HUMBOLDT-UNIVERSITÄT ZU BERLIN

Concurrency, Specification and Programming
CS&P'2016

Rostock
September 28 - September 30, 2016

edited by Holger Schlingloff

Informatik-Bericht Nr. 247

INFORMATIK-BERICHTE

Herausgeber: Professoren des Institutes für Informatik
Redaktion: Publikationsstelle, Tel. (+49 30) 2093 3114
Druckerei: Humboldt-Universität zu Berlin
ISSN: 0863 - 095X

Die Reihe Informatik-Berichte erscheint aperiodisch.

Prof. Holger Schlingloff
Institut für Informatik
Sitz: Rudower Chaussee 25
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin

hs@informatik.hu-berlin.de

September 2016

Preface

This volume contains the papers presented at CS&P 2016, the 25^{th} International Workshop on Concurrency, Specification and Programming, held on September 28 - 30, 2016 in Rostock, Germany.

Since the early seventies Warsaw University and Humboldt University have alternately organized an annual workshop - since the early nineties known as CS&P. Over time, it has grown from a bilateral seminar to a well-known meeting attended also by colleagues from many other countries than Poland and Germany.

This year marks an anniversary: we celebrate the quarter-centenary edition of CS&P. We do so on the Baltic Sea coast, in one of the oldest universities in the world, the University of Rostock, founded in 1419. The gathering is hosted by Rostock University's department of computer science, and the editor would like to thank Prof. Karsten Wolf and his local team for their hospitality and great organization.

During the three-day meeting, there are 14 sessions in two parallel tracks. Additionally, there is a number of short presentations on current and emerging topics, as well as tool demos and open discussions.

This volume contains 26 papers supplementing the presentations, selected from the submissions by the program committee. Following the workshops tradition, we strive to retain an informal working atmosphere. Therefore, the proceedings includes drafts and extended abstracts as well as fully elaborated contributions.

The proceedings are published by Humboldt University and CEUR. The editor would like to thank the university's printing office, the team at CEUR Workshop Proceedings, and EasyChair for their help in producing this publication.

Berlin, September 2016 Holger Schlingloff

Table of Contents

Trying to understand PEG 1
Roman Redziejowski

Timed Processes of Interval-Timed Petri Nets 13
Elisabeth Pelz

On Generation of Time-based Label Refinements 25
Niek Tax, Emin Alasgarov, Natalia Sidorova and Reinder Haakma

Simple Bounded MTL Model Checking for Discrete Timed Automata (extended abstract) 37
Agnieszka Zbrzezny and Andrzej Zbrzezny

Rough Sets and Sorites Paradox 49
Andrzej Jankowski, Andrzej Skowron and Piotr Wasilewski

A version of rough mereology suitable for rough sets 61
Lech Polkowski

Rough Set Based Approximations of Classes in the OWL Ontology of Places in Poland (extended abstract) 68
Piotr Grochowalski, Krzysztof Pancerz and Tomasz Szul

Reversing Transitions in Bounded Petri Nets 74
Kamila Barylska, Łukasz Mikulski, Marcin Piatkowski, Maciej Koutny and Evgeny Erofeev

Towards Efficient Verification of Elementary Object Systems 86
Ismaila Jihad Abdullahi and Berndt Müller

Context-dependent Lexical and Syntactic Disambiguation in Ontology Population 101
Natalia Garanina and Elena Sidorova

Semantic Rendering of Data Tables: Multivalued Information Systems Revisited 113
Marcin Wolski and Anna Gomolinska

Superposition Principle in Composable Hybrid Automata 125
Jafar Akhundov, Peter Tröger and Matthias Werner

Extending Taylor Approximation to Hybrid Systems with Integrals 141
Ruggero Lanotte and Simone Tini

Analysing of M-AHIDS with future states on DARPA and KDD99 benchmarks 153
Mikulas Pataky and Damas Gruska

A graph-based reduction in Planics abstract planning, based on partial orders of services (extended abstract) 165
Maciej Szreter

TripICS - a Web Service Composition System for Planning Trips and Travels (extended abstract) 171
Artur Niewiadomski and Wojciech Penczek

An Algorithmic Way to Generate Simplexes for Topological Data Analysis . 180
Krzysztof Rykaczewski, Piotr Wisniewski and Krzysztof Stencel

Extrapolation of an Optimal Policy using Statistical Probabilistic Model Checking 189
Artur Rataj and Bozena Wozna-Szczesniak

A GPGPU-based Simulator for Prism: Statistical Verification of Results of PMC (extended abstract) 199
Marcin Copik, Artur Rataj and Bozena Wozna-Szczesniak

Process Environment Opacity 209
Damas Gruska

Coordinator Synthesis for Hierarchical Structure of Artificial Neural Network 221
Stanislaw Placzek

On the model checking of sequential reactive systems 233
Darya Kozlova and Vladimir Zakharov

Computing Bisimulation-Based Comparisons 245
Linh Anh Nguyen

Towards model checking argumentative dialogues with emotional reasoning (extended abstract) 257
Magdalena Kacprzak, Anna Sawicka and Andrzej Zbrzezny

Shapes of Concurrency 269
Piotr Chrzastowski-Wachtel

A Protocol of Mutual Exclusion for DSM Based on Vectors of Global Timestamps 274
Ludwik Czaja

Author and Keyword Index 286
(indexed by article number)

Organization

CS&P 2016 is organized by the Institut für Informatik, Humboldt Universität zu Berlin, in cooperation with Fraunhofer FOKUS, Berlin. Local organization is by the Universität Rostock.

Program Committee

Hans-Dieter Burkhard	Humboldt University
Ludwik Czaja	Institute of Informatics, Warsaw University
Anna Gomolinska	University of Bialystok, Institute of Informatics
Wojtek Jamroga	Polish Academy of Sciences
Magdalena Kacprzak	Politechnika Białostocka
Hung Son Nguyen	Institute of Mathematics, The University of Warsaw
Wojciech Penczek	ICS PAS and Siedlce University
Lech Polkowski	Polish-Japanese Institute of Information Technology
Louchka Popova-Zeugmann	Humboldt Universität, Institut für Informatik
Holger Schlingloff	Fraunhofer FOKUS and Humboldt University
Andrzej Skowron	Warsaw University
Zbigniew Suraj	Chair of Computer Science, University of Rzeszów
Marcin Szczuka	Institute of Mathematics, University of Warsaw
Matthias Werner	TU Chemnitz
Karsten Wolf	Universität Rostock

Additional Reviewers

Akhundov, Jafar	Pancerz, Krzysztof	Tröger, Peter
Grabowski, Adam	Salwicki, Andrzej	Zbrzezny, Andrzej
Knapik, Michał	Sawicka, Anna	
Niewiadomski, Artur	Szreter, Maciej	

Sponsoring Institution

Fraunhofer Institut für offene Kommunikationssysteme FOKUS, Berlin

Trying to understand PEG

Roman R. Redziejowski

roman@redz.se

Abstract. Parsing Expression Grammar (PEG) encodes a recursive-descent parser with limited backtracking. Its properties are useful in many applications. In its appearance, PEG is almost identical to a grammar in the Extended Backus-Naur Form (EBNF), but may define a different language. Recent research formulated some conditions under which PEG is equivalent to its interpretation as EBNF. However, PEG has a useful feature, namely syntactic predicate, that is alien to EBNF. The equivalence results apply thus only to PEG without predicates. The paper considers PEG with predicates. Not being able to investigate equivalence, the paper turns to the limited backtracking that is the main source of difficulty in understanding PEG. It is shown that the limitation of backtracking has no effect under conditions similar to those for PEG without predicates. There is, in general, no mechanical way to check these conditions, but they can be often checked by inspection. The paper outlines an experimental tool to facilitate such inspection.

1 Introduction

Parsing Expression Grammars (PEGs) have been introduced by Ford in [3] as a new formalism for describing syntax of programming languages. The formalism encodes a recursive-descent parser with limited backtracking. Backtracking removes the LL(1) restriction usually imposed on top-down parsers. The backtracking being limited makes it possible for the parser to work in a linear time, which is achieved with the help of "memoization" or "packrat" technology described in [1, 2].

In addition to circumventing the LL(1) restriction, PEG can be used to define parsers that do not require a separate "scanner" or "tokenizer". All this makes it useful, but PEG is not well understood as a language definition tool. Literature contains many examples of surprising behavior.

In its appearance, PEG is almost identical to a grammar in the Extended Backus-Naur Form (EBNF). Few minor typographical changes convert EBNF to PEG. As EBNF is familiar to most, one expects that the identically-looking PEG defines the same language. This is often the case, but the confusion comes when it is not.

In [6], the author tried to construct exact formulas for the language defined by a given PEG. This was a failure as the formulas became extremely complex with increasing complexity of the grammar.

In a pioneering work [5][1], Medeiros used "natural semantics" to describe both PEG and EBNF. Using this approach, he demonstrated that any EBNF grammar satisfying the LL(1) condition defines exactly the same language as its PEG counterpart. In [7], the author extended this result to a much wider class of grammars. However, there is no general way to mechanically check the conditions specified there. Some manual methods were suggested, based mainly on inspection.

PEG has one feature that does not have a counterpart in EBNF: the syntactic predicate. All results about the equivalence of PEG and EBNF must be thus restricted to PEG without predicates. And this is the case for all results from [5, 7]. But, predicates are useful in defining some features of the language like the distinction between keywords and identifiers.

This paper is an attempt to better understand PEG even in the presence of predicates. Because one can no longer investigate the equivalence of PEG and EBNF, we concentrate on the main source of confusion: the limited backtracking. We find that limited backtracking has no effect on choice expressions that can be identified as "disjoint". The conditions for disjointness are similar to those from [7], with LL(1) as the strongest one. Again, there is no general mechanical way to check disjointness. We outline an experimental tool, called *PEG Analyzer*, that combines the LL(1) test with heuristics to investigate disjointness of choice expressions.

We start by recalling, in Section 2, the definition of Parsing Expression Grammar and its formal semantics. In Section 3, we discuss limited backtracking and disjoint expressions. Section 4 introduces the PEG Analyzer with the help of three examples. The results obtained in Section 3 require a modification to the relation Follow known from the classical literature. This modification involves a rather tedious treatment not relevant for the main subject, so it is presented separately in Section 5. Finally, Section 6 contains few comments.

2 The grammar

We start with a simplified Parsing Expression Grammar $\mathbb{G}$ over alphabet Σ. The grammar is a set of *rules* of the form $A = e$ where A belongs to a set N of symbols distinct from the letters of Σ and e is an *expression*. Each expression is one of these:

$$\begin{array}{llll} \varepsilon & (\text{"empty"}), & !\,e & (\text{"predicate"}), \\ a \in \Sigma & (\text{"terminal"}), & e_1 e_2 & (\text{"sequence"}), \\ A \in N & (\text{"nonterminal"}), & e_1 |\, e_2 & (\text{"choice"}), \end{array}$$

where each of e_1, e_2, e is an expression. The set of all expressions is in the following denoted by $\mathbb{E}$. There is exactly one rule $A = e$ for each $A \in N$. The expression e appearing in this rule is denoted by $e(A)$. The predicate operator binds stronger than sequence and sequence stronger than choice.

[1] This work is in Portuguese. An extended English version is available in [4].

The expressions represent parsing procedures, and rules represent named parsing procedures. In general, parsing procedure is applied to an input string from Σ^* and tries to recognize an initial portion of that string. If it succeeds, it returns "success" and usually consumes the recognized portion. Otherwise, it returns "failure" and does not consume anything. The actions of different procedures are specified in Figure 1.

ε	Indicate success without consuming any input.
a	If the text ahead starts with a, consume a and return success. Otherwise return failure.
A	Call $e(A)$ and return result.
$!\,e$	Call e. Return failure if succeeded. Otherwise return success without consuming any input.
$e_1\, e_2$	Call e_1. If it succeeded, call e_2 and return success if e_2 succeeded. If e_1 or e_2 failed, backtrack: reset the input as it was before the invocation of e_1 and return failure.
$e_1 \mid e_2$	Call e_1. Return success if it succeeded. Otherwise call expression e_2 and return success if e_2 succeeded or failure if it failed.

Fig. 1. Actions of expressions as parsing procedures

We note the limited backtracking: once e_1 in $e_1 \mid e_2$ succeeded, e_2 will never be tried. The backtracking done by the sequence expression may only roll back $e_1 \mid e_2$ as a whole.

The actions of parsing procedures can be formally defined using "natural semantics" introduced in [4,5]. For $e \in \mathbb{E}$, we write $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ to mean that e applied to string xy consumes x, and $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ to mean that e fails when applied to x. One can see that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$, respectively $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$, holds if and only if it can be formally proved using the inference rules shown in Figure 2.

The PEG parser may end up in an infinite recursion, the well-known nemesis of top-down parsers. Formally, it means that there is no proof according to the rules of Figure 2. It has been demonstrated that if the grammar $\mathbb{G}$ is free from left-recursion, then for every $e \in \mathbb{E}$ and $x \in \Sigma^*$ there exists a proof of $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ or $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} y$ for some $y \in \Sigma^*$. This has been shown in [4,5] by checking that PEG defined by natural semantics is equivalent to that defined by Ford in [3] and using the result from there. An independent proof for grammar without predicates is given in [7]. It is easily extended to grammar with predicates. We assume from now on that $\mathbb{G}$ is free from left-recursion.

For $e \in \mathbb{E}$, we denote by $\mathcal{L}(e)$ the set of words $x \in \Sigma^*$ such that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ for some $y \in \Sigma^*$. This is the language accepted by e. Note that, in general, $x \in \mathcal{L}(e)$ does not mean $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ for each y.

