mathcal{T}(w_{i,j}, w_{i+1,j}) \wedge \bigwedge_{j=0}^{\widehat{f_k}(\psi)} \bigvee_{l=0}^{k} B_l^{=}(u_j) \tag{2}$$

where $w_{i,j}$ are symbolic states, u_j is a symbolic number, $I_\iota(w_{0,0})$ and $\mathcal{B}_l^{=}(u_j)$ are formulae encoding the initial state, and the value l, respectively. $H(w, w')$ is a formula that encodes equality of two global states. The formula $\mathcal{T}(w_{i,j}, w_{i+1,j})$ is disjunction of three formulas: $T(w_{i,j}, w_{i+1,j})$, $TA(w_{i,j}, w_{i+1,j})$, and $A(w_{i,j}, w_{i+1,j})$ that encode respectively the time, time-action, and action successors of $\mathcal{M}$.

The formula $[\psi]_{\mathcal{M},k} := [\psi]_k^{[0,1,F_k(\psi)]}$ – the 2nd conjunct of Formula (1) – encodes the translation of a HLTL formula ψ along a k-path, whose number belongs to the set $F_k(\psi) = \{j \in \mathbb{N} \mid 1 \leq j \leq \widehat{f_k}(\psi)\}$. The main idea of this translation consists in translating every subformula α of ψ using only $f_k(\alpha)$ k-paths. More precisely, given a formula ψ and a set $F_k(\psi)$ of indices of k-paths, following [17], we divide the set $F_k(\psi)$ into subsets needed for translating the subformulae of

ψ. We assume that the reader is familiar with this division process, and here we only recall definitions of the functions we use in the definition of the formula $[\psi]_k^{[0,1,F_k(\psi)]}$.

First, we recall the relation $\prec$ that is defined on the power set of $\mathbb{N}$ as: $A \prec B$ iff for all natural numbers x and y, if $x \in A$ and $y \in B$, then $x < y$. Now, let $A \subset \mathbb{N}$ be a finite nonempty set, and $n, d \in \mathbb{N}$, where $d \leq |A|$. Then,

- $g_l(A, d)$ denotes the subset B of A such that $|B| = d$ and $B \prec A \setminus B$.
- $g_r(A, d)$ denotes the subset C of A such that $|C| = d$ and $A \setminus C \prec C$.
- $g_s(A)$ denotes the set $A \setminus \{min(A)\}$.
- if n divides $|A| - d$, then $hp(A, d, n)$ denotes the sequence $(B_0, \ldots, B_n)$ of subsets of A such that $\bigcup_{j=0}^{n} B_j = A$, $|B_0| = \ldots = |B_{n-1}|$, $|B_n| = d$, and $B_i \prec B_j$ for every $0 \leq i < j \leq n$.

Now let $h_k^{\mathrm{U}}(A, d) \stackrel{df}{=} hp(A, d, k)$ and $h_k^{\mathrm{R}}(A, d) \stackrel{df}{=} hp(A, d, k+1)$. Note that if $h_k^{\mathrm{U}}(A, d) = (B_0, \ldots, B_k)$, then $h_k^{\mathrm{U}}(A, d)(j)$ denotes the set B_j, for every $0 \leq j \leq k$. Similarly, if $h_k^{\mathrm{R}}(A, d) = (B_0, \ldots, B_{k+1})$, then $h_k^{\mathrm{R}}(A, d)(j)$ denotes the set B_j, for every $0 \leq j \leq k+1$. Further, the function g_s is used in the translation of subformulae of the form $\mathrm{H}_h\alpha$, if a set A is used to translate this formula, then the path of the number $min(A)$ is used to translate the operator H_h and the set $g_s(A)$ is used to translate the subformula α. For more details on the remaining functions we refer to [17].

Now we are ready to define the formula $[\psi]_k^{[0,1,F_k(\psi)]}$. Let ψ be a HLTL formula, and $k \geq 0$ a bound. We can define inductively the translation $[\psi]_k^{[m,n,A]}$ of ψ along the n-th symbolic $k-$path $\boldsymbol{\pi}_n$ ($n \in F_k(\psi)$) with starting point m by using the set A as shown below. Let $cl(w_{m,n}, h)$ denote the fragment of the symbolic state $w_{m,n}$ that encodes the h-th clock from the set $\mathbb{Y}$, $n' = min(A)$, $h_k^{\mathrm{U}} = h_k^{\mathrm{U}}(g_s(A), f_k(\beta))$, and $h_k^{\mathrm{R}} = h_k^{\mathrm{R}}(g_s(A),\ f_k(\alpha))$. Then,

$$
\begin{aligned}
&[\top]_k^{[m,n,A]} := \top,\ [\bot]_k^{[m,n,A]} := \bot,\ [p]_k^{[m,n,A]} := p(w_{m,n}),\ [\neg p]_k^{[m,n,A]} := \neg p(w_{m,n}),\\
&[\alpha \wedge \beta]_k^{[m,n,A]} := [\alpha]_k^{[m,n,g_l(A,f_k(\alpha))]} \wedge [\beta]_k^{[m,n,g_r(A,f_k(\beta))]},\\
&[\alpha \vee \beta]_k^{[m,n,A]} := [\alpha]_k^{[m,n,g_l(A,f_k(\alpha))]} \vee [\beta]_k^{[m,n,g_l(A,f_k(\beta))]},\\
&[\mathrm{H}_h(\alpha \mathrm{U} \beta)]_k^{[m,n,A]} := \textstyle\bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}\\
&\quad (w_{j,n}, w_{j,n'}) \wedge \big(\textstyle\bigvee_{j=m}^{k}([\beta]_k^{[j,n',h_k^{\mathrm{U}}(k)]} \wedge \bigwedge_{i=m}^{j-1}[\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]})\big) \vee\\
&\quad \textstyle\bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge\\
&\quad \big(\textstyle\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge\\
&\quad \textstyle\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}))\big) \wedge \big(\textstyle\bigvee_{j=0}^{m-1}(\mathcal{B}_j^{>}(u_{n'}) \wedge [\beta]_k^{[j,n',h_k^{\mathrm{U}}(k)]}\\
&\quad \wedge \textstyle\bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_{n'}) \rightarrow [\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]}))\big) \wedge \textstyle\bigwedge_{i=m}^{k}[\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]},\\
&[\mathrm{H}_h(\alpha \mathrm{R} \beta)]_k^{[m,n,A]} := \textstyle\bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}\\
&\quad (w_{j,n}, w_{j,n'}) \wedge \big(\textstyle\bigvee_{j=m}^{k}([\alpha]_k^{[j,n',h_k^{\mathrm{R}}(k+1)]} \wedge \bigwedge_{i=m}^{j}[\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]})\big)\\
&\quad \vee \textstyle\bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge\\
&\quad \big(\textstyle\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge\\
&\quad \textstyle\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}))\big) \wedge \big(\textstyle\bigvee_{j=0}^{m}(\mathcal{B}_j^{>}(u'_n) \wedge [\alpha]_k^{[j,n',h_k^{\mathrm{R}}(k+1)]}
\end{aligned}
$$

$$\wedge \bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_{n'}) \rightarrow [\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]}))) \wedge \bigwedge_{i=m}^{k}[\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]}$$
$$\vee \bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k}$$
$$H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]} \wedge \mathcal{B}^{\leq}_{right(h)}(cl(w_{k,n'}, h))$$
$$\vee \mathcal{B}^{>}_{right(h)}(cl(w_{k,n'}, h)) \wedge \bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}$$
$$(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]}$$
$$\wedge (\bigvee_{l=m}^{k-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'})) \vee \mathcal{B}^{>}_{right(h)}(cl(w_{k,n'}, h)) \wedge H_{h=0}(w_{m,n}, w_{m,n'})$$
$$\wedge \bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]} \wedge$$
$$\big(\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge$$
$$\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=l+1}^{m-1}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]})\big).$$

where $p(w)$ is a formula that encodes a set of states of M in which $p \in \mathcal{PV}$ holds; $H(w, w')$ is a formula that encodes equality of two global states; $H_{h=0}(w, w')$ is a formula that for two global states encodes the equality of their locations, the equality of values of the original clocks (i.e., clocks from $\mathbb{X}$), and the equality of values of the new clocks (i.e., clocks from $\mathbb{Y}$) but the value of clock y_h. For clock y_h the formula guarantees that its value in the 2nd global state is equal to zero; $H_{\neq h}(w, w')$ is a formula that for two global states encodes the equality of their locations, the equality of values of the original clocks, and the equality of the values of the new clocks with the potential exception of clock y_h. For clock y_h the formula guarantees that its value in the 2nd global state is greater than zero; $H_{\mathbb{X}}(w, w')$ is a formula that encodes equality of two global states on locations and values of the original clocks; $\mathcal{B}_j^{\sim}(\mathbf{v})$ is a formula that encodes that the value represented by the vector of propositional variables $\mathbf{v}$ is in arithmetic relation $\sim$ with the value j, where $\sim \in \{<, \leqslant, =, \geqslant, >\}$; $\mathcal{L}_k^l(\boldsymbol{\pi}_j) := \mathcal{B}_k^{>}(u_j) \wedge H_{\mathbb{X}}(w_{k,j}, w_{l,j})$.

The following theorem, whose proof will be provided in the full version of the paper, guarantees that the bounded model checking problem can be reduced to the SAT-problem.

Theorem 1. *Let $\mathcal{M}$ be a discrete abstract model, and ψ a HLTL formula. Then for every $k \in \mathbb{N}$, ψ is existentially valid in $\mathcal{M}$ with the bound k if, and only if, the propositional formula $[\mathcal{M}, \psi]_k$ is satisfiable.*

4 Experimental results

Our SAT-based BMC method for MTL, interpreted over the discrete time models, and discrete timed automata is, to our best knowledge, the first one formally presented in the literature, and moreover there is no any other model checking technique for the considered MTL language. Further, our implementation of the presented BMC method uses Reduced Boolean Circuits (RBC) [1] to represent the propositional formula $[\mathcal{M}, \psi]_k$. An RBC represents subformulae of $[\mathcal{M}, \psi]_k$ by fresh propositions such that each two identical subformulae correspond to the same proposition.

For the tests we have used a computer with Intel Core i3-2125 processor, 8 GB of RAM, and running Linux 2.6. We set the time limit to 900 seconds, and memory limit to 8GB, and we used the state of the art SAT-solver MiniSat 2. The specifications for the described benchmark are given in the universal form, for which we verify the corresponding counterexample formula, i.e., the formula which is negated and interpreted existentially.

To evaluate the performance of our SAT-based BMC algorithms for the verification of several properties expressed in MTL, we have analysed a Timed Generic Pipeline Paradigm (TGPP) discrete timed automata model shown in Figure 1. It consists of Producer producing data ($ProdReady$) or being inactive, Consumer receiving data ($ConsReady$) or being inactive, and a chain of n intermediate Nodes which can be ready for receiving data ($Node_iReady$), processing data ($Node_iProc$), or sending data ($Node_iSend$). The example can be scaled by adding intermediate nodes or by changing the length of intervals (i.e., the parameters a, b, c, d, e, f, g, h) that are used to adjust the time properties of Producer, Consumer, and of the intermediate Nodes.

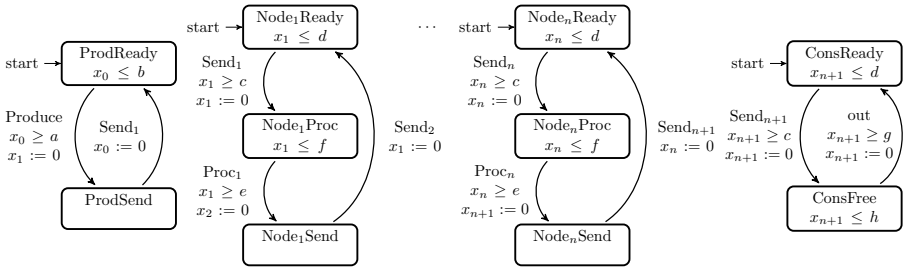


Fig. 1. A Generic Timed Pipeline Paradigm discrete timed automata model

We have tested the TGPP discrete timed automata model, scaled in the number of intermediate nodes and with all the intervals set to $[1, 3]$, on the following MTL formulae:

- $\varphi_1 = \mathrm{G}_{[0,\infty)}(ProdSend \Rightarrow \mathrm{F}_{[2n+1,2n+2)}ConsFree)$, where n is the number of nodes. It expresses that each time Producer produces data, then Consumer receives this data in $2n + 1$ time units.
- $\varphi_2 = \mathrm{G}_{[0,\infty)}(ProdSend \Rightarrow \mathrm{F}_{[2n+1,2n+2)}(ConsFree \wedge \mathrm{F}_{[1,2)}ConsReady))$, where n is the number of nodes. It expresses that each time Producer produces data, then Consumer receives this data in $2n + 1$ time units and one unit after that it will be ready to receive another data.

Since there is no model checker that supports the MTL properties of systems modelled by discrete timed automata, we were not able to compare results of the application of our method to a TGPP system with others.