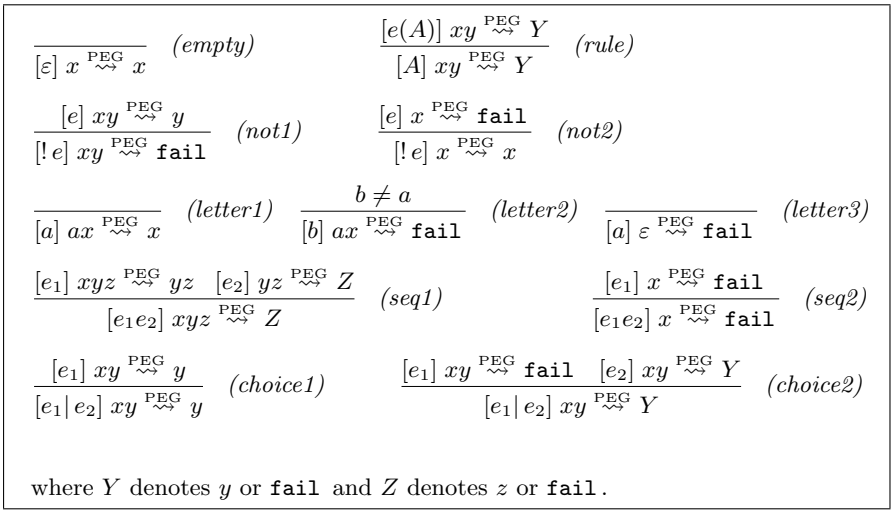


Fig. 2. Formal semantics of PEG

We define the EBNF interpretation of $\mathbb{G}$ as the language $\mathcal{L}^E(e)$ accepted by expression $e \in \mathbb{E}$. It is defined recursively as

$$\begin{aligned} &\mathcal{L}^E(\varepsilon) = \{\varepsilon\}, && \mathcal{L}^E(!\,e) = \{\varepsilon\}, \\ &\mathcal{L}^E(a) = \{a\}, && \mathcal{L}^E(e_1 e_2) = \mathcal{L}^E(e_1)\mathcal{L}^E(e_2), \\ &\mathcal{L}^E(A) = \mathcal{L}^E(e(A)), && \mathcal{L}^E(e_1 |\, e_2) = \mathcal{L}^E(e_1) \cup \mathcal{L}^E(e_2). \end{aligned}$$

By defining $\mathcal{L}^E(!\,e) = \{\varepsilon\}$, we extended the interpretation to PEG with predicates. This is not what one can expect looking at the grammar, just an approximation. The following result from [4, 5] is easily extended to our interpretation of predicates:

$$\mathcal{L}(e) \subseteq \mathcal{L}^E(e) \text{ for any } e \in \mathbb{E}\,. \tag{1}$$

The opposite of (1) does not, in general hold. This is, to some extent, due to the different interpretation of predicates, but the main cause is limited backtracking.

3 Disjoint choice and limited backtracking

We say that choice $e = e_1 |\, e_2$ is *strictly disjoint* if

$$\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\Sigma^* = \varnothing. \tag{2}$$

Such choice is not affected by the limitation of backtracking. Indeed, suppose that e is applied to some string s and e_1 succeeds. This means $s \in \mathcal{L}(e_1)\Sigma^*$.

After this, any attempt to backtrack must result in a failure. It would mean applying e_2 to s, and a success would mean $s \in \mathcal{L}(e_2)\Sigma^*$, which is excluded by (2). So, not attempting it does not make any difference.

The choice `aa|a` is not disjoint in the sense of (2). However, it is not affected by partial backtracking when it appears in some contexts, for example, in `(aa|a)b`. A success of `aa` means that input has the form $\texttt{aa}\Sigma^*$, so backtracking to `a` is useless as it will later result in a failure by not finding `b`.

We are going to look at limited backtracking in the context of our grammar $\mathbb{G}$ successfully parsing a complete input string. Thus, we assume that $\mathbb{G}$ has a unique *start symbol* $S \in N$ with the corresponding rule $S = e\,\texttt{\#}$ where e is an expression and `#` is a unique end-of-text marker that appears only in this rule. We say that a string $w \in \Sigma^*$ is *accepted* by S to mean that $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$.

For an expression $e \in \mathbb{E}$, we define $\text{Tail}(e)$ to be the set of strings y such that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ appears as partial result in the proof of $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ for some w. We say that the choice $e = e_1|\,e_2$ is *disjoint (in the context of* $\mathbb{G}$*)* if

$$\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\,\text{Tail}(e) = \varnothing. \tag{3}$$

The same argument as before shows that such choice is not affected by the limitation of backtracking. If e is applied to some string s in a proof $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ and e_1 succeeds, we have $s \in \mathcal{L}(e_1)\Sigma^*$. An attempt to backtrack would mean applying e_2 to s, and a success would mean $s \in \mathcal{L}(e_2)\,\text{Tail}(e)$, which is excluded by (3).

A mechanical checking of (3) is in general impossible because of complexity of $\mathcal{L}(e)$ and $\text{Tail}(e)$, and because it may involve checking emptiness of intersection of context-free languages - known to be undecidable. However, it can be often checked using approximation.

With (1), we can approximate $\mathcal{L}(e_1)$ and $\mathcal{L}(e_2)$ by $\mathcal{L}^E(e_1)$ respectively $\mathcal{L}^E(e_2)$. This gives a stronger condition:

$$\mathcal{L}^E(e_1)\Sigma^* \cap \mathcal{L}^E(e_2)\,\text{Tail}(e) = \varnothing. \tag{4}$$

It was shown in [7] that if this condition holds for all choice expressions in a grammar without predicates, we have $\mathcal{L}(e) = \mathcal{L}^E(e)$ for all $e \in \mathbb{E}$.

To approximate $\text{Tail}(e)$, we need to modify the relation Follow known from the classical literature. The modification is described in Section 5, where it is shown (Proposition 1) that with the modified relation, we have $\text{Tail}(e) \subseteq \mathcal{L}^E(\text{Follow}(e))\Sigma^*$ where $\mathcal{L}^E(\text{Follow}(e)) = \cup_{x \in \text{Follow}(e)}\mathcal{L}^E(x)$. This gives an even stronger condition:

$$\mathcal{L}^E(e_1)\Sigma^* \cap \mathcal{L}^E(e_2)\mathcal{L}^E(\text{Follow}(e))\Sigma^* = \varnothing. \tag{5}$$

Using known methods one can compute the sets of symbols that appear as first in strings from $\mathcal{L}^E(e_1)\Sigma^*$ respectively $\mathcal{L}^E(e_2)\mathcal{L}^E(\text{Follow}(e))\Sigma^*$. Condition (5) is then obviously satisfied if these sets are disjoint. This is the familiar LL(1) condition.

As suggested in [7, 8], one can obtain a weaker condition by approximating $\mathcal{L}^E(e_1)\Sigma^*$ and $\mathcal{L}^E(e_2)\mathcal{L}^E(\text{Follow}(e))\Sigma^*$ with sets of the form $F\Sigma^*$ where F is some suitably chosen subset using the classical relation First. Some ways of choosing F have been suggested, but they can not, in general, be mechanized. Another approximation was suggested by Schmitz in [9].

The grammar $\mathbb{G}$ considered up to now is a simplified version of full PEG. This latter allows expressions such as $e_1 | e_2 | \ldots | e_n$, $e_1 e_2 \ldots e_n$, e^*, e^+, and $e?$. The expression $E = e_1 | e_2 | \ldots | e_n$ is a syntactic sugar for $E = e_1 | E_1$, $E_1 = e_2 | E_2$, $\ldots$, $E_n = e_n$ so (3) must hold for all of $E, E_1, \ldots, E_{n-1}$. One can verify that this is true if

$$\mathcal{L}(e_i)\Sigma^* \cap \mathcal{L}(e_j)\,\text{Tail}(E) = \varnothing \quad \text{for } 1 \le i < j < n. \tag{6}$$

The expressions $E = e^*$, $E = e^+$, and $E = e?$ constitute syntactic sugar for, respectively $E = eE/\varepsilon$, $E = eE/e$, and $E = e/\varepsilon$ so (3) must hold for each of them. One can verify that this is true if

$$\mathcal{L}(e)\Sigma^* \cap \text{Tail}(E) = \varnothing. \tag{7}$$

The rules for computing Follow(e) given in the Section 5 can be similarly extended to the full PEG.

The terminals in full PEG are not necessary single letters, and may be multi-letter quoted strings. Instead of sets of "first letters" used in the test for LL(1), one has to compute sets of "first terminals" and check their disjointness.

4 PEG Analyzer

Giving up all hope for an automatic verification of (3), the author created an experimental tool, the *PEG Analyzer*, that combines the LL(1) check with heuristics. It takes a grammar, tests all choice expressions for LL(1) using (5), and presents for inspection those that did not pass the test. To facilitate inspection, it gradually expands the involved expressions by replacing them with their definitions, somewhat in the spirit of what was suggested in [7, 8].

4.1 Example 1: Simple calculator

To give some idea of the Analyzer, we apply it to the grammar shown in Figure 3. The grammar defines the syntax of a simple calculator.

When Analyzer is applied to this grammar, it indicates that the choice between the first two alternatives of `Factor` does not satisfy LL(1), and opens a window shown in Figure 4. It is an invitation to verify (3) for e_1 = `Digits? Fraction`, e_2 = `Digits`, and Tail(`Factor`).

The first two lines show these two expressions. The second is, in fact, a pseudo-expression, with pseudo-expression `Tail(Factor)` representing the tail. The third line tells that both expressions have `[0-9]` as "first terminal".

```
Start    = Sum #
Sum      = Product (AddOp Product)*
Product  = Factor (MultOp Factor)*
Factor   = Digits? Fraction | Digits | Lparen Sum Rparen
Fraction = "." Digits
AddOp    = [-+]
MultOp   = [*/]
Lparen   = "("
Rparen   = ")"
Digits   = [0-9]+
```

Fig. 3. A simple calculator

```
Factor.1 = Digits? Fraction
Factor.2 = Digits Tail(Factor)
  [0-9]  <==>  [0-9]

Digits? Fraction
  Digits Fraction
  [0-9] [0-9]* "." Digits
    <==>
Digits Tail(Factor)
Digits (AddOp Product | MultOp Factor | Rparen) ...
[0-9] [0-9]* (AddOp Product | MultOp Factor | Rparen) ...
```

Fig. 4. Presentation of a non-LL(1) case

The subsequent lines show the two expressions in more detail. Thus, the first expression stands for two alternative expressions, `Digits Fraction` and `Fraction`. Since this latter does not start with `[0-9]`, only `Digits Fraction` appears, with an indentation showing that it is one of alternatives. In the next line, `Digits` is expanded following its definition to `[0-9] [0-9]*` and `Fraction` to `"." Digits`. The result is supposed to give an idea of strings starting with `[0-9]` that are accepted by `Digits? Fraction`.

In the second expression, `Tail(Factor)` is replaced by the approximation $\mathcal{L}^E(\text{Follow}(\texttt{Factor}))\Sigma^*$ in the form of pseudo-expression. Again, `Digits` is expanded to `[0-9] [0-9]*`.

Verifying (3) means checking if any string represented by e_1 can be a prefix of any string in $\mathcal{L}(e_2)\,\text{Tail}(\texttt{Factor})$. One can easily see that `Digits` in the first expression is always followed by a dot, while in the second it can be only followed by `AddOp`, `MultOp`, or `Rparen`, none of which is a dot. The condition (3) is thus satisfied.

4.2 Example 2: Grammar with predicates

The second example illustrates treatment of predicates. The grammar in Figure 5 is a fragment of larger grammar that uses identifiers, with some of them being reserved as "keywords". Only one keyword, `"print"` is shown. Its definition is followed by `! Letter` to make sure it is not recognized as a prefix of an identifier. The definition of `Identifier` is preceded by `! Keyword` to ensure that keyword is not recognized as an identifier.

```
Statement  = (Keyword Number | Identifier Number) ";" #
Keyword    = "print" !Letter
Identifier = !Keyword Letter+
Letter = [a-z]
Number = [0-9]+
```

Fig. 5. Grammar with predicates

The Analyzer applied to this grammar indicates that the choice in `Statement` does not satisfy LL(1), and opens the window shown in Figure 6.
To check LL(1), the Analyzer approximated $\mathcal{L}$(`"print"! Letter`) with $\mathcal{L}^E$(`"print"`) and $\mathcal{L}$(`! Keyword Letter+`) with $\mathcal{L}^E$(`Letter+`):

$$\mathcal{L}(\texttt{"print"! Letter}) \subseteq \mathcal{L}^E(\texttt{"print"!Letter}) = \mathcal{L}^E(\texttt{"print"})\,,$$
$$\mathcal{L}(\texttt{! Keyword Letter+}) \subseteq \mathcal{L}^E(\texttt{! Keyword Letter+}) = \mathcal{L}^E(\texttt{Letter+})\,.$$

It found their first terminals to be, respectively, `"print"` and `[a-z]`. As they are not disjoint, the choice does not satisfy LL(1) and is signaled as such. But, this is a false alarm. One can easily see that

$$\mathcal{L}(\texttt{"print" ! Letter ...}) \cap \mathcal{L}(\texttt{! Keyword Letter+ ...}) = \varnothing$$

showing that the expression satisfies (3).

```
Statement.1.1 = Keyword Number
Statement.1.2 = Identifier Number Tail(Statement.1)
  "print"  <==>  [a-z]

Keyword Number
"print" !Letter Number
  <==>
Identifier Number Tail(Statement.1)
Identifier Number ";" ...
!Keyword Letter+ Number ";" ...
```

Fig. 6. Presentation of a non-LL(1) case

4.3 Example 3: Non-disjoint expressions

Suppose now that in the calculator from Example 1, we want sometimes to skip the multiplication sign and write, for example `2(.3+4)` instead of `2*(.3+4)`. To achieve this, we replace the definition of `MultOp` by `MultOp = "*"? | "/"`.

The Analyzer applied to the modified grammar shows now two cases not satisfying LL(1). The first produces the window shown in Figure 7. It says that `[0-9]` in `[0-9]+` is followed by something that may start with `[0-9]`, namely any of two different alternatives of `Factor` after omitted first alternative of `MultOp`. Clearly, `[0-9]` is a prefix of each alternative. The condition (3) is not satisfied.

```
Digits.1 = [0-9]
Tail(Digits)
  [0-9]  <==>  [0-9]

 [0-9]
   <==>
 Tail(Digits)
 MultOp Factor ...
   Factor ...
     Digits? Fraction ...
       Digits Fraction ...
       [0-9] [0-9]* Fraction ...
     Digits  ...
     [0-9] [0-9]* ...
```

Fig. 7. Presentation of a non-LL(1) case

How does this happen and what does it mean? A look at the grammar shows that the offending `[0-9]+` is one appearing at the end of the first `Factor` in `Factor (MultOp Factor)*`. With omitted `MultOp` it will gobble up any digits at the beginning of the second `Factor`, without any attempt to backtrack. Thus, for example, `234` will be treated by PEG as a single `Factor`, and not as shorthand for `2*3*4`.