We provide a preliminary evaluation of our method by means of the running time and the consumed memory. We have observed that for both formulae φ_1 and

φ_2, we managed to compute the results for 5 nodes in the time of 900 seconds. The exact data for the mentioned maximal number of nodes are the following:

φ_1: $k = 20$, $f_k(\varphi_1) = 3$, bmcT is 6.50, bmcM is 19.54, satT is 25.24, satM is 43.00, bmcT+satT is 31.74, max(bmcM,satM) is 43.00;

φ_2: $k = 20$, $f_k(\varphi_2) = 24$, bmcT is 89.96, bmcM is 163.40, satT is 610.23, satM is 310.00, bmcT+satT is 700.19, max(bmcM,satM) is 310.00;

where k is the bound, $f_k(\varphi)$ is the number of symbolic k-paths, $bmcT$ is the encoding time, $bmcM$ is memory use for encoding, $satT$ is the satisfiability checking time, $satM$ is memory use for the satisfiability checking.

The preliminary results are very promising and indicate that the method is worthy of further investigation for which purpose especially designed benchmarks will be developed.

5 Conclusions

We have introduced a SAT-based approach to bounded model checking of discrete timed automata and properties expressed in MTL with discrete semantics. The method is based on a translation of the existential model checking for MTL to the existential model checking for HLTL, and then on the translation of the existential model checking for HLTL to the propositional satisfiability problem. The two translations have been implemented and tested on the benchmark, which has been carefully selected in such a way as to reveal the advantages and disadvantages of the presented approaches.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000.
2. R. Alur and D. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*, pages 118–149. Academic Press, 2003.
4. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using Branching Time Temporal Logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
6. E. A. Emerson, A.K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, December 1992.
7. M. Felder, D. Mandrioli, and A. Morzenti. Proving Properties of Real-Time Systems Through Logical Specifications and Petri Net Models. *IEEE Transaction on Software Engineering*, 20(2):127–141, 1994.

8. C. A. Furia and P. Spoletini. Tomorrow and all our yesterdays: MTL satisfiability over the integers. In *Proceedings of the Theoretical Aspects of Computing - ICTAC 2008*, volume 5160 of *LNCS*, pages 253–264. Springer-Verlag, 2008.
9. M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M.Szreter, B. Woźna, and A. Zbrzezny. VerICS 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.
10. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
11. A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking for knowledge and real time. *Artificial Intelligence*, 171:1011–1038, 2007.
12. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
13. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE International Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Presss, 1977.
14. M. Pradella, A. Morzenti, and P. San Pietro. A metric encoding for bounded model checking. In *Proceedings of the 2nd World Congress on Formal Methods (FM 2009)*, volume 5850 of *LNCS*, pages 741–756. Springer-Verlag, 2009.
15. B. Woźna-Szcześniak and A. Zbrzezny. A translation of the existential model checking problem from MITL to HLTL. *Fundamenta Informaticae*, 122(4):401–420, 2013.
16. A. Zbrzezny. A new discretization for timed automata. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 178–189. Humboldt University, 2004.
17. A. Zbrzezny. A new translation from ECTL* to SAT. *Fundamenta Informaticae*, 120(3–4):377–397, 2012.

On Boolean Encodings of Transition Relation for Parallel Compositions of Transition Systems*

Extended abstract

Andrzej Zbrzezny

IMCS, Jan Długosz University in Częstochowa,
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
a.zbrzezny@ajd.czest.pl

Abstract. We present and compare different Boolean encodings of the transition relation for the parallel composition of transition systems, both for the asynchronous and the synchronous semantics. We compare the encodings considered by applying them to the SAT-based bounded model checking (BMC) of ECTL* properties. We provide experimental results which show that our new encoding for the asynchronous semantics significantly increases the efficiency of the SAT-based BMC.

Keywords: transition system, parallel composition, SAT-based bounded model checking, Boolean encoding.

1 Introduction

One of the most important practical problems in model checking is the exponential growth of number of states of the transition system, depending on the number of components of the modelled system. Edmund M. Clarke, co-inventor of the model checking, stressed that the problem of overcoming the exponential explosion in the number of states has been one of the most important research questions he was dealing with since the inventing of model checking [1–3]. Reducing the negative impact of the exponential explosion of the state space requires in particular the methods and algorithms to have the highest possible efficiency.

The aim of this paper is to present and compare different Boolean encodings of the transition relation for the parallel composition of transition systems, both for the asynchronous and the synchronous semantics. We provide experimental results which show that our new encoding for the asynchronous semantics significantly increases the efficiency of the SAT-based bounded model checking.

2 Preliminaries

2.1 Transition systems

Transition systems ([4]) are often used as models to describe the behaviour of systems. They are basically directed graphs where nodes represent states, and edges model transi-

* Partly supported by National Science Center under the grant No. 2011/01/B/ST6/05317.

tions, i.e., state changes. A state describes some information about a system at a certain moment of its behaviour.

Definition 1. *A* transition system *is a tuple $\mathcal{M} = (S, Act, \longrightarrow, I, AP, L)$, where S is a nonempty finite set of states, Act is a set of actions, $I \subseteq S$ is the set of initial states, $\longrightarrow \subseteq S \times Act \times S$ is a transition relation, AP is a set of atomic propositions, and $L : S \to 2^{AP}$ is a* labelling function *that assigns to each state a set of atomic propositions that are assumed to be true at that state. Transition systems are also called* models.

For convenience, we write $s \xrightarrow{\sigma} s'$ instead of $(s, \sigma, s') \in \longrightarrow$. Moreover, we write $s \longrightarrow s'$ if $s \xrightarrow{\sigma} s'$ for some $\sigma \in Act$.

From now on we assume that transition systems are finite, i.e. the sets S, Act and AP are finite We also assume that a transition system has no terminal states, i.e. for every $s \in S$ there exist $s' \in S$ such that $s \longrightarrow s'$. The set of all natural numbers is denoted by $\mathbb{N}$ and the set of all positive natural numbers by $\mathbb{N}_+$. A *path* in $\mathcal{M}$ is an infinite sequence $\rho = (s_0, s_1, \ldots)$ of states such that $s_j \longrightarrow s_{j+1}$ for each $j \in \mathbb{N}$.

2.2 Parallel compositions of transition systems

Definition 2. *Let J be a non-empty, finite set of indices and let $\{\mathcal{M}_j \mid j \in J\}$ be a family of transition systems, i.e. $\mathcal{M}_j = (S_j, Act_j, \longrightarrow_j, I_j, AP_j, L_j)$. Assume that $AP_i \cap AP_j = \emptyset$, for $i \neq j$. Moreover, let $J(\sigma) = \{j \in J \mid \sigma \in Act_j\}$, and $\varepsilon \notin Act_j$ for each $j \in J$.*

1. *The* asynchronous parallel composition *of the family $\{\mathcal{M}_j \mid j \in J\}$ of transition systems is the transition system $\mathcal{M} = (S, Act, \longmapsto, I, AP, L)$ such that $S = \prod_{j \in J} S_j$, $Act = \bigcup_{j \in J} Act_j$, $I = \prod_{j \in J} I_j$, $AP = \bigcup_{j \in J} AP_j$, $L = \bigcup_{j \in J} L_j$, and $\longmapsto$ is defined as follows: for every $s, s' \in S$ and every $\sigma \in Act$: $s \xmapsto{\sigma} s'$ if and only if the following conditions hold:*
   (a) $s_j \xrightarrow{\sigma}_j s'_j$ for $j \in J(\sigma)$,
   (b) $s'_j = s_j$ for $j \in J \setminus J(\sigma)$.
2. *The* synchronous parallel composition *of the family $\{\mathcal{M}_j \mid j \in J\}$ of transition systems is the transition system $\mathcal{M} = (S, \overline{Act}, \longrightarrow, I, AP, L)$ such that $S = \prod_{j \in J} S_j$, $\overline{Act} = \prod_{j \in J} (Act_j \cup \{\varepsilon\})$, $I = \prod_{j \in J} I_j$, $AP = \bigcup_{j \in J} AP_j$, $L = \bigcup_{j \in J} L_j$, and $\longrightarrow$ is defined as follows: for every $s, s' \in S$ and every $\overline{\sigma} \in \overline{Act}$: $s \xrightarrow{\overline{\sigma}} s'$ if and only if the following conditions hold:*
   (a) $(\forall j \in J)\ (\sigma_j = \varepsilon \implies s'_j = s_j)$,
   (b) $(\exists j \in J)((\sigma_j \neq \varepsilon)$,
   (c) $(\forall j \in J)\ (\sigma_j \neq \varepsilon \implies s_j \xrightarrow{\sigma_j}_j s'_j)$,
   (d) $(\forall i \in J)(\sigma_i \neq \varepsilon \implies (\forall j \in J(\sigma_i))\ \sigma_j = \sigma_i)$.

Recall that if $|J(\sigma)| > 1$ for an action $\sigma \in \bigcup_{j \in J} Act_j$, then the action σ is called a *shared* action; otherwise, i.e. if $|J(\sigma)| = 1$, it is called a *local* action. The actions from $\overline{Act}$ are called *joint* actions.

In the transition system $\mathcal{M}$ being the asynchronous parallel composition of a family of transition systems only one local or shared action may be performed at a given time in a given global state of $\mathcal{M}$. Moreover, a shared action σ has to be performed in all the components $\mathcal{M}_j$, for $j \in J(\sigma)$.

In the transition system $\mathcal{M}$ being the synchronous parallel composition of a family of transition systems a joint action $\overline{\sigma} \in \overline{Act}$ is to be performed by the system at a given time in a given global state of $\mathcal{M}$. It means, that every component $\mathcal{M}_j$ of the system can perform an action (also an empty action ε). Notice however, that at least one component has to perform a non-empty action. Moreover, if one of the components performs a shared action σ, then the action σ has to be performed in all the components $\mathcal{M}_j$, for $j \in J(\sigma)$.

Observe that if $s \longmapsto s'$ then there exists $\overline{\sigma} \in \overline{Act}$ such that $(s, \xrightarrow{\overline{\sigma}}, s')$. Thus every path of the asynchronous parallel composition of transition systems is also a path of the synchronous parallel composition of transition systems. It follows that every ECTL^* formula valid in the asynchronous parallel composition of a given family of transition systems is also valid in the synchronous parallel composition of this family. Note that the converse of the implication does not hold. However, let us note that each formula of the form $\mathbf{E}(\mathbf{F}\psi_1 \wedge \ldots \wedge \mathbf{F}\psi_m)$, where $\psi_1, \ldots, \psi_m$ are propositional formulae, valid in the synchronous parallel composition of the given family of transition systems is also valid in the asynchronous parallel composition of this family.

3 Boolean Encodings of States, Actions and Transition Relations

We start with describing the Boolean encoding of states, actions and the transition relation of a given model by means of propositional formulae, which are built over a set PV of propositional variables plus the constants $\mathbf{true}$ and $\mathbf{false}$, with help of the propositional connectives $\neg$, $\wedge$, $\vee$ and $\rightarrow$.

Let $\{\mathcal{M}_j \mid j \in J\}$ be a finite family of transition systems and let $\mathcal{M}$ be the parallel composition (either asynchronous or synchronous) of $\{\mathcal{M}_j \mid j \in J\}$. Since the set S_j of states of each $\mathcal{M}_j$ is finite, every element of S_j can be encoded as a bit vector of the length $\lceil \log_2 |S_j| \rceil$. Therefore, each state of $\mathcal{M}_j$ can be represented by a valuation of a vector $\mathbf{w}_j$ (called a *symbolic local state*) of unique *propositional state variables*. Then, each state of $\mathcal{M}$ can be represented by a valuation of a vector $\mathbf{w}$ (called a *symbolic global state*) of the vectors $\mathbf{w}_j$.

Similarly, since the set Act_j of actions of each $\mathcal{M}_j$ is finite, the set $\overline{Act}$ is also finite, and it follows that every element of Act_j can be encoded as a bit vector of the length $\lceil \log_2 |Act| \rceil$. Therefore, each action of $\mathcal{M}_j$ can be represented by a valuation of a vector $\mathbf{a}_j$ (called a *symbolic action*) of unique *propositional action variables*. Then, each element of the set $\overline{Act}$ can be represented by a valuation of a vector $\mathbf{a}$ (called a *symbolic joint action*) of the vectors $\mathbf{a}_j$.

Let $SV = \{w_1, w_2, \ldots\}$ be a set of *propositional state variables* and $AV = \{v_1, v_2, \ldots\}$ be a set of *propositional action variables*. We assume that $SV \cap AV = \emptyset$.

Moreover, let $PV = SV \cup AV$ and $V : PV \to \{0,1\}$ be a *valuation of propositional variables* (a *valuation* for short). For every $m, r \in \mathbb{N}_+$ each valuation V induces the functions $\mathbf{S} : SV^m \to \{0,1\}^m$ and $\mathbf{A} : AV^r \to \{0,1\}^r$ defined as follows:

$$\mathbf{S}(w_{j_1}, \ldots, w_{j_m}) = (V(w_{j_1}), \ldots, V(w_{j_m}))$$
$$\mathbf{A}(v_{j_1}, \ldots, v_{j_r}) = (V(v_{j_r}), \ldots, V(v_{j_r}))$$

Our aim is to define either a Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{w}')$ so that for each valuation $V \in \{0,1\}^{SV}$, V satisfies $\mathcal{T}(\mathbf{w}, \mathbf{w}')$ iff $\mathbf{S}(\mathbf{w}) \longmapsto \mathbf{S}(\mathbf{w}')$ in $\mathcal{M}$ or a Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{a}, \mathbf{w}')$ so that for each valuation $V \in \{0,1\}^{PV}$, V satisfies $\mathcal{T}(\mathbf{w}, \mathbf{a}, \mathbf{w}')$ iff $\mathbf{S}(\mathbf{w}) \stackrel{\mathbf{S}(\mathbf{a})}{\longrightarrow} \mathbf{S}(\mathbf{w}')$ in $\mathcal{M}$.