In this example, the grammar interpreted as EBNF has an ambiguity, and PEG just selects one of the possible parses.

The second case not satisfying LL(1) is the same as for the original grammar: the choice between the first two alternatives of `Factor`, and is reported exactly as shown in Figure 4. However, `MultOp` in `MultOp Factor` that appears in the tail of `Factor` may be omitted. And `Factor` has an alternative that begins with a dot. Thus, `Digits` in `Digits Tail(Factor]` can be followed by a dot and (3) is not satisfied. It means that, for example, `2.34` will be treated by PEG as a single `Factor`, and not as shorthand for `2*.34`.

Here the grammar interpreted as EBNF has another ambiguity and PEG chooses one possible parse. In both cases, we may accept the PEG's choice because it corresponds to the perception of a human reader.

5 Computation of Follow

We define a number of relations $R \subseteq \mathbb{E} \times \mathbb{E}$, writing $e' \in R(e)$ to mean $(e, e') \in R$.

- $\text{Derive}_{ss}(e)$ is the set of all $e' \in \mathbb{E}$ such that $[e']\ x'y \overset{\text{PEG}}{\rightsquigarrow} y$ can be derived from $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using one inference rule.
  Thus, $e' \in \text{Derive}_{ss}(e)$ if and only if:
  - $e' = A \in N$ where $e(A) = e$,
  - $e' = e\, e_2$ for some e_2 such that $\varepsilon \in \mathcal{L}(e_2)$,
  - $e' = e_1\, e$ for some e_1,
  - $e' = e |\, e_2$ for some e_2,
  - $e' = e_1 |\, e$ for some e_1.
- $\text{Derive}_{sf}(e)$ is the set of all $e' \in \mathbb{E}$ such that $[e']\ xy \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ can be derived from $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using one inference rule.
  Thus, $e' \in \text{Derive}_{sf}(e)$ if and only if:
  - $e' = !\, e$,
  - $e' = e\, e_2$ for some e_2.
- $\text{Derive}_{ff}(e)$ is the set of all $e' \in \mathbb{E}$ such that $[e']\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ can be derived from $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ using one inference rule.
  Thus, $e' \in \text{Derive}_{ff}(e)$ if and only if:
  - $e' = A \in N$ where $e(A) = e$,
  - $e' = e_1\, e$ for some e_1,
  - $e' = e\, e_2$ for some e_2,
  - $e' = e |\, e_2$ for some e_2,
  - $e' = e_1 |\, e$ for some e_1 that can fail.
- $\text{Next}_s(e)$ is the set of all $e' \in \mathbb{E}$ such that $\varepsilon \notin \mathcal{L}(e')$ and there exists $e\, e' \in \mathbb{E}$.
- $\text{Next}_f(e) = \{\varepsilon\}$ for all e such that there exists $!\, e \in \mathbb{E}$ or $e |\, e_2 \in \mathbb{E}$ for some e_2.

Define $\text{Follow} = \text{Derive}_{ss}^* \times \text{Next}_s \cup \text{Derive}_{ss}^* \times \text{Derive}_{sf} \times \text{Derive}_{ff}^* \times \text{Next}_f$.

Proposition 1. *For each partial result* $[e]\ xy \overset{PEG}{\rightsquigarrow} y$ *in the proof of* $[S]\ w \overset{PEG}{\rightsquigarrow} \varepsilon$ *holds* $y \subseteq \mathcal{L}^E(\text{Follow}(e))\Sigma^*$ *where* $\mathcal{L}^E(\text{Follow}(e)) = \bigcup_{e' \in \text{Follow}(e)} \mathcal{L}^E(e')$.

Proof. Consider any partial result $[E_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ in the proof of $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$. It is the first in a chain of $n \geq 1$ partial results derived successively using the rules of Figure 2 and other partial results. The chain ends with final result $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ derived from $[E_n\, \texttt{\#}]\ x_n\, \texttt{\#} \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$. In this chain, the first $j \geq 1$ partial results are of the form $[E_i]\ x_i y \overset{\text{PEG}}{\rightsquigarrow} y$. By definition of Derive_{ss}, we have $E_i \in \text{Derive}_{ss}^*(E_1)$ for $1 \leq i \leq j$. The first partial result in a different form must be one of these:

(a) $[E_j\, e_2]\ x_j uv \overset{\text{PEG}}{\rightsquigarrow} v$ where $y = uv$, $u \neq \varepsilon$, and $[e_2]\ uv \overset{\text{PEG}}{\rightsquigarrow} v$.
(b) $[!\, E_j]\ x_j y \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$.
(c) $[E_j\, e_2]\ x_j y \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ where $[e_2]\ x_j y \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$.

In case (a) we have $e_2 \in \mathrm{Next}_s(E_j)$, so $e_2 \in (\mathrm{Derive}^*_{ss} \times \mathrm{Next}_f)(E_1) \subseteq \mathrm{Follow}(E_1)$. We have also $y = uv$ where $u \in \mathcal{L}(e_2) \subseteq \mathcal{L}^E(e_2) \subseteq \mathcal{L}^E(\mathrm{Follow}(E_1))$, so $y \in \mathcal{L}^E(\mathrm{Follow}(E_1))\Sigma^*$.

In cases (b) and (c), we have partial result $[E_{j+1}]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$. According to the definition of Derive_{sf}, we have $E_{j+1} \in \mathrm{Derive}_{sf}(E_j)$. It is first in the chain of $k \geq 1$ partial results in the form $[E_i]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}$ derived successively using the rules of Figure 2 and other partial results. By definition of Derive_{ff}, we have $E_i \in \mathrm{Derive}^*_{ff}(E_j)$ for $j+1 \leq i \leq j+k$. The first partial result in a different form must be one of these:

(d) $[!\,E_{j+k}]\ uv \overset{\text{PEG}}{\rightsquigarrow} uv$.
(e) $[E_{j+k}|\,e_2]\ uv \overset{\text{PEG}}{\rightsquigarrow} v$ where $[e_2]\ uv \overset{\text{PEG}}{\rightsquigarrow} v$,

where $uv = x$. We only know that the original y is a suffix of uv, but is very unlikely to be the same as v. We can only approximate it as $y \in \Sigma^*$. In each of (d)-(e), we have $\mathrm{Next}_f(E_{j+k}) = \{\varepsilon\}$, so

$$\{\varepsilon\} = (\mathrm{Derive}^*_{ss} \times \mathrm{Derive}_{sf} \times \mathrm{Derive}^*_{ff} \times \mathrm{Next}_f)(E_1) \subseteq \mathrm{Follow}(E_1).$$

We have $\varepsilon \in \mathcal{L}^E(\mathrm{Follow}(E_1))$, so $y \in \mathcal{L}^E(\mathrm{Follow}(E_1))\Sigma^*$. □

Note that the very rough approximation of $\mathrm{Tail}(e)$ by Σ^*, changing (5) into a strict disjointness, applies to e that appears in a partial proof resulting in a failure (which is eventually needed to prove $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$).

6 Final remarks

The fact that EBNF does not have predicates spoils the useful correspondence with PEG. The basic conditions for absence of effects of limited backtracking remain in principle unchanged, but there is no way of talking about equivalence.

There is no way to just include the recognition-oriented predicates as part of the construction-oriented EBNF. But one may consider some more natural extensions to EBNF that could serve the same purpose as predicates in defining useful grammars.

The example of PEG Analyzer shows that disjointness can often be checked by inspection. The tool as described here is quite primitive. One can improve it by letting the user interactively choose specific parts of expressions for detailed inspection.

The subject for further research is a careful analysis of what happens in the case of non-disjoint choice.

References

1. Ford, B.: Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking. Master's thesis, Massachusetts Institute of Technology (Sep 2002), `http://pdos.csail.mit.edu/papers/packrat-parsing:ford-ms.pdf`
2. Ford, B.: Packrat parsing: simple, powerful, lazy, linear time, functional pearl. In: Wand, M., Jones, S.L.P. (eds.) Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002. pp. 36–47. ACM (2002)
3. 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. pp. 111–122. ACM, Venice, Italy (14–16 January 2004)
4. Mascarenhas, F., Medeiros, S., Ierusalimschy, R.: On the relation between context-free grammars and Parsing Expression Grammars. Science of Computer Programming 89, 235–250 (2014)
5. Medeiros, S.: Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto. Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro (Aug 2010)
6. Redziejowski, R.R.: Some aspects of Parsing Expression Grammar. Fundamenta Informaticae 85(1–4), 441–454 (2008)
7. Redziejowski, R.R.: From EBNF to PEG. Fundamenta Informaticae 128, 177–191 (2013)
8. Redziejowski, R.R.: More about converting BNF to PEG. Fundamenta Informaticae 133(2-3), 177–191 (2014)
9. Schmitz, S.: Modular syntax demands verification. Tech. Rep. I3S/RR-2006-32-FR, Laboratoire I3S, Université de Nice - Sophia Antipolis (Oct 2006)

Timed Processes of Interval-Timed Petri Nets

Elisabeth Pelz

LACL, Université Paris-Est Créteil, France
pelz@u-pec.fr

Abstract. In this paper we use partial order semantics to express the truly concurrent behaviour of interval-timed Petri nets (ITPNs) in their most general setting, i.e. with autoconcurrency and zero duration, as studied with its standard maximal step semantics in [**?**]. First we introduce the notion of timed processes for ITPNs inductively. Then we investigate if the equivalence of inductive and axiomatic process semantics - true for classical Petri nets - could hold for ITPNs too. We will see that the notions of independence and immediate firing obligation seem to be antagonistic ones, and that local axioms, adequate to define processes of classical Petri nets, are not sufficient to caracterize timed Processes of IITPNs. We propose several original "global" axioms which reveal to be an effective solution. Thus we yield finally a full axiomatic definition of timed processes for ITPNs.

1 Introduction

Petri nets are an algebraic and graphical formalism, proposed by Carl Adam Petri [**?**], used to represent complex interactions and activities in a system, they model situations like synchronisation, sequentiality, concurrency and conflict. Classical Petri nets do not carry any time information and so they are not suitable for quantitative analysis of the performance and reliability of systems with respect to time.

Several time extensions of Petri nets have been proposed in the literature, [**?**,**?**]. In this paper, we consider Interval-Timed Petri Nets (ITPNs) with autoconcurrency which are a generalisation of Timed Petri Nets. The main feature of Interval-Timed Petri Nets is that transitions which are enabled need to start immediately their firing, and the firing lasts some time within an (integer) interval. Thus in the observation the startfiring and the endfiring of a transition are considered as two distinct events. Furthermore, in the class of ITPNs we consider here, we allow transitions to take no time (i.e. to have zero duration). This obligation of immediate firing led to the standard execution of Timed Petri Nets in (simultanous) maximal steps.

Let us state precisely the concepts which lead to the describtion of the behaviour of ITPNs under maximal step semantics, cf. [**?**]. Each transition has its own clock. The progress of time is managed by a global clock by means of (discrete) ticks which increment all local clocks. Inbetween two ticks, we must fire as many transitions as possible, i.e., for a maximal number of enabled transitions there are startfiring events; but also, we have to endfire every fired transition

which reach its maximal duration (i.e., which must endfire) plus an arbitrary multiset of fired transitions which may endfire. The maximal multiset of events which occur between two time ticks is called a global step. In [**?**] such standard semantics of ITPNs in terms of firing step sequences are exhaustively defined.

By convention, we only consider ITPNs with zero duration which are well-formed, as in [**?**], i.e. where no infinite global step is possible.

Partial order semantics allow to describe the behaviour of concurrent systems by expressing explicitly concurrency, seen as independency. Processes are the usual partial order semantics for classical Petri nets [**?**,**?**]. They are also defined for time Petri nets [**?**,**?**,**?**] as well as for high level Timed Petri nets (with one-safe markings) [**?**]. Processes of classical Petri Nets can be defined by axiomatic definitions or by inductive ones using firing sequences, as discussed in [**?**].

Inspired by this approach, we define in this paper timed processes for ITPNs, inductively using firing step sequences. To our knowledge processes have never been introduced for this class. Our definitions will be coherent with the approach in [**?**], albeit arbitrary markings and auto-concurrency introduce new challenges. We propose a way to respect also the above quoted concepts of immediate firing obligation and global tick in the inductive definition.

But when trying to define timed processes axiomatically, some antagonism appears. Let us remind that the axiomatic definition (for classical Petri Nets) states local properties which are true for all events in the process, independently of all other events and independently of any particular cut (or marking). Now, for ITPNs, events have to satisfy global contraints too, they are no longer independent but inter-dependent and cuts (before ticks) will play the role of synchronisation barriers. In particular, each tick event depends on the set of events which precede it. We will see the limits of an axiomatization where only local properties are defined and illustrate them with an example. Then gradually the global constraints are discussed and formulated in "global" axioms. Such global axioms are something original in Petri Net semantics. We succeed to give them in first order logic without quantification on sets (or cuts). Finally we are able to present a group of local and global axioms which form a total axiomatization of timed processes.

The remaining of the paper is organized as follows: Section 2 contains formal definitions about Interval-Timed Petri Nets. In Section 3 basic definitions around partial order structures can be found. Section 4 gives an inductive definition of timed processes and Section 5 details the discussion and formulation of an axiomatic one. Section 6 presents some concluding remarks.

2 Some Definitions

Let us start to define classical Petri nets.

Definition 1 (***Petri Nets***)
A Petri net is a 3-tuple $N = (P, T, v)$ *such that*
- P *and* T *are finite sets of places and transitions respectively with* $P \cap T = \emptyset$
- $v : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ *is its valuation function.* □

The states of Petri nets are described by *markings* $M : P \longrightarrow \mathbb{N}$ which are represented by vectors of dimension $|P|$.

Let x be a node such that, $x \in P \cup T$. The preset of x is denoted by ${}^\bullet x$, with ${}^\bullet x = \{y \in P \cup T \mid v(y,x) > 0\}$. Similarly, $x^\bullet$ denotes the postset of x, with $x^\bullet = \{y \in P \cup T \mid v(x,y) > 0\}$. In the same manner, the preset of a net N is defined by ${}^\bullet N = \{x \in P \cup T \mid {}^\bullet x = \emptyset\}$, the postset of a net N is defined by $N^\bullet = \{x \in P \cup T \mid x^\bullet = \emptyset\}$.