3.1 The standard Boolean encoding of asynchronous parallel composition

This encoding is implemented in all the SAT-based verification modules of VerICS [5–10]. In this encoding we use the following auxiliary propositional formulae:

- $\mathcal{H}_j(\mathbf{w}_j, \mathbf{w}'_j)$ for $j \in J$ is a Boolean formula defined so that for each valuation $V \in \{0,1\}^{SV}$, V satisfies $\mathcal{T}(\mathbf{w}_j, \mathbf{w}'_j)$ iff $\mathbf{S}(\mathbf{w}'_j) = \mathbf{S}(\mathbf{w}_j)$;
- $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{w}'_j)$ for $j \in J$ and $\sigma \in Act_j$ is a Boolean formula defined so that for each $V \in \{0,1\}^{SV}$, V satisfies $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{w}'_j)$ iff $\mathbf{S}(\mathbf{w}_j) \stackrel{\sigma}{\longrightarrow} \mathbf{S}(\mathbf{w}'_j)$ in $\mathcal{M}_j$.

Now it is possible to define the Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{w}')$

$$\mathcal{T}(\mathbf{w}, \mathbf{w}') = \bigvee_{\sigma \in Act} \left(\bigwedge_{j \in J(\sigma)} \mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{w}'_j) \ \wedge \bigwedge_{j \notin J(\sigma)} \mathcal{H}_j(\mathbf{w}_j, \mathbf{w}'_j) \right)$$

which symbolically encodes the transition relation $\longmapsto$ of the asynchronous parallel composition of the family $\{\mathcal{M}_j \mid j \in J\}$.

3.2 The Boolean encoding of synchronous parallel composition

We have recently implemented this encoding in the module BMC4ELTLK of VerICS. This module was used for performing experimental results presented in the article [11], which was recently accepted for publication. However, the encoding in question was not described in [11]. In this encoding we use the following auxiliary propositional formulae:

- $\mathcal{H}_\sigma(\mathbf{a}_j)$ for $j \in J$ and $\sigma \in Act_j \cup \{\varepsilon\}$ is a Boolean formula defined so that for each valuation $V \in \{0,1\}^{AV}$, V satisfies $\mathcal{H}_\sigma(\mathbf{a}_j)$ iff $\mathbf{A}(\mathbf{a}_j) = \sigma$;
- $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j)$ for $j \in J$ and $\sigma \in Act_j$ is a Boolean formula defined so that for each valuation $V \in \{0,1\}^{PV}$, V satisfies $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j)$ iff $\mathbf{A}(\mathbf{a}_j) = \sigma$ and $\mathbf{S}(\mathbf{w}_j) \stackrel{\sigma}{\longrightarrow} \mathbf{S}(\mathbf{w}'_j)$ in $\mathcal{M}_j$.

Now we can define the Boolean formula $\mathcal{T}_j(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j)$

$$\mathcal{T}_j(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j) = \big(\mathcal{H}(\mathbf{w}_j, \mathbf{w}'_j) \wedge \mathcal{H}_\varepsilon(\mathbf{a}_j)\big) \ \vee \bigvee_{\sigma \in Act_j} \mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j)$$

which symbolically encodes the transition relation $\longrightarrow_j$ of $\mathcal{M}_j$. Eventually, we define the Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{a}, \mathbf{w}')$ which symbolically encodes the transition relation $\longrightarrow$ of the synchronous parallel composition of the family $\{\mathcal{M}_j \mid j \in J\}$.

$$\mathcal{T}(\mathbf{w}, \mathbf{a}, \mathbf{w}') = \bigwedge_{j \in J} T_j(\mathbf{w}_j, \mathbf{a}_j, \mathbf{w}'_j) \wedge \bigwedge_{\sigma \in Act} \left(\bigwedge_{j \in \mathbf{J}(\sigma)} \mathcal{H}_\sigma(\mathbf{a}_j) \ \vee \bigwedge_{j \in \mathbf{J}(\sigma)} \mathcal{H}_\varepsilon(\mathbf{a}_j) \right)$$

The first subformula of the above conjunction assures that executing a joint action $\overline{\sigma}$ consists of executing all the actions that are the components of $\overline{\sigma}$. The second subformula requires that each action σ being a component of a joint action has to be executed in all the components of the synchronous parallel composition in which it appears.

3.3 A new Boolean encoding of asynchronous parallel composition

This encoding is inspired by the encoding for the synchronous parallel composition. However, it is not directly derived from the encoding for the synchronous parallel composition. In the new encoding for asynchronous parallel composition $\mathbf{b}$ stands for a vector (of the length $\lceil \log_2 |Act| \rceil$) of propositional action variables so that every action in Act can be be represented by a valuation of propositional variables in $\mathbf{b}$. In the encoding we use the following auxiliary propositional formula:

- $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j)$ for $j \in J$ and $\sigma \in Act_j$ is a Boolean formula defined so that for each valuation $V \in \{0,1\}^{PV}$, V satisfies $\mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j)$ iff $\mathbf{A}(\mathbf{b}) = \sigma$ and $\mathbf{S}(\mathbf{w}_j) \xrightarrow{\sigma} \mathbf{S}(\mathbf{w}'_j)$ in $\mathcal{M}_j$.

Now we can define the Boolean formula $\mathcal{T}_j(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j)$ which symbolically encodes the transition relation $\longrightarrow_j$ of $\mathcal{M}_j$:

$$\mathcal{T}_j(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j) = \left(\mathcal{H}(\mathbf{w}_j, \mathbf{w}'_j) \wedge \bigwedge_{\sigma \in Act_j} \neg \mathcal{H}_\sigma(\mathbf{b}) \right) \ \vee \bigvee_{\sigma \in Act_j} \mathcal{T}_{j,\sigma}(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j)$$

Intuitively, $\mathcal{T}_j(\mathbf{w}_j, \mathbf{b}, \mathbf{w}'_j)$ assures that either $\mathbf{A}(\mathbf{b}) \notin Act_j$ and $\mathbf{S}(\mathbf{w}_j) = \mathbf{S}(\mathbf{w}'_j)$ (i.e. no action is performed in $\mathcal{M}_j$ in a given step) or $\mathbf{S}(\mathbf{w}_j) \xrightarrow{\mathbf{A}(\mathbf{b})} \mathbf{S}(\mathbf{w}'_j)$ in $\mathcal{M}_j$.