Formally, a multiset U of events E is a mapping $U : E \to \mathbb{N}$, such that, for $e \in E$ the natural number $U(e)$ is called the multiplicity of e. The multiset U can be written in the extended set notation $U = \{e^{U(e)} \mid e \in E \text{ and } U(e) \neq 0\}$.

Several time extensions of classical Petri nets have been proposed to integrate temporal modeling properties, time Petri nets [**?**], timed Petri nets [**?**] and causal time Petri nets [**?**].

In this paper, Interval-Timed Petri Nets (ITPNs) are considered in their most general setting, i.e. allowing zero duration and autoconcurrency. ITPNs are an extension of Timed Petri Nets in which the firing duration of each transition is given within an interval. We recall shortly the definitions of [**?**], which contains much more details and examples.

D*efinition 2** (Interval-Timed Petri Nets***)
An Interval-Timed Petri Net is a 5-tuple $\mathcal{N} = (P, T, v, M_0, I)$ *such that*
- (P, T, v) *is a Petri net, called skeleton of the net* $\mathcal{N}$
- $M_0 : P \longrightarrow \mathbb{N}$ *is its initial marking*
- $I : T \longrightarrow \big[\mathbb{N}, \mathbb{N}\big]$ *is its interval function.* □

We suppose that the set of transitions is enumerated: $T = \{t_1, t_2, .., t_{|T|}\}$. The time interval associated with a transition t is given by
$I(t) = \big[sfd(t), lfd(t)\big]$, where $sfd(t)$ is called the *shortest firing duration* and $lfd(t) \geq sfd(t)$ the *longest firing duration.*
Only ITPNs are considered whose transitions have a non empty preset and postset i.e., for each transition $t \in T$ holds that $|\,{}^\bullet t| > 0$ and $|t^\bullet| > 0$.

In Interval-Timed Petri Nets a marking is not sufficient to describe completely the state of a net. The state must also include temporal informations. This is given by a matrix which codes the transitions clocks.

D*efinition 3** (State***)
A state of an ITPN $\mathcal{N} = (P, T, v, M_o, I)$ *is a pair* $S = (M, h)$ *such that*
- M *is a marking.*
- h *is a clock matrix which has* $|T|$ *rows and* d *columns s.t.* $d = \max\limits_{t_i \in T} \big(lfd(t_i) + 1\big)$.
The value $h_{i,j+1}$ *represents the number of* ***active*** *transitions* t_i *with age* j *(i.e.* ***fired*** *since* j *time ticks).* □

The set of all possible states of $\mathcal{N}$ is denoted by $States(\mathcal{N})$. The initial state of $\mathcal{N}$ is denoted $S_0 = (M_0, h_0)$ where M_0 is the *initial marking* of the skeleton and h_0 is a zero matrix, i.e. no transition is active.

2.1 Firing rules for ITPNs

Definition 4 *(**Autoconcurrently enabled transitions**)*
Let N be an ITPN and $S = (M, h)$ its current state. Then a transition t is enabled at the marking M n times autoconcurrently if $\forall p \in P, \quad n \cdot v(p,t) \leqslant M(p)$. If $n = 1$ this is the usual definition of (single) enabling.
For each transition the value $E_i(M)$ tells how many times (at most) transition t_i can be fired autoconcurrently at marking M. Thus $E_i(M) = n_i$ if
$\forall p \in P, \quad (n_i \cdot v(p,t_i) \leqslant M(p)$ *and* $\exists p \in P, \quad (n_i + 1) \cdot v(p,t_i) > M(p))$. □

When transitions may fire, firing starts *immediately*; this is done by removing input tokens from the preplaces of the chosen transitions. A startfired transition t stays *active* for some time delay in between its associated time interval $\left[sfd(t), lfd(t)\right]$, until it may or must *endfire* by delivering the output tokens to its postplaces.
Three types of events are distinguished : *startfire, endfire and tick events*. The effect of each of these events on the state of an ITPN is given below.

Definition 5 *(**State change rules**)*
Let $\mathcal{N}$ be an ITPN and (M, h) its current state.

1. **Startfire events :** *A startfire event, denoted by $\lceil \mathtt{t_i}$, may occur immediately, even up to n times, if t_i is enabled at M, resp. if $E_i(M) = n$. For each occurrence of $\lceil \mathtt{t_i}$ the needed input tokens of t_i are removed from their preplaces, the clock associated with t_i will count this occurrence by incrementing the number $h_{i,1}$.*
   $(M,h) \xrightarrow{\lceil \mathtt{t_i}} (M',h')$ *with* $M' = M - \sum_{p \in {}^\bullet t_i} v(p,t_i)$ *and* $h'_{i,1} = h_{i,1} + 1$.
   There may be conflicts between enabled transitions, and the way they are solved is arbitrary. Tick events may not occur when there are still enabled transitions, i.e., if for some i, $E_i(M) > 0$.
2. **Endfire events :** *An endfire event, denoted by $\mathtt{t_i}\rangle$ must occur (even n times) if the clock associated with some t_i reach the upper bound of its associated interval i.e. $h_{i,j+1} \geq 1$ with $j = lfd(t_i)$. An endfire event $\mathtt{t_i}\rangle$ may occur if there is an active transition t_i with age in $[sfd(t_i), lfd(t_i)[$. The corresponding $h_{i,j+1}$ is then decremented.*
   $(M,h) \xrightarrow{\mathtt{t_i}\rangle} (M',h')$ *if* $\sum_{sfd(t_i) \leqslant j \leqslant lfd(t_i)} h_{i,j+1} > 1$ *with*
   $M' = M + \sum_{p \in t_i^\bullet} v(t_i,p)$ *and* $h'_{i,j+1} = h_{i,j+1} - 1$ *for some j with $h_{i,j+1} > 0$.*
   If this new state enables some transitions, one or more startfire events must then occur, and if endfire events(of zero duration) must occur in the sequel, they have to be handled, and so on.
3. **Tick events :** *A tick event, denoted by ✓, is enabled once neither a startfire event nor a must endfire event have to occur. The tick event increments the clocks for all active transitions and models the passing of time.*
   $(M,h) \xrightarrow{\checkmark} (M',h')$ *with $M' = M$ and for all i holds if*
   $E_i(M) = 0$ *and* $h_{i,lfd(t_i)+1} = 0$ *then* $h'_{i,j} = \begin{cases} h_{i,j-1} & \text{if } 1 < j \leqslant d \\ 0 & \text{if } j = 1 \end{cases}$. □

The ITPN $\mathcal{N}$ presented in Fig.1. is used as a running example.

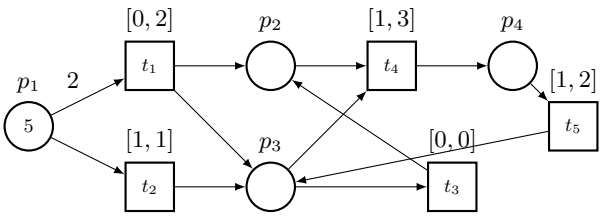


Fig. 1: ITPN $\mathcal{N}$

The condition of the occurrence of a tick event ensures that what have occurred since the previous tick event (i.e. in between two ticks) is maximal. In the case of transitions which may late zero time, which is allowed in the net class considered here, the notion of start firing event which need to occur "immediately", means "before the next tick". There is no time scale within zero time. The following definition will precise the notion of *global step* which happens in between two ticks and where multisets of startfire and endfire events will alternate until nothing more need to occur.

We only consider *wellformed* ITPNs in this article. They ensure to have always only a finite number of events which appear between two ticks. If there is no firing sequence of transitions of possible zero duration which increases a marking, the ITPN is wellformed. This property is decidable on the subnet restricted to transitions whose sfd is zero.

All results given here could be easily extended to ITPNs which are not necessarily wellformed: they allow infinite global steps where no tick event can follow. Thus such an infinite global step would be the "end" of a step firing sequence. We just like to limite the considerations here to the standard wellformed case.

2.2 Maximal Step Semantics for ITPNs

The executions of wellformed ITPNs with zero duration and autoconcurrency under maximal step semantics are given by the so called *firing step sequence* as defined in [**?**], where a *firing step sequence* is an alternating sequence of *globalsteps* and ticks.

A *globalstep* is a multiset of firing events of an ITPN in between two tick events, it consists on two principal multisets, the first one is called *Endstep* and the second one is called *Iteratedstep*. So a firing step is a triplet $(Endstep, Iteratedstep, tick)$, or a couple $(globalstep, tick)$.

An *Endstep* at state S contains all endfire events which must occur at S and an arbitrary multiset of endfire events which may occur at S. At the beginning, at initial state S_0, it is always empty as no transition has startfired. In the following steps, it can be empty; then the whole global step can be empty and one tick

event follows immediately the previous one. An *Iteratedstep* is an alternating sequence of multisets of startfire events (not necessarily maximal) and multisets of endfire events with zero duration (containing all must endfire ones). The alternation ends if neither a transition is enabled nor an endfire event with zero duration must occur, thus the iterated step is maximal and finite. This situation happens because of the wellformedness.

Thus a firing step sequence σ of length n is given by
$\sigma = S_0 \xrightarrow{globalstep_1} \tilde{S}_0 \xrightarrow{\checkmark} S_1 \xrightarrow{globalstep_2} \tilde{S}_1 \xrightarrow{\checkmark} S_2 \cdots S_{n-1} \xrightarrow{globalstep_n} \tilde{S}_{n-1} \xrightarrow{\checkmark} S_n$.
If in $\tilde{S}_{n-1} = (\tilde{M}_{n-1}, \tilde{h}_{n-1})$ no transition is active, i.e. if $\tilde{h}_{n-1}$ is the Zero-matrix, then we have a deadlock , and the last tick and S_n do not exist.

The following example illustrates firing steps.

Example 1 *Consider the ITPN $\mathcal{N}$, one possible initial firing step from its initial state S_0 is given in the sequel.*

The first Endstep is necessarily empty ($Endstep_1 = \emptyset$). Then, suppose that we fire two times t_1 and one time t_2, i.e. $\{[\mathtt{t_1}^2, [\mathtt{t_2}\}$ then we choose to endfire t_1 at zero duration, i.e. $\{\mathtt{t_1}\rangle\}$, after that we fire $\{[\mathtt{t_4}\}$, no further startfire event is possible now, so a tick event is executed. The first iterated step is the union of all these multisets :
$Iteratedstep_1 = \{[\mathtt{t_1}^2, [\mathtt{t_2}\} \uplus \{\mathtt{t_1}\rangle\} \uplus \{[\mathtt{t_4}\} = \{[\mathtt{t_1}^2, [\mathtt{t_2}, \mathtt{t_1}\rangle, [\mathtt{t_4}\}$.

Thus a possible first firing step of $\mathcal{N}$ is $\big(\{\}, \{[\mathtt{t_1^2}, [\mathtt{t_2}, \mathtt{t_1}\rangle, [\mathtt{t_4}\}, \checkmark\big)$. □

3 True concurrent semantics

In this paper we plan to study the behaviour of ITPNs without sequentializing the observation. Thus we will use partial order semantics to express true concurrency and in particular, nonsequential processes [**?**,**?**,**?**].

Processes have been defined and investigated for classical Petri nets and for some other net classes like **time** Petri Nets [**?**,**?**,**?**]. The only known contribution on process semantics of a **timed** Petri Net class is [**?**], but in a context of high level nets with one-safe markings. Arbitrary markings, zero durations of events and auto-concurrency give us new challenges. Also, no axiomatic approach exists until now for time or timed Petri nets.

3.1 Partial order structures

Concurrent runs or executions of an ITPN are usually represented by condition/event nets where all arcs have an arc weight 1.

$N' = (B, E, G)$ is a *condition/event net* if $B \cap E = \emptyset$ and $G \subseteq (B \times E) \cup (E \times B)$. The places of B are called *conditions* and the transitions of E are called *events*.

A *causal net* is a condition/event net $N' = (B, E, G)$ such that

- for every $b \in B, |\ {}^\bullet b| \leqslant 1$ and $|b^\bullet| \leqslant 1$,
- G^*, the transitive closure of G is acyclic,

– N' is finitly preceded, i.e., for every $x \in B \cup E$ the set $\{y \mid (y,x) \in G^*\}$ is finite.

Causal nets do not allow any branching at conditions. In a causal net $N' = (B, E, G)$, the transitive closure of the flow relation G is acyclic and therefore a partial order. We call it the *precedence relation* and denote it by $\prec$. The symbol $\preceq$ denotes the reflexive and transitive closure of G.

A *homomorphism* ϕ is a mapping that preserves the nature of nodes and the environment of events. A homomorphism is used to connect conditions and events of a *causal net* to places and transitions of the executed net whose behaviour is observed.

A *chain* C of a causal net is a set of totally ordered events, i.e.,
$\mathrm{C} \subset E$ and $\forall e \in \mathrm{C}\ \forall e' \in \mathrm{C}\ \big((e' \preceq e) \vee (e \preceq e')\big)$. It can be seen as a sequence of events that occurred during the run of the system.

A set AC of nodes of a causal net is an *antichain* if
$\forall x \in \mathrm{AC}\ \forall x' \in \mathrm{AC}\ \ (\neg(x \prec x') \wedge \neg(x' \prec x))$. An antichain AC is a *maximal antichain* or a *cut* if $\forall x \notin \mathrm{AC}$ the union $\mathrm{AC} \cup \{x\}$ is not an antichain.

Note that usually, cuts are considered restricted to conditions or restricted to events. In particular, a cut restricted to conditions is called *cut of conditions* or *B-cut*. Note that each *B-cut* of a process of a classical Petri Net represent a possible marking that may occur during the concurrent execution for some observer.

4 Process semantics for ITPNs

A **timed process** of an ITPN $\mathcal{N}$ will be defined as a pair (N', ϕ) where N' is a *causal net* and ϕ a homomorphism which labels the causal net with information from the ITPN $\mathcal{N}$. The set of *clock labels* is introduced to capture information about time elapsed since a transition is active. It is defined by

$$CL = \{(t, j) \mid t \in T \text{ and } j \leqslant \mathit{lfd}(t_i)\} \text{ and } P \cap CL = \emptyset.$$

A clock label (t, j) means that t is active and has age j.

The following set of *firing events* denoted by $\mathcal{FE}$ will label the events:

$$\mathcal{FE} = \{[\mathtt{t} \mid t \in T\} \cup \{\mathtt{t}\rangle \mid t \in T\} \cup \{\checkmark\} \text{ and } P \cap \mathcal{FE} = \emptyset.$$

Thus in a causal net where conditions are labeled in $P \cup CL$ a *B-cut* is able to represent a time-state (M,h). Note that for any set $B' \subseteq B$ the image $\phi(B')$ defines a *multi-set* of labels.