Eventually, we define the Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{b}, \mathbf{w}')$ which symbolically encodes the transition relation $\longmapsto$ of the synchronous parallel composition of the family $\{\mathcal{M}_j \mid j \in J\}$:

$$\mathcal{T}(\mathbf{w}, \mathbf{b}, \mathbf{w}') = \bigwedge_{j \in J} \mathcal{T}_j(\mathbf{w}, \mathbf{b}, \mathbf{w}') \ \wedge \bigvee_{\sigma \in Act} \mathcal{H}_\sigma(\mathbf{b})$$

The first subformula of the above conjunction assures that for each action from Act the following condition is satisfied: if this action is to be performed in one of the components of the asynchronous parallel composition, then it has to be performed in all the components in which it appears. The second subformula requires that the vector $\mathbf{b}$ represents an action $\sigma \in Act$.

4 Experimental Results

Our experiments were performed on a computer equipped with Intel Xeon 2 GHz processor, 4GB of RAM and the operating system Ubuntu Linux Server with the kernel 3.5.0. We set the default limits of 1 GB of memory and 1800 seconds. Moreover, we used PicoSAT [12] in version 957 to test the satisfiability of the propositional formulae generated by the module BMC4ECTLS [9].

As the first benchmark we used the generic pipeline paradigm (GPP) introduced in [13]. The GPP consists of the Producer that is able to produce data, the Consumer that is able to receive data, and a chain of n intermediate Nodes that are able to receive, process, and send data. We assume that the following local states ProdReady, Ready_j and ConsReady are initial, respectively, for Producer, Node_j and Consumer. The global system is obtained as the parallel composition of the components, which are shown in Figure 4.

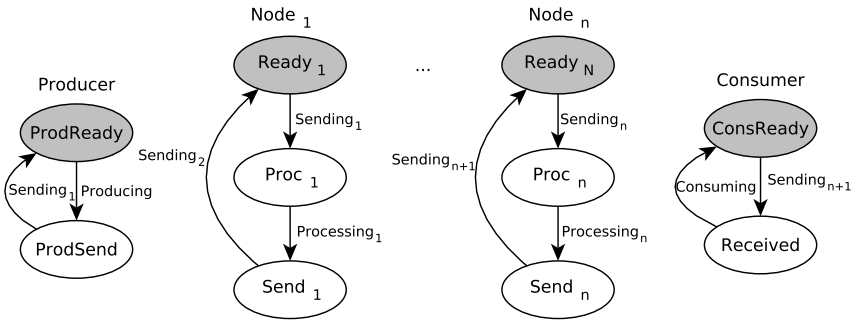


Fig. 1. The automata for the Sender, the Nodes and the Producer.

As the second benchmark we used the train controller system (TC) introduced in [14]. The TC consists of a controller, and n trains (for $n \geqslant 2$). Each train uses its own circular track for travelling in one direction. Eventually, each train has to pass through a tunnel, but because there is only one track in the tunnel, the trains arriving from each direction cannot use it simultaneously. There are signals on both sides of the tunnel, which can be either red or green. Each train notifies the controller when it request entry to the tunnel or when it leave the tunnel. The controller controls the colour of the displayed signal. The local state Away_j is initial for Train_j, and the local state Green is initial for Controller. The global system is obtained as the parallel composition of the components, which are shown in Figure 4.

As the third benchmark we used the dining philosophers problem [15, 16]. We have modelled this problem by means of a parallel composition of $2n + 1$ transition systems. The global system consists of n transition system each of which models a philosopher, together with n transition systems each of which models a fork, together with one transition system which models the lackey. The latter transition system is used to

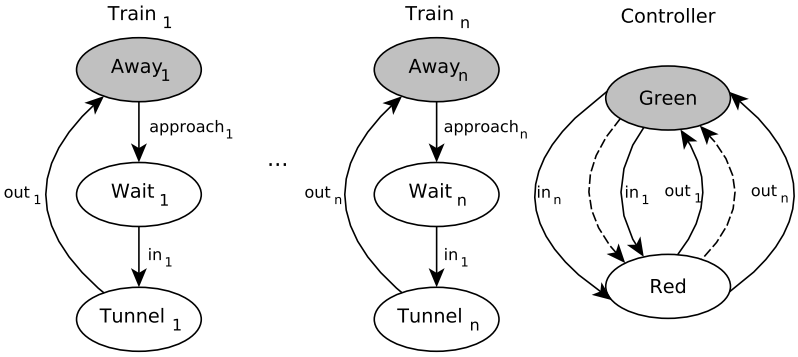


Fig. 2. The automata for the Trains and the Controller.

coordinate the philosophers' access to the dining-room. In fact, this automaton ensures that no deadlock is possible. The global system is obtained as the parallel composition of the components, which are shown in Figure 3.

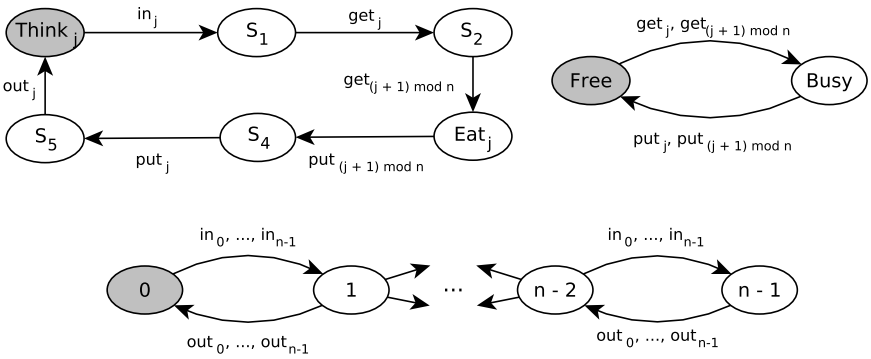


Fig. 3. The automata for the j-th Philosopher, the j-th Fork and the Lackey.

Let $\mathcal{I}(\mathbf{w})$ be a propositional formula such that for each valuation $V \in \{0,1\}^{SV}$, V satisfies $\mathcal{I}(\mathbf{w})$ iff $\mathbf{S}(w)$ is an initial state of the model. In order to compare experimental results for the three Boolean encodings of transition relation we have tested first (on a symbolic path of the length 1) the ECTL* formula $\varphi_0 = \mathbf{E}(\mathbf{true})$ that is valid in the models considered. The translation of the formula φ_0 results in the propositional formula:

– $\mathcal{I}(\mathbf{w}) \wedge \mathcal{T}(\mathbf{w}, \mathbf{w}')$ for the old encoding of the asynchronous parallel composition,

- $\mathcal{I}(\mathbf{w}) \wedge \mathcal{T}(\mathbf{w}, \mathbf{a}, \mathbf{w}')$ for the encoding of the synchronous parallel composition,
- $\mathcal{I}(\mathbf{w}) \wedge \mathcal{T}(\mathbf{w}, \mathbf{b}, \mathbf{w}')$ for the new encoding of the asynchronous parallel composition.