4.1 Inductive definition

Let $\mathcal{N} = (P, T, v, M_0, I)$ be an ITPN. A **timed process π of $\mathcal{N}$** is constructed along a possible firing step sequence σ of $\mathcal{N}$, whose length is n_o, as follows.

We construct successively labeled causal nets $\pi_i = (N_i', \phi_i) = (B_i, E_i, G_i, \phi_i)$ where $\phi_i : B_i \cup E_i \to (P \cup CL) \cup \mathcal{FE}$ by induction on i by using three **Add**-procedures given below for the creation of events. The $i-$th induction step corresponds to the $i-$th firing step of a σ. We stop if $\pi = (N', \phi) = \pi_{n_o}$.

The sets B_{CL} and B_P will be the sets of conditions whose postset is currently empty and which are labeled by clock labels and by places respectively.

Base of induction $\mathbf{i = 0}$: ${}^\bullet\pi_0 = B_0$ will be a set of conditions with $\phi_0 : B_0 \to P$ representing the initial marking such that
$\forall p \in P \;\; |{\phi_0}^{-1}(p) \cap B_0| = M_0(p)$. We set $E_0 = G_0 = \emptyset$, $B_P = B_0$ and $B_{CL} = \emptyset$.

Hypothesis: Let $\mathbf{n \geqslant 1}$. We suppose that $\forall i < n$, $\pi_i = (B_i, E_i, G_i, \phi_i)$ has been constructed and the current B_{CL} and B_P are known.

Induction step $\mathbf{i = n}$: We start by setting $B_i = B_{i-1}, E_i = E_{i-1}, G_i = G_{i-1}$ and $\phi_i = \phi_{i-1}$. Then π_i is constructed as follows:

a) (*Treatment of the first Endstep of the current globalstep*)
For each condition $b \in B_{CL}$:
- If $\phi_i(b) = (t, j)$ for some t with $j = \mathit{lfd}(t)$ then $\mathbf{Add}\big(b, \mathtt{t}\rangle, i\big)$.
- If $\phi_i(b) = (t, j)$ for some t with $\mathit{sfd}(t) \leqslant j < \mathit{lfd}(t)$ then $\mathbf{Add}\big(b, \mathtt{t}\rangle, i\big)$ or do nothing.

b) (*Treatment of the Iteratedstep of the current globalstep*)

b.1) (*Treatment of Startfirings*)
If there exists a set $B' \subseteq B_P$ with $\phi_i(B') = {}^\bullet t$ for some $t \in T$, then $\mathbf{Add}\big(B', [\mathtt{t}, i\big)$.
Repeat b.1) or goto to b.2) .

b.2) (*Treatment of an Endstep*)
For each condition $b \in B_{CL}$:
- If $\phi_i(b) = (t, 0)$ for some t with $\mathit{lfd}(t) = 0$, then $\mathbf{Add}\big(b, \mathtt{t}\rangle, i\big)$.
- If $\phi_i(b) = (t, 0)$ for some t with $\mathit{sfd}(t) = 0$ and $\mathit{lfd}(t) \neq 0$, then $\mathbf{Add}\big(b, \mathtt{t}\rangle, i\big)$ or do nothing.

(*Maximality of the globalstep*)
Repeat step b) until $\forall t \in T : {}^\bullet t \not\subseteq \phi(B_P))$ (i.e. until no startfire event is possible) , then go to c).

c) (*Treatment of a Tickevent*)
If $B_{CL} \neq \emptyset$ then $\mathbf{Add}\big(\checkmark, i\big)$. □

End (*If* $i = n_o$)

The three **Add**-procedures are as follows:

The startfire event creation $\mathbf{Add}\big(B', [\mathtt{t}, i\big)$:
- We add an event e with $\phi_i(e) = [\mathtt{t}$: $E_i = E_i \cup \{e\}$.
- We add arcs $G_i = G_i \cup \{(b, e) | b \in B'\}$.
- We add a condition b' with $\phi_i(b') = (t, 0)$: $B_i = B_i \cup \{b'\}; B_{CL} = B_{CL} \cup \{b'\}$.
- We add an arc $G_i = G_i \cup (e, b')$ and reset $B_P = B_P \setminus B'$.

The endfire event creation $\mathbf{Add}\big(b, \mathtt{t}\rangle, i\big)$:
- We add an event e with $\phi_i(e) = \mathtt{t}\rangle$: $E_i = E_i \cup \{e\}$.
- We add an arc $G_i = G_i \cup (b, e)$ and redefine $B_{CL} = B_{CL} \setminus \{b\}$.

– For each $p \in t^\bullet$, we add $v(t,p)$ conditions $B' = \{b'_1, .., b'_{v(t,p)}\}$ with $\phi_i(b') = p$ for all $b' \in B'$: $B_i = B_i \cup B'$ and $B_P = B_P \cup B'$.
– We add arcs $G_i = G_i \cup \{((e, b')|b' \in B'\}$.

The tick event creation $\mathbf{Add}(\checkmark, i)$:
– We add an event e with $\phi_i(e) = \checkmark$: $E_i = E_i \cup \{e\}$.
– We add arcs $G_i = G_i \cup \{(b, e)|b \in B_{CL}\}$.
– For each $b \in B_{CL}$, if $\phi_i(b) = (t, j)$ for some t and j, then we add a condition b' with $\phi_i(b') = (t, j + 1)$: $B_i = B_i \cup \{b'\}$ and an arc $G_i = G_i \cup (e, b')$.
– We redefine $B_{CL} = e^\bullet$.

We observe that c) happens at each step because of the wellformedness of the executed ITPN. If $B_{CL} = \emptyset$ then a deadlock appeared; otherwise a tick is added. A process construction stops after the creation of some tick event, except for deadlock. Thus global steps are fully included. It is easy to see, that for each i the cut $\pi_i^\bullet$ is a *B-cut* and represents the time-state S_{i+1} reached after the execution of the considered firing step sequence of length i, respectively $\tilde{S}_{n_0-1}$ in the case of a deadlock after n_0 global steps.

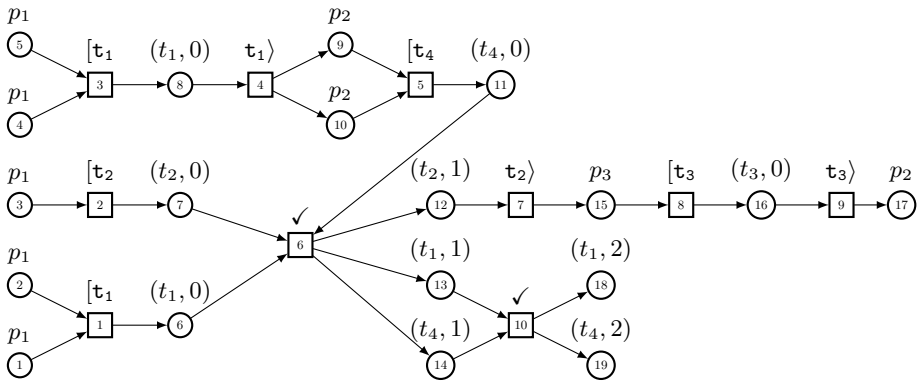


Fig. 2: An arbitrary process of the ITPN $\mathcal{N}$ of Fig.1.
We use abreviation as follows: (i) Denotes condition b_i and [j] denotes event e_j.

4.2 Axiomatic definition and how to overcome its limits

We start by proposing an axiomatization by local properties of events as processes of a classical Petri Nets have been axiomatized, for instance in [**?**].
The causal net $\pi = (N', \phi)$ is an **timed evolution of** $\mathcal{N}$ if

$\phi : B \cup E \to (P \cup CL) \cup \mathcal{FE}$ is a homomorphism verifying
– $\forall b \in B,\ \phi(b) \in P \cup CL$ and $\forall e \in E,\ \phi(e) \in \mathcal{FE}$ (coherence of labeling).
– ${}^\bullet\pi \subseteq B$ and $\forall p \in P,\ |\phi^{-1}(p) \cap {}^\bullet\pi| = M_0(p)$ (the initial marking).
– For each event e of the causal net N' it holds :
  • **Case 1 :** If $\phi(e) = [\mathtt{t}$ for some $t \in T$ then
    ∗ $\forall p \in P \quad |\phi^{-1}(p) \cap {}^\bullet e| = v(p,t)$ and $|e^\bullet| = 1$ with $\phi(e^\bullet) = \{(t, 0)\}$

- **Case 2:** If $\phi(e) = \mathtt{t}\rangle$ for some $t \in T$ then
  - $| {}^\bullet e| = 1$ and $\phi({}^\bullet e) = \{(t, j)\}$ for some $j \in [sfd(t), lfd(t)]$ and
  - $\forall b \in e^\bullet \ \phi(b) \in P$ and $\forall p \in P \ \ |\phi^{-1}(p) \cap e^\bullet| = v(t, p)$.
- **Case 3:** If $\phi(e) = \checkmark$ then
  - $\forall b \in {}^\bullet e \cup e^\bullet \quad \phi(b) \in CL$ and
  - $\forall b \in {}^\bullet e \quad \phi(b) = (t, j))$ for some t and some $j < lfd(t)$ and
  - $\forall t \in T \quad \forall j \in [0, d] \ |\phi^{-1}((t, j)) \cap {}^\bullet e| = |\phi^{-1}((t, j + 1)) \cap e^\bullet|$

These axioms define especially local properties of events in the same way as axioms of processes for classical Petri Nets, i.e., they ensure that each event has a correct pre- and postset of conditions with respect to the firing rule. Only the initial cut and the final one are evoked. It is evident that each event of an inductively defined process satisfies clearly the corresponding axiom.

First let us state the following sentence.

Proposition 1 *There are evolutions which are not processes.*

Proof. An evolution of the net N of Fig.1 is given in Fig.3. It respects all points of the axiomatic definition but does not correspond to any firing step sequence of N. We can see in this example that tick event e_2 is not global as it should be, and may only occur when neither a startfire event nor a must endfire event is possible. In particular, e_3 and e_4 are independent from e_2, thus conditions b_8 and b_9 are also independent from e_2 instead of entering it. These axioms also allow infinite evolutions, and we could have added an axiom like $[\pi^\bullet \neq \emptyset$ and $\forall b \in B \ \ \exists x \in \pi^\bullet \ \ b \leq x]$ to ensure that π is finite. But as finiteness will be a consequence of the axioms adjoined in the sequel, we omit it here.

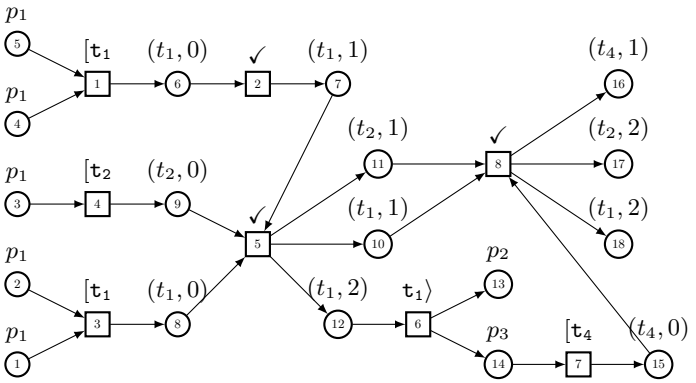


Fig. 3: An evolution which is not a process of running example.

Therefore, with the given axiomatic definition we are unable to avoid some partial orders that violate important properties like the fact that tick events have to be global and have to form a chain. Let us try to formulate supplementary axioms about non local properties for tick events:

Globality axioms:

- (a) $\forall e \ \forall e' \ \ \big((\phi(e) = \checkmark \wedge \phi(e') = \checkmark) \Rightarrow \quad (e = e' \vee e' \prec e \vee e \prec e')\big)$
- (b) $\forall e \ \forall e' \ \ \big((\phi(e) = \checkmark \wedge \phi(e') = \checkmark \wedge e' \prec e)) \Rightarrow (\forall b \in {}^\bullet e \ \ e' \prec b)\big)$

The axiom (a) ensures that all tick events form a chain in the partial order, and (b) that all conditions entering later tick events are necessarily greater than other preceding tick events, thus in particular greater than its potential direct predecessor tick. In particular, for the running example, point (b) makes impossible to creat a "partial" tick event like e_2 in Fig.3, as b_9 and b_8 entering e_5 are not comparable to (and not greater than) e_2.

Finally global axioms about maximality and concerning the final cut have to be defined.

Final cut axioms:

- (d) $\forall t \in T \;\; \exists p \in {}^\bullet t \;\; |\phi^{-1}(p) \cap \pi^\bullet| < v(p,t)$
- (e) $\forall t \in T \;\; \forall b \;\; (\phi(b) = (t, lfd(t)) \Rightarrow |b^\bullet| = 1)$
- (f) $\forall x \in \pi^\bullet \; \phi(x) \in P \;\; \vee \;\; \exists e \; \big(\; (\phi(e) = \checkmark \wedge \forall x \; (e \prec x \Rightarrow \phi(x) \in CL) \wedge \forall b \notin e^\bullet \; (\phi(b) \in CL) \Rightarrow b^\bullet \neq \emptyset) \; \big)$

As by (d) no startfire event is possible at the final cut $\pi^\bullet$, availible tokens (P labeled conditions) are maximally used for startfire events and thus the obligation of startfiring, up to some choice, is satisfied.

By axiom (e) must endfire events must occur.

Axiom (f) ensures that either (case 1) all elements in the final cut are place labeled, which together with (d) means that the process ends by an deadlock; or (case 2) there is a last tick event whose postset are clock labeled conditons in the final cut $\pi^\bullet$ and all other clock labeled conditons have a successor; i.e., they enter in an endfiring event, or they enter in a tick event which is by axiom (b) the appropriate one and not a later one.

We may conclude that what happens between two tick events is a global step and axioms (d) and (e) together ensure its maximality. Axiom (f) also implies the finiteness of the process. Finally the finite cut correspond to the time state reached after the execution of all events of the evolution.

The initial cut ${}^\bullet\pi$ and the final cut $\pi^\bullet$ are the only sets of nodes evoked in the given axioms; they are just used like constant sets. Thus we have successfully avoided to use second order quantification over sets - representing intermediate *B-cut*s - in all proposed axioms.

As consequence of these observations we obtain the desired result.

Proposition 2 *Let $\mathcal{N}$ be a wellformed ITPN. Then the class of timed evolutions of $\mathcal{N}$ which also satisfy the axioms (a) to (f) is the same as that of timed processes of $\mathcal{N}$ defined inductively.*

5 Conclusion

In this paper we investigate ITPNs in their most general setting, i.e., with autoconcurrency and with zero duration. In a previous paper their usual maximal step semantics were introduced [**?**], in terms of firing step sequences.

The goal of the present article is to present their truly concurrent behaviour. Thus first, timed processes of ITPNs have been defined inductively along firing step sequences. Then the possibility of defining these processes in an axiomatic

way too are studied. Our first attempt was to propose local axioms, similar to the way processes of classical Petri Nets are defined axiomatically, obtaining the so called timed evolutions. Then we stated and illustrated the fact that some timed evolutions do not correspond to any firing step sequence and therefore they cannot be timed processes.

Several supplementary "global" axioms are gradually formulated and discussed. They are a novelty when defining processes, but the price to pay to capture global timing and firing constraints.

We succeed to give a full axiomatization of timed processes totally compatible with the firing step semantics.

6 Ackknowlegment

My thanks go to Raymond Devillers for his attentive reading of divers versions and a lot of pertinent remarks and suggestions.

References

1. T. Aura and J. Lilius. Time processes for time petri-nets. In *Proc. of 18th International Conference ICATPN '97, LNCS*, volume 1248, pages 136–155. Springer, 1997.
2. E. Best and C. Fernández. *Nonsequential Processes - A Petri Net View*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.
3. C. Bui Thanh, H. Klaudel, and F. Pommereau. Petri nets with causal time for system verification. *Electr. Notes Theor. Comput. Sci.*, 68(5):85–100, 2002.
4. E.Best and R.Devillers. Sequential and concurrent behaviour in petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.
5. H. Fleischhack and E. Pelz. Hierarchical Timed High Level Nets and their Branching Processes. In *Proc. of 26th International Conference ICATPN'03, LNCS* , volume 2679, pages 397–416. Springer, 2003.
6. U. Goltz and W. Reisig. The non-sequential behavior of petri nets. *Information and Control*, 57(2/3):125–147, 1983.
7. P. Merlin. *A Study of the Recoverability of Communication Protocols*. PhD thesis, Irvine, 1974.
8. E. Pelz, A. Kabouche, and L. Popova-Zeugmanm. Interval-timed petri nets with auto-concurrent semantics and their state equation. In *Proc. of International Workshop on Petri Nets and Software Engineering (PNSE'15), CEUR Workshop, http://ceur-ws.org/Vol-1372*, pages 245–265, 2015.
9. C. A. Petri. Fundamentals of a Theory of Asynchronous Information Flow. In *IFIP Congress*, 1962.
10. C. Ramchandani. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. *Project MAC-TR 120, MIT*, February 1974.
11. P. H. Starke. Processes in petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 17(8/9):389–416, 1981.
12. V. Valero Ruiz, D. de Frutos-Escrig, and F. Cuartero. Timed processes of timed petri nets. In *Proc. of 16th International Conference, ICATPN '95, LNCS*, volume 616, pages 490–509. Springer, 1995.
13. J. Winkowski. Algebras of processes of timed petri nets. In *Proc. of 5th International Conference CONCUR '94, LNCS*, volume 836, pages 194–209. Springer, 1994.

On Generation of Time-based Label Refinements

Niek Tax[1,2], Emin Alasgarov[1,2], Natalia Sidorova[1], and Reinder Haakma[2]

[1] Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600MB Eindhoven, The Netherlands
{n.tax,n.sidorova}@tue.nl,{e.alasgarov}@student.tue.nl
[2] Philips Research, Prof. Holstlaan 4, 5665 AA Eindhoven, The Netherlands
{niek.tax,reinder.haakma}@philips.com

Abstract. Process mining is a research field focused on the analysis of event data with the aim of extracting insights in processes. Applying process mining techniques on data from smart home environments has the potential to provide valuable insights in (un)healthy habits and to contribute to ambient assisted living solutions. Finding the right event labels to enable application of process mining techniques is however far from trivial, as simply using the triggering sensor as the label for sensor events results in uninformative models that allow for too much behavior (overgeneralizing). Refinements of sensor level event labels suggested by domain experts have shown to enable discovery of more precise and insightful process models. However, there exist no automated approach to generate refinements of event labels in the context of process mining. In this paper we propose a framework for automated generation of label refinements based on the time attribute of events. We show on a case study with real life smart home event data that behaviorally more specific, and therefore more insightful, process models can be found by using automatically generated refined labels in process discovery.

Keywords: Label Refinements, Process Discovery, Unsupervised Learning

1 Introduction

Process mining is a fast growing discipline that combines knowledge and techniques from data mining, process modeling, and process model analysis [22]. Process mining techniques concern the analysis of events that are logged during process execution, where event records contain information on what was done, by whom, for whom, where, when, etc. Events are grouped into cases (process instances), e.g. per patient for a hospital log, or per insurance claim for an insurance company. *Process discovery* plays an important role in process mining, focusing on extracting interpretable models of processes from event logs. One of the attributes of the events is usually used as its label and its values become transition/activity labels in the process models generated by process discovery algorithms.

The scope of process mining have broadened in recent years from business process management to other application domains, one of them being analysis of

Table 1. An example of an event log from a smart home environment.

Id	Timestamp	Address	Sensor	Sensor value
1	03/11/2015 04:59:54	Mountain Rd. 7	Motion sensor - Bedroom	1
2	03/11/2015 06:04:36	Mountain Rd. 7	Motion sensor - Bedroom	1
3	03/11/2015 08:45:12	Mountain Rd. 7	Motion sensor - Living room	1
4	03/11/2015 09:10:10	Mountain Rd. 7	Motion sensor - Kitchen	1
5	03/11/2015 09:12:01	Mountain Rd. 7	Power sensor - Water cooker	1200
6	03/11/2015 09:15:45	Mountain Rd. 7	Power sensor - Water cooker	0
...	03/11/2015 ...	Mountain Rd. 7	...	...
7	03/12/2015 01:01:23	Mountain Rd. 7	Motion sensor - Bedroom	1
8	03/12/2015 03:13:14	Mountain Rd. 7	Motion sensor - Bedroom	1
9	03/12/2015 07:24:57	Mountain Rd. 7	Motion sensor - Bedroom	1
10	03/12/2015 08:34:02	Mountain Rd. 7	Motion sensor - Bedroom	1
11	03/12/2015 09:12:00	Mountain Rd. 7	Motion sensor - Living room	1
...	03/12/2015 ...	Mountain Rd. 7	...	...
12	03/14/2015 03:41:46	Mountain Rd. 7	Motion sensor - Bedroom	1
13	03/14/2015 05:00:17	Mountain Rd. 7	Motion sensor - Bedroom	1
14	03/14/2015 08:52:32	Mountain Rd. 7	Motion sensor - Bedroom	1
15	03/14/2015 09:30:54	Mountain Rd. 7	Motion sensor - Living room	1
16	03/14/2015 09:35:25	Mountain Rd. 7	Power sensor - TV	160
17	03/14/2015 10:27:37	Mountain Rd. 7	Power sensor - TV	0
...	03/14/2015 ...	Mountain Rd. 7	...	...
...	...	...	...	...

events of human behavior with data originating from sensors in smart home environments [19, 21, 20]. Table 1 shows an example of such an event log. Events in the event log are generated by e.g. motion sensors placed in the home, power sensors placed on appliances, open/close sensors placed on closets and cabinets, etc. Particularly challenging in applying process mining in this application domain is the extraction of meaningful event labels that allow for discovery of insightful process models. Simply using the sensor that generates an event (the *sensor* column in Table 1) as event label is shown to produce non-informative process models that overgeneralize the event log and allow for too much behavior [21]. Abstracting sensor-level events into events at the level of human activity (e.g. *eating*, *sleeping, etc.*) using techniques closely related to techniques used in the activity recognition field helps to discover more behaviorally more constrained and insightful process models [20], but applicability of this approach relies on the availability of a reliable diary of human behavior at the activity level, which is often just impossible to obtain.

In our earlier work [21] we showed that better process models can be discovered by taking the name of the sensor that generated the event as a starting point for the event label and then refining these labels using information on the time within the day at which the event occurred. The refinements used in [21] were based on domain knowledge, and not identified automatically from the data. In this paper, we aim at automatic generation of semantically interpretable label refinements that can be explained to the user, by basing label refinements on data attributes of events. We explore methods to bring parts of the timestamp information to the event label in an intelligent and fully automated way, with the end goal of discovering behaviorally more precise and therefore more insightful process models.

We start by introducing basic concepts and notations used in this paper in Section 2. In Section 3, we introduce a framework for the generation of event labels refinements based on the time attribute. In Section 4, we apply this framework on a real life smart home data set and show the effect of the refined event labels on process discovery. We continue by describing related work in Section 5 and conclude in Section 6.

2 Preliminaries

In this section we introduce the notions related to event logs and relabeling functions for traces and then define the notions of refinements and abstractions. We also introduce the Petri net process model notation.

We use the usual sequence definition, and denote a sequence by listing its elements, e.g. we write $\langle a_1, a_2, \ldots, a_n \rangle$ for a (finite) sequence $s : \{1, \ldots, n\} \to A$ of elements from some alphabet A, where $s(i) = a_i$ for any $i \in \{1, \ldots, n\}$; $|s|$ denotes the length of sequence s; $s_1 s_2$ denotes the concatenation of sequences s_1 and s_2. A *language* $\mathfrak{L}$ over an alphabet A is a set of sequences over A. $\mathfrak{L}^p$ is the prefix closure of a language $\mathfrak{L}$ (with $\mathfrak{L} \subseteq \mathfrak{L}^p$).

An event is the most elementary element of an event log. Let $\mathcal{I}$ be a set of event identifiers, and $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ be an attribute domain consisting of n attributes (e.g. timestamp, resource, activity name, cost, etc.). An event is a tuple $e = (i, a_1, \ldots, a_n)$, with $i \in \mathcal{I}$ and $(a_1, \ldots, a_n) \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$. The *event label* of an event is the attribute set $(a_1 \ldots, a_n)$; e_i, and e_a respectively denote the identifier and label of event e. The timestamp attribute of an event is denoted by a_t. $\mathcal{E} = \mathcal{I} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is a universe of events over $\mathcal{A}_1, \ldots, \mathcal{A}_n$. The rows of Table 1 are events from an event universe over the event attributes *timestamp*, *sensor*, *address*, and *sensor value*.

Events are often considered in the context of other events. We call $E \subseteq \mathcal{E}$ an *event set* if E does not contain any events with the same event identifier. The events in Table 1 together form an event set. A *trace* σ is a finite sequence formed by the events from an event set $E \subseteq \mathcal{E}$ that respects the time ordering of events, i.e. for all $k, m \in \mathbb{N}$, $1 \leq k < m \leq |E|$, we have: $\sigma(k)_t \leq \sigma(m)_t$. We define the *universe of traces* over event universe $\mathcal{E}$, denoted $\Sigma(\mathcal{E})$, as the set of all possible traces over $\mathcal{E}$. We omit $\mathcal{E}$ in $\Sigma(\mathcal{E})$ and use the shorter notation Σ when the event universe is clear from the context.

Often it is useful to partition an event set into smaller sets in which events belong together according to some criterion. We might for example be interested in discovering the typical behavior within a household over the course of a day. In order to do so, we can e.g. group together events with the same *address* and the same day-part of the *timestamp*, as indicated by the horizontal lines in Table 1. For each of these event sets, we can construct a trace; time stamps define the ordering of events within the trace. For events of a trace having the same time stamps, an arbitrary ordering can be chosen within a trace.

An *event partitioning function* is a function $ep : \mathcal{E} \to T_{id}$ that defines the partitioning of an arbitrary set of events $E \subseteq \mathcal{E}$ from a given event universe $\mathcal{E}$ into event sets $E_1, \ldots, E_j, \ldots$ where each E_j is the maximal subset of E such that for any $e_1, e_2 \in E_j$, $ep(e_1) = ep(e_2)$; the value of ep shared by all the elements of E_j defines the value of the *trace attribute* T_{id}. Note that multidimensional trace attributes are also possible, i.e. a combination of the name of the person performing the event activity and the date of the event, so that every trace contains activities of one person during one day. The event sets obtained by applying an event partitioning can be transformed into traces (respecting the time ordering of events).

An event log L is a finite set of traces $L \subseteq \Sigma(\mathcal{E})$. $A_L \subseteq \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ denotes the *alphabet of event labels* that occur in log L. The traces of a log are often transformed before doing further analysis: very detailed but not necessarily informative event descriptions are transformed into some *informative* and *repeatable* labels. For the labels of the log in Table 1, the sensor values could be abstracted to *on*, and *off* or labels can be redefined to a subset of the event attributes, e.g. leaving the sensor values out completely. Next to that, if the event partitioning function maps each event from Table 1 to its address and the day-part of the timestamp, these attributes (indicated in gray) become the trace attribute and can safely be removed from individual events.

After this relabeling step, some traces of the log can become identically labeled (the event id's would still be different). The information about the number of occurrences of a sequence of labels in an event log is highly relevant for process mining, since it allows differentiating between the mainstream behavior of a process (frequently occurring behavioral patterns) and exceptional behavior.

Let $\mathcal{E}, \mathcal{E}'$ be an event universe. A function $l : \mathcal{E} \to \mathcal{E}'$ is an *event relabeling function*. A relabeling function can be used to obtain more useful event labels than the full set of event attribute values. We lift l to event logs. Let $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ be event universes with $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ being pairwise different. Let $l_1 : \mathcal{E} \to \mathcal{E}_1$ and $l_2 : \mathcal{E} \to \mathcal{E}_2$ be event relabeling functions. Relabeling function l_1 is a *refinement* of relabeling function l_2, denoted by $l_1 \preceq l_2$, iff $\forall_{e_1,e_2 \in \mathcal{E}} : l_1(e_1) = l_1(e_2) \implies l_2(e_1) = l_2(e_2)$; l_2 is then called an *abstraction* of l_1.