As we expected, the experimental results for the formula φ_0 (Table 1) show that the new encoding of the transition relation for the asynchronous parallel composition significantly influence the size (i.e. the number of clauses) of the resulting propositional formulae: for the GPP and 1800 nodes the number of clauses for the new encoding is 56 times less than for the old one; for the TC and 2800 trains the number of clauses for the new encoding is 25 times less than for the old one; for the DP and 1250 philosophers the number of clauses for the new encoding is 26 times less than for the old one.

Moreover, the results from Table 1 show that the encoding of the transition relation for the synchronous parallel composition results in propositional formulae that are shorter than formulae for the old encoding and significantly longer than formulae for the new encoding.

Encoding of the transition relation	(*Max) number of components	Number of variables	Number of clauses
Generic Pipeline Paradigm			
New	*105000	3727243	10131599
Sync	*28000	3689932	8130256
New	28000	989629	2688769
Old	*1800	3283218	9833442
Sync	1800	190792	426140
New	1800	63577	172637
Train Controller			
New	*87000	3829207	10617501
Sync	*29000	4262182	9800524
New	29000	1320132	3670282
Old	*2800	3991404	11949004
Sync	2800	346605	809243
New	2800	131235	365609
Dining Philosophers			
New	*2400	288650	805804
Sync	*2350	812694	1994856
New	2350	283150	790554
Old	*1150	3374711	10137875
Sync	1150	376831	927590
New	1150	138855	387680

Table 1. Results for the formula φ_0 generated by the SAT-based BMC translations

Although, in our opinion, testing the formula φ_0 for different systems allows to draw reliable conclusions about the presented encodings, we have also tested some for-

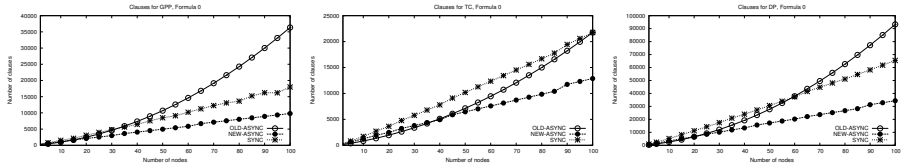


Fig. 4. Number of clauses for the formula φ_0.

mulae each of which expresses either a reachability or an extended reachability property (i.e. a formula of the form $\mathbf{E}(\mathbf{F}\psi_1 \wedge \ldots \wedge \mathbf{F}\psi_m)$, where $\psi_1, \ldots, \psi_m$ are propositional formulae). For all the tested systems we assume that for every local state s, $L(s) = \{s\}$.

For the GPP we have tested the following two formulae:

- $\varphi_1(n) = \mathbf{EF}(Received)$
- $\varphi_2(n) = \mathbf{EF}(Send_1 \wedge \ldots \wedge Send_n)$

Let us note that for the asynchronous parallel composition the witness for the formula $\varphi_1(n)$ has the length $2n + 2$, and the witness for the formula $\varphi_2(n)$ has the length $n^2 + 2n$, whereas for the synchronous parallel composition the witness for the formula $\varphi_1(n)$ has the length $2n + 2$, and the witness for the formula $\varphi_2(n)$ has the length $3n$. The experimental results for the tested formulae φ_1 and φ_2, are presented at the pictures 9 and 10 respectively.

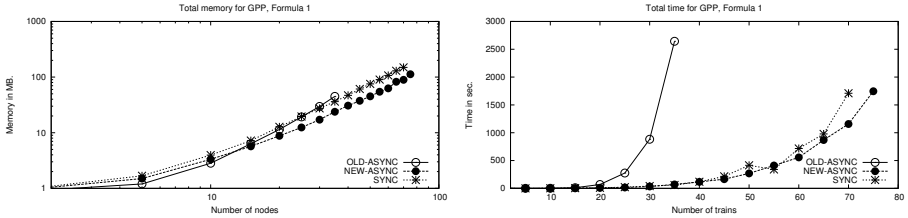


Fig. 5. Experimental results for GPP and the formula φ_1.

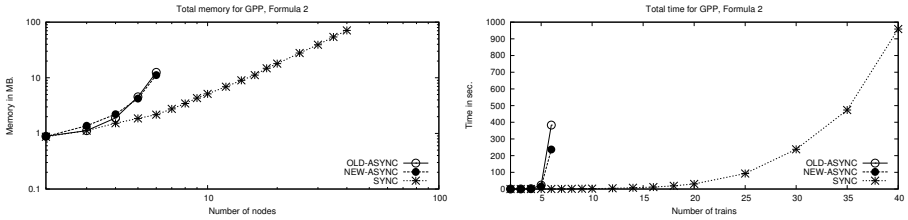


Fig. 6. Experimental results for GPP and the formula φ_2.

For the TC we have tested the following two formulae:

- $\varphi_3(n) = \mathbf{E}(\mathbf{F}(Tunnel_1) \wedge \mathbf{F}(Tunnel_n)$
- $\varphi_4(n) = \mathbf{E}(\mathbf{F}(Tunnel_1) \wedge \ldots \wedge \mathbf{F}(Tunnel_n))$

Let us note that for the asynchronous parallel composition the witness for the formula $\varphi_3(n)$ has the length 5, and the witness for the formula $\varphi_4(n)$ has the length $3(n-1)+2$, whereas for the synchronous parallel composition the witness for the formula $\varphi_4(n)$ has the length 4, and the witness for the formula $\varphi_4(n)$ has the length $2n$. The experimental results for the tested formulae φ_3 and φ_4, are presented at the pictures 7 and 8 respectively.

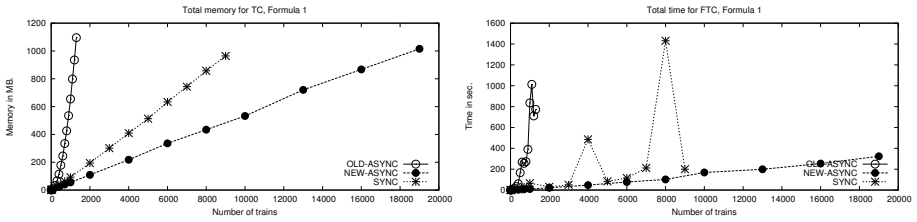


Fig. 7. Experimental results for TC and the formula φ_1.

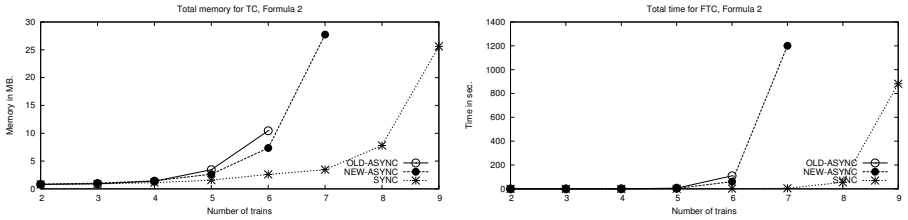


Fig. 8. Experimental results for TC and the formula φ_2.