The goal of process discovery is to discover a process model that represents the behavior seen in an event log. A frequently used process modeling notation in the process mining field is the Petri net [16]. Petri nets are directed bipartite graphs consisting of transitions and places, connected by arcs. Transitions represent activities, while places represent the enabling conditions of transitions. Labels are assigned to transitions to indicate the type of activity that they model. A special label τ is used to represent invisible transitions, which are only used for routing purposes and not recorded in the execution log.

A *labeled Petri net* $N = \langle P, T, F, A_M, \ell \rangle$ is a tuple where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation, A_M is an alphabet of labels representing activities, with $\tau \notin A_M$ being a label representing invisible events, and $\ell : T \to A_M \cup \{\tau\}$ is a labeling function that assigns a label to each transition. For a node $n \in (P \cup T)$ we use $\bullet n$ and $n\bullet$ to denote the set of input and output nodes of n, defined as $\bullet n = \{n | (n', n) \in F\}$ and $n\bullet = \{n | (n, n') \in F\}$. An example of a Petri net can be seen in Figure 1, where circles represent places and squares represent transitions. Gray transitions with smaller width represent τ transitions.

A state of a Petri net is defined by its *marking* $M \in \mathbb{N}^P$ being a multiset of places. A marking is graphically denoted by putting $M(p)$ tokens on each place $p \in P$. A pair (N, M) is called a marked Petri net. State changes occur through transition firings. A transition t is enabled (can fire) in a given marking M if each input place $p \in \bullet t$ contains at least one token. Once a transition fires, one token is removed from each input place of t and one token is added to each output place of t, leading to a new marking. An *accepting Petri net* is a

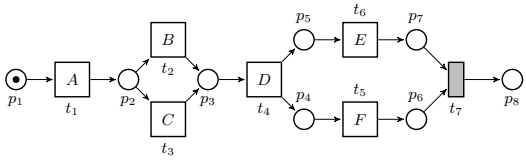


Fig. 1. An example Petri net.

3-tuple (N, M_i, M_f) with N a labeled Petri net, M_i an initial marking, and M_f a set of final markings. Many process modeling notations, including accepting Petri nets, have formal executional semantics and a model defines a *language of accepting traces* $\mathfrak{L}$. For the Petri net in Figure 1, the language of accepting traces is $\{\langle A, B, D, E, F\rangle, \langle A, B, D, F, E\rangle, \langle A, C, D, E, F\rangle, \langle A, C, D, F, E\rangle\}$.

3 A Framework for Time-based Label Refinements

To generate potential label refinements for every label based on time we take a clustering based approach by identifying dense areas in time space for each label. The time part of the timestamps consists of values between *00:00:00* and *23:59:59*, equivalent to the timestamp attribute from Table 1 with the day-part of the timestamp removed. This timestamp can be transformed into a real number hourfloat representation in interval $[0, 24)$. We chose to apply soft clustering (also referred to as fuzzy clustering), which has the benefit of assigning to each data point a likelihood of belonging to each cluster. A well-known approach to soft clustering is based on the combination of the Expectation-Maximization (EM) algorithm with mixture models, which are probability distributions consisting of multiple components of the same probability distribution. Each component in the mixture represents one cluster and the probability of a data point belonging to that cluster is the probability that this cluster generated that data point. The EM algorithm is used to obtain a maximum likelihood estimate of the mixture model parameters, i.e. the parameters of the probability distributions in the mixture.

A well-known example of a mixture model is the Gaussian Mixture Model (GMM), where the components in the mixture distributions are normal distributions. The data space of time is, however, non-euclidean: it has a circular nature, e.g. 23.99 is closer to 0 than to 23. This circular nature of the data space introduces problems for GMMs, as shown in Figure 2. The GMM fitted to the timestamps of the sensor events consists of two components, one with the mean at 9.05 and one with a mean at 20. The histogram representation of the same data shows that some events happened just after midnight, which is actually closer on the clock to 20 than to 9.05. The GMM however is unaware of the circularity of the clock, which results in the mixture model that seems inappropriate when visually comparing with the histogram. The field of circular statistics (also referred to as directional statistics), concerns analysis of such circular data spaces (cf. [14]).

Here, we introduce a framework for generating refinements of event labels based on time attributes using techniques from the field of circular statistics. This framework consists of three stages:

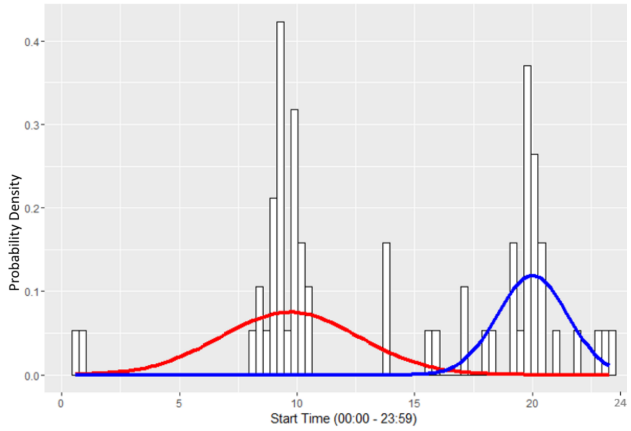


Fig. 2. The histogram representation and a Gaussian Mixture Model fitted to timestamps values of the plates cupboard sensor in the van Kasteren data set [23].

Data-model pre-fitting stage A known problem with many clustering techniques is that they return clusters even when the data should not be clustered. In this stage we assess how many clusters the events of a sensor type contain.

Data-model fitting stage In this stage we cluster the events of a sensor type by timestamp using a mixture consisting of components that take into account the circularity of the data.

Data-model post-fitting stage In this stage the quality of the label refinements is assessed from both a cluster quality perspective and a process model (event ordering statistics) perspective.

3.1 Data-model pre-fitting stage

We now describe a test for uniformity, a test for unimodality, and a method to select the number of clusters in the data.

Uniformity Check - Rao's Spacing Test Rao's spacing test [15] tests the uniformity of the timestamps of the events from a sensor around the circular clock. This test is based on the idea that uniform circular data is distributed evenly around the circle, and n observations are separated from each other $\frac{360}{n}$ degrees. The null hypothesis is that the data is uniform around the circle.

Given n successive observations $f_1, \ldots, f_n$, either clockwise or counterclockwise, the test statistics U for Rao's Spacing Test is defined as $U = \frac{1}{2} \sum_{i=1}^{n} \mid T_i - \lambda \mid$, where $\lambda = \frac{360^\circ}{n}$, $T_i = f_{i+1} - f_i$ for $1 \leq i \leq n-1$ and $T_n = (360^\circ - f_n) + f_1$.

Unimodality Check - Hartigan's Dip Test Hartigan's dip tests [7] the null hypothesis that the data follows a unimodal distribution on a circle. When the null hypothesis can be rejected, we know that the distribution of the data is at

least bimodal. Hartigan's dip test measures the maximum difference between the the empirical distribution function and the unimodal distribution function that minimizes that maximum difference.

Number of Component Selection - Bayesian Information Criterion The Bayesian Information Criterion (BIC) [17] introduces a penalty for the number of model parameters to the evaluation of a mixture model. Adding a component to a mixture model increases the number of parameters of the mixture with the number of parameters of the distribution of the added component. The likelihood of the data given the model can only increase by adding extra components, adding the BIC penalty results in a trade-off between number of components and the likelihood of the data given the mixture model. BIC is formally defined as $BIC = -2 * ln\hat{L} + k * ln(n)$, where $\hat{L}$ is a maximized value for the data likelihood, n is the sample size, and k is the number of parameters to be estimated. A lower BIC value indicates a better model. We start with 1 component, and iteratively increase from k to $k+1$ components as long as the decrease in BIC is larger than 10, which is the threshold for decisive evidence of high BIC [10].

3.2 Data-model fitting stage

We cluster events generated by one sensor using a mixture model consisting of components of the von Mises distribution, which is a circular version of the normal distribution. This technique is based on the approach of Banerjee et al. [1], who introduce a clustering method based on a mixture of von Mises-Fisher distribution components, which is a generalization of the 2-dimensional von Mises distribution to n-dimensional spheres. A probability density function for a von Mises distribution with mean direction μ and concentration parameter κ is defined as $pdf(\theta \mid \mu, \kappa) = \frac{1}{2\pi I_0(\kappa)} e^{\kappa \cos(\theta - \mu)}$, where mean μ and data point θ are expressed in radians on the circle, such that $0 \leq \theta \leq 2\pi,\ 0 \leq \mu \leq 2\pi,\ \kappa \geq 0$. I_0 represents the modified Bessel function of order 0, defined as $I_0(k) = \frac{1}{2\pi} \int_0^{2\pi} e^{\kappa \cos(\theta)} d\theta$. As κ approaches 0, the distribution becomes uniform around the circle. As κ increases, the distribution becomes relatively concentrated around the mean μ and the von Mises distribution starts to approximate a normal distribution. We fit a mixture model of von Mises components using the package movMF [9] provided in R.

3.3 Data-model post-fitting stage

After fitting a mixture of von Mises distributions to the sensor events, we perform a goodness-of-fit test to check whether the data could have been generated from this distribution. We describe the Watson U^2 statistic [25], a goodness-of-fit assessment based on hypothesis testing. The Watson U^2 statistic measures the discrepancy between the cumulative distribution function $F(\theta)$ and the empirical distribution function $F_n(\theta)$ of some sample θ drawn from some population and is defined as $U^2 = n \int_0^{2\pi} \left[F_n(\theta) - F(\theta) - \int_0^{2\pi} \left\{ F_n(\phi) - F(\phi) \right\} dF(\phi) \right]^2 dF(\theta)$.

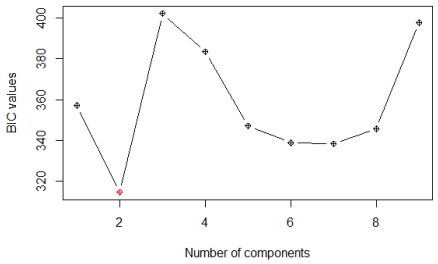


Fig. 3. BIC values for different numbers of components in the mixture model.

Table 2. Estimated parameters for a mixture of von Mises components for bedroom door sensor events.

Cluster	α	μ (radii)	κ
Cluster 1	0.76	2.05	3.85
Cluster 2	0.24	5.94	1.56

Furthermore we assess the quality of refining the event label into a new label for each cluster from a process perspective using the label refinement evaluation method described in [21]. This method tests whether the log statistics that are used in many process discovery algorithms become significantly more deterministic by applying the label refinement.

4 Case Study

We show the results of our time-based label refinements approach on the real life smart home data set described in van Kasteren et al. [23]. The van Kasteren data set consists of 1285 events divided over fourteen different sensors. We segment in days from midnight to midnight to define cases. Figure 4a shows the process model discovered on this event log with the Inductive Miner infrequent [11] with 20% filtering, which discovers a process model that describes the most frequent 80% of behavior in the log. Note that this process model overgeneralises allowing too much behaviour. At the beginning a (possibly repeated) choice is made between five transitions. At the end of the process, the model allows any sequence over the alphabet of five activities, where each activity occurs at least once.

We illustrate our proof of concept by applying the framework to the *bedroom door* sensor. Rao's spacing test results in a test statistic of 241.0 with 152.5 being the critical value for significance level 0.01, indicating that we can reject the null hypothesis of a uniformly distributed set of *bedroom door* timestamps. Hartigan's dip test results in a p-value of 3.95×10^{-4}, indicating that we can reject the null hypothesis that there is only one cluster in the *bedroom door* data. Figure 3 shows the BIC values for different numbers of components in the model. The figure indicates that there are two clusters in the data, as this corresponds to the lowest BIC value. Table 2 shows the mean and κ parameters of the two clusters found by optimizing the von Mises mixture model with the EM algorithm. A value of $0 = 2\pi$ radii equals midnight. After applying the von Mises mixture model to the *bedroom door* events and assigning each event to the maximum likelihood cluster we obtain a time range of [3.08-10.44] for cluster 1 and a time range of [17.06-0.88] for cluster

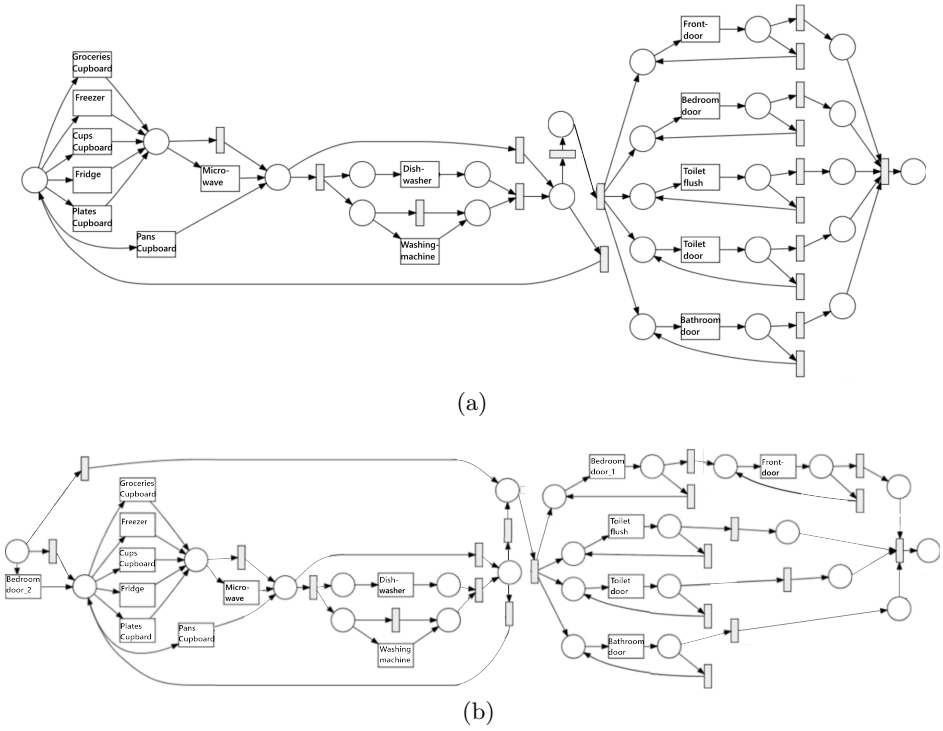


Fig. 4. Process models discovered on the van Kasteren data with sensor-level labels (a) and refined labels (b) with the Inductive Miner infrequent (20% filtering) [11].