For the DP we have tested the following two formulae:

- $\varphi_5(n) = \mathbf{E}(\mathbf{F}(Eat_0) \wedge \mathbf{F}(Eat_{n-1})$
- $\varphi_6(n) = \mathbf{E}(\mathbf{F}(Eat_0) \wedge \ldots \wedge \mathbf{F}(Eat_{n-1}))$

Let us note that for the asynchronous parallel composition the witness for the formula $\varphi_5(n)$ has the length 7, and the witness for the formula $\varphi_6(n)$ has the length $4n+1$, whereas for the synchronous parallel composition the witness for the formula $\varphi_5(n)$ has the length 6, and the witness for the formula $\varphi_6(n)$ has the length $n+3$ for $n > 6$ and 9 otherwise. The experimental results for the tested formulae φ_5 and φ_6, are presented at the pictures 7 and 8 respectively.

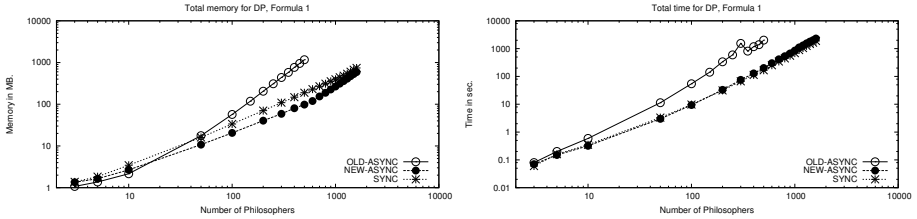


Fig. 9. Experimental results for DP and the formula φ_1.

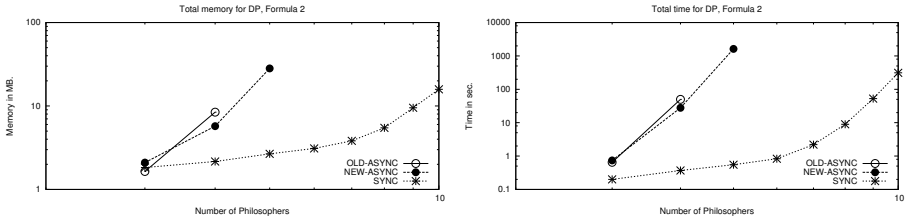


Fig. 10. Experimental results DP and for the formula φ_2.

5 Conclusions and Future Work

The experiments showed that our new Boolean encoding for asynchronous parallel composition is the most effective one while comparing to the other two considered. Moreover, the encoding of the transition relation for the synchronous parallel composition is nearly as effective as the new encoding for asynchronous parallel composition. The standard Boolean encoding used so far in the SAT-based verification modules of VerICS turned out to be the worst one. This is clearly seen for the formula $\varphi_0(n)$ that is verified on the path of the length 1. The length equal to 1 assures that the experimental results for the formula $\varphi_0(n)$ are affected by the Boolean encoding of the transition relation only.

The above conclusions, which are based on the experimental results presented in Section 4, are also supported by the fact that the length of the resulting Boolean formula for standard Boolean encoding for asynchronous parallel composition is $O(|Act| \cdot n)$, whereas the length of the resulting Boolean formula for our new Boolean encoding for asynchronous parallel composition is $O(|Act| \cdot \log |Act|)$.

Let us note that the experimental results for the formulae $\varphi_j(n)$, for $j = 1, \ldots, 6$, are affected not only by the Boolean encoding of the transition relation but also by the encoding of the translation of ECTL^* formulae to SAT. Nevertheless, the experimental results for these formulae confirm that the new encoding for asynchronous semantics significantly increases the efficiency of the SAT-based bounded model checking. It means that the effectiveness of the most of the SAT-based verification modules of VerICS could be significantly improved.

Moreover, the experimental results for the formulae $\varphi_2(n)$, $\varphi_4(n)$ and $\varphi_6(n)$ show that for formulae for which the length of the witness is significantly shorter for the

synchronous semantics, it is worth to apply the synchronous semantics instead of the asynchronous one.

We strongly believe that the choice of a SAT-solver will not change our conclusions. Nevertheless, in the nearest future we are going to repeat all the experiments for the other SAT-solvers.

References

1. Clarke, E.M.: The birth of model checking. In: 25 Years of Model Checking - History, Achievements, Perspectives. Volume 5000 of Lecture Notes in Computer Science., Springer (2008) 1–26
2. Clarke, E.M.: Model checking - my 27-year quest to overcome the state explosion problem. In: Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings. Volume 5330 of Lecture Notes in Computer Science., Springer (2008) 182
3. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Communications of the ACM **52**(11) (2009) 74–84
4. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
5. Zbrzezny, A.: Improvements in SAT-based reachability analysis for timed automata. Fundamenta Informaticae **60(1-4)** (2004) 417–434
6. Zbrzezny, A.: SAT-based reachability checking for timed automata with diagonal constraints. Fundamenta Informaticae **67(1-3)** (2005) 303–322
7. Kacprzak, M., Nabiałek, W., Niewiadomski, A., Penczek, W., Półrola, A., Szreter, M., Woźna, B., Zbrzezny, A.: VerICS 2007 - a model checker for knowledge and real-time. Fundamenta Informaticae **85(1-4)** (2008) 313–328
8. Zbrzezny, A.: Improving the translation from ECTL to SAT. Fundamenta Informaticae **85**(1-4) (2008) 513–531
9. Zbrzezny, A.: A new translation from ECTL* to SAT. Fundamenta Informaticae **120**(3-4) (2012) 377–397
10. Zbrzezny, A., Półrola, A.: SAT-based reachability checking for timed automata with discrete data. Fundamenta Informaticae **79(3-4)** (2007) 579–593
11. Męski, A., Penczek, W., Szreter, M., Woźna-Szcześniak, B., Zbrzezny, A.: BDD- versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: Algorithms and their performance. Autonomous Agents and Multi-Agent Systems (2013) to appear.
12. Biere, A.: Picosat essentials. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) **4** (2008) 75–97
13. Peled, D.: All from one, one for all: On model checking using representatives. In: Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93). Volume 697 of LNCS., Springer-Verlag (1993) 409–423
14. Hoek, W., Wooldridge, M.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia Logica **75(1)** (2003) 125–157
15. Dijkstra, E.: Hierarchical ordering of sequential processes. Acta Inf. **1** (1971) 115–138
16. Hoare, C.: Communicating sequential processes. Prentice Hall (1985)

Author Index