2. The Watson U^2 test results in a test statistic of 0.368 and 0.392 for cluster 1 and 2 respectively with a critical value of 0.141 for a 0.01 significance level, indicating that the data is likely to be generated by the two von Mises distributions found. The label refinement evaluation method [21] finds statistically significant differences between the events from the two *bedroom door* clusters with regard to their control-flow relations with other activities in the log for 10 other activities using the significance level of 0.01, indicating that the two clusters are different from a control-flow perspective. Figure 4b shows the process model discovered with the Inductive Miner infrequent with 20% filtering after applying this label refinement to the van Kasteren event log. The process model still overgeneralizes in general, but the label refinement does help restricting the behavior, as it shows that the evening *bedroom door* events are succeeded by one or more events of type *groceries cupboard*, *freezer*, *cups cupboard*, *fridge*, *plates cupboard*, or *pans cupboard*, while the morning *bedroom door* events are followed by one or more *frontdoor* events. It seems that this person generally goes to the bedroom in-between coming home from work and starting to cook. The loop of the *frontdoor* events could be caused by the person leaving the house in the morning for work, resulting in no logged events until the person comes home again by opening the *frontdoor*. Note that in Figure 4a *bedroom door* and *frontdoor* events can occur an arbitrary number of

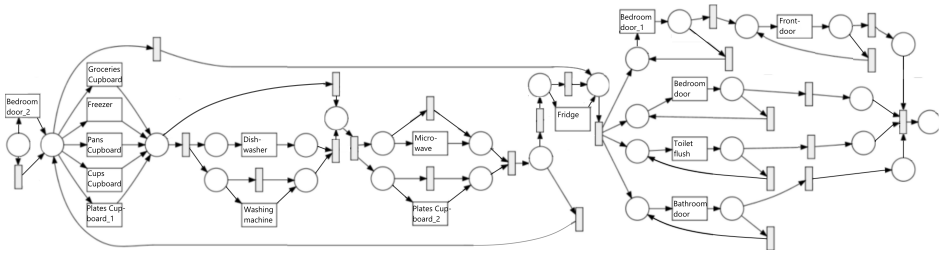


Fig. 5. Inductive Miner infrequent (20% filtering) [11] result after a second label refinement.

times in any order. Figure 4a furthermore does not allow for the *bedroom door* to occur before the whole block of kitchen-located events at the beginning of the net.

Label refinements can be applied iteratively. Figure 5 shows the effect of a second label refinement step, where *Plates cupboard* using the same methodology is refined into two labels, representing time ranges [7.98-14.02] and [16.05-0.92] respectively. This refinement shows the additional insight that the evening version of the *Plates cupboard* occurs in directly before or after the microwave.

5 Related Work

Refining event labels in the event log is closely related to the task of mining process models with duplicate activities, in which the resulting process model can contain multiple transitions/nodes with the same label. From the point of view of the behavior allowed by a process model, it makes no difference whether a process model is discovered on an event log with refined labels, or whether a process model is discovered with duplicate activities such that each transition/node of the duplicate activity precisely covers one versions of the refined label. The first process discovery algorithm capable of discovering duplicate tasks was proposed by Herbst and Karagiannis in 2004 [8], after which many others have been proposed, including the Genetic Miner [4], the Evolutionary Tree Miner [2], the α^*-algorithm [12], the $\alpha^\#$-algorithm [6], the EnhancedWFMiner [5], and a simulated annealing based algorithm [18]. An alternative approach has been proposed by Vázques-Barreiros [24] et al., who describe a local search based approach to repair a process model to include duplicate activities, starting from an event log and a process model without duplicate activities. Existing work on mining models with duplicate activities all base their duplicate activities on how well the event log fits the process model, and do not try to find any semantic difference between the multiple versions of the activities in the form of data attribute differences.

The work that is closest to our work is the work by Lu et al. [13], who describe an approach to pre-process an event log by refining event labels with the goal of discovering a process model with duplicate activities. The method proposed by Lu et al., however, does not base the relabelings on data attributes of those events but instead bases them solely on the control flow context, leaving uncertainty whether two events relabeled differently are actually semantically different.

Another area of related work is data-aware process mining, where the aim is to discover rules with regard to data attributes of events that decide decision points in the process. De Leoni and van der Aalst [3] proposed a method that discovers data guards for decision points in the process based on alignments and decision tree learning. This approach relies on the discovery of a behaviorally well-fitting process model from the original event log. When only overgeneralizing process models (i.e. allowing for too much behavior) can be discovered from an event log, the correct decision points might not be present in the discovered process model at all, resulting in this approach not being able to discover the data dependencies that are in the event log. Our label refinements use data attributes prior to process discovery to enable discover more behaviorally constrained process models by bringing parts of the event attribute space to the event label.

6 Conclusion & Future Work

We have proposed a framework based on techniques from the field of circular statistics to refine event labels automatically based on their timestamp attribute. We have shown through a proof of concept on a real life event log that this framework can be used to discover label refinements that allow for discovery of more insightful and behaviorally more specific process models. An interesting area of future work is to explore the use of other types of event data attributes to refine labels, e.g. power values of sensors. A next research step would be to explore label refinements based on multiple data attributes combined. This would bring challenge of clustering on partially circular and partially euclidean data spaces.

References

1. Banerjee, A., Dhillon, I. S., Ghosh, J., and Sra, S. Clustering on the unit hypersphere using von mises-fisher distributions. *Journal of Machine Learning Research 6*, Sep (2005), 1345–1382.
2. Buijs, J. C. A. M., Van Dongen, B. F., and van der Aalst, W. M. P. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"* (2012), Springer, pp. 305–322.
3. de Leoni, M., and van der Aalst, W. M. P. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th annual ACM symposium on applied computing* (2013), ACM, pp. 1454–1461.
4. de Medeiros, A. K. A., Weijters, A. J. M. M., and van der Aalst, W. M. P. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery 14*, 2 (2007), 245–304.
5. Folino, F., Greco, G., Guzzo, A., and Pontieri, L. Discovering expressive process models from noised log data. In *Proceedings of the 2009 international database engineering & applications symposium* (2009), ACM, pp. 162–172.
6. Gu, C.-Q., Chang, H.-Y., and Yi, Y. Workflow mining: Extending the α-algorithm to mine duplicate tasks. In *2008 International Conference on Machine Learning and Cybernetics* (2008), vol. 1, IEEE, pp. 361–368.

7. Hartigan, J. A., and Hartigan, P. M. The dip test of unimodality. *The Annals of Statistics* (1985), 70–84.
8. Herbst, J., and Karagiannis, D. Workflow mining with inwolve. *Computers in Industry 53*, 3 (2004), 245–264.
9. Hornik, K., and Grün, B. movmf: An r package for fitting mixtures of von mises-fisher distributions. *Journal of Statistical Software 58*, 10 (2014), 1–31.
10. Kass, R. E., and Raftery, A. E. Bayes factors. *Journal of the American Statistical Association 90*, 430 (1995), 773–795.
11. Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. Discovering block-structured process models from event logs containing infrequent behaviour. In *International Conference on Business Process Management* (2013), Springer, pp. 66–78.
12. Li, J., Liu, D., and Yang, B. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In *Advances in Web and Network Technologies, and Information Management*. Springer, 2007, pp. 396–407.
13. Lu, X., Fahland, D., van den Biggelaar, F. J. H. M., and van der Aalst, W. M. P. Handling duplicated tasks in process discovery by refining event labels. In *International Conference on Business Process Management* (2016), Springer, p. To appear.
14. Mardia, K. V., and Jupp, P. E. *Directional statistics*, vol. 494. John Wiley & Sons, 2009.
15. Rao, J. Some tests based on arc-lengths for the circle. *Sankhyā: The Indian Journal of Statistics, Series B* (1976), 329–338.
16. Reisig, W., and Rozenberg, G. *Lectures on Petri nets I: basic models: advances in Petri nets*, vol. 1491. Springer Science & Business Media, 1998.
17. Schwarz, G. Estimating the dimension of a model. *The Annals of Statistics 6*, 2 (1978), 461–464.
18. Song, W., Liu, S., and Liu, Q. Business process mining based on simulated annealing. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for* (2008), IEEE, pp. 725–730.
19. Sztyler, T., Völker, J., Carmona, J., Meier, O., and Stuckenschmidt, H. Discovery of personal processes from labeled sensor data–an application of process mining to personalized health care. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data (ATAED)* (2015), pp. 22–23.
20. Tax, N., Sidorova, N., Haakma, R., and van der Aalst, W. M. P. Event abstraction for process mining using supervised learning techniques. In *Proceedings of the SAI Conference on Intelligent Systems (IntelliSys)* (2016), IEEE, pp. 161–170.
21. Tax, N., Sidorova, N., Haakma, R., and van der Aalst, W. M. P. Log-based evaluation of label splits for process models. *Procedia Computer Science* (2016), To appear.
22. van der Aalst, W. M. P. *Process mining: data science in action*. Springer Science & Business Media, 2016.
23. van Kasteren, T., Noulas, A., Englebienne, G., and Kröse, B. Accurate activity recognition in a home setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing* (2008), ACM, pp. 1–9.
24. Vázquez-Barreiros, B., Mucientes, M., and Lama, M. Mining duplicate tasks from discovered processes. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data (ATAED)* (2015), pp. 78–82.
25. Watson, G. S. Goodness-of-fit tests on a circle. ii. *Biometrika 49*, 1/2 (1962), 57–63.

Simple Bounded MTL Model Checking for Discrete Timed Automata (Extended abstract) ⋆

Agnieszka M. Zbrzezny and Andrzej Zbrzezny

IMCS, Jan Długosz University. Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
{agnieszka.zbrzezny,a.zbrzezny}@ajd.czest.pl

Abstract. We present a new translation of Metric Temporal Logic to the Linear Temporal Logic with a new set of the atomic propositions. We investigate a SAT-based bounded model checking method for Metric Temporal Logic that is interpreted over linear discrete infinite time models generated by discrete timed automata. We show how to implement the bounded model checking technique for Linear Temporal Logic with a new set of the atomic propositions and discrete timed automata, and as a case study we apply the technique in the analysis of the Timed Generic Pipeline Paradigm modelled by a network of discrete timed automata. We also present a comparison of the two translations of Metric Temporal Logic on common instances that can be scaled up to for performance evaluation. The theoretical description is supported by the experimental results that demonstrate the efficiency of the method.

1 Introduction

Bounded model checking [2, 3, 5] (BMC) is one of the symbolic model checking technique designed for finding witnesses for existential properties or counterexamples for universal properties. Its main idea is to consider a model reduced to a specific depth. The method works by mapping a bounded model checking problem to the satisfiability problem (SAT). For metric temporal logic (MTL) [4] and discrete time automata [1] the BMC method can by described as follows: given a model $\mathcal{M}$ for a discrete timed automaton , an MTL formula φ, and a bound k, a model checker creates a propositional formula $[\mathcal{M}, \varphi]_k$ that is satisfiable if and only if the formula φ is true in the model $\mathcal{M}$.

The novelty of our paper lies in :

1. defining a translation of the existential model checking problem for MTL to the existential model checking problem for linear temporal logic with additional propositional variables q_I. This logic is denoted by LTL$_q$;
2. defining bounded sematics for LTL$_q$ and defining the BMC algorithm;
3. implementing the new method.

The translation from MTL to LTL$_q$ requires neither new clocks nor new transitions, whereas the translation to HLTL [7] requires as many new clocks as there are intervals in a given formula. It also requires an exponential number of resetting transitions.

⋆ Partly supported by National Science Centre under the grant No. 2014/15/N/ST6/05079.

Moreover, our BMC method needs only one path, whereas the BMC method from [7] needs a number of paths depending on a given formula φ. Thus, one may expect that our method is much more effective since intuition is that an encoding which results in fewer variables and clauses is usually easier to solve.

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 and compare with the corresponding method [7].

The rest of the paper is structured as follows. In Section 2 we briefly recall the basic notion used through the paper. In Section 3 we define the translation to LTL$_q$. In Section 4 we define the BMC method for LTL$_q$. In Section 5 we discuss our experimental results. In Section 6 we conclude the paper.

2 Preliminaries

2.1 Discrete Timed Automata

Let $\mathbb{N}$ be a set of natural numbers. We assume a finite set $\mathbb{X} = \{x_0, \ldots, x_{n-1}\}$ of variables, called *clocks*. Each clock is a variable ranging over a set of non-negative natural numbers.

A *clock valuation* is a total function $v : \mathbb{X} \mapsto \mathbb{N}$ that assigns to each clock x a non-negative integer value $v(x)$. The set of all the clock valuations is denoted by $\mathbb{N}^n$. For $X \subseteq \mathbb{X}$, the valuation $v' = v[X := 0]$ is defined as: $\forall x \in X,\ v'(x) = 0$ and $\forall x \in \mathbb{X} \setminus X,\ v'(x) = v(x)$. For $\delta \in \mathbb{N}$, $v + \delta$ denotes the valuation v'' such that $\forall x \in \mathbb{X}, v''(x) = v(x) + \delta$. Let $x \in \mathbb{X}$, $c \in \mathbb{N}$, and $\sim \in \{<, \leqslant, =, \geqslant, >\}$. The set $\mathcal{C}(\mathbb{X})$ of *clock constraints* over the set of clocks $\mathbb{X}$ is defined by the following grammar:

$$\mathfrak{cc} := x \sim c \mid \mathfrak{cc} \wedge \mathfrak{cc}.$$

Let v be a clock valuation, and $\mathfrak{cc} \in \mathcal{C}(\mathbb{X})$. A clock valuation v satisfies a clock constraint $\mathfrak{cc}$, written as $v \models \mathfrak{cc}$, iff $\mathfrak{cc}$ evaluates to true using the clock values given by the valuation v.

Definition 1. *A* discrete timed automaton $\mathcal{A}$ *is a tuple* $(Act, Loc, \ell^0, T, \mathbb{X}, Inv, AP, V)$*, where* Act *is a finite set of actions,* Loc *is a finite set of locations,* $\ell^0 \in Loc$ *is an initial location,* $T \subseteq Loc \times Act \times \mathcal{C}(\mathbb{X}) \times 2^{\mathbb{X}} \times Loc$ *is a transition relation,* $\mathbb{X}$ *is a finite set of clocks,* $Inv : Loc \mapsto \mathcal{C}(\mathbb{X})$ *is a state invariant function,* AP *is a set of atomic proposition, and* $V : Loc \mapsto 2^{AP}$ *is a valuation function assigning to each location a set of atomic propositions true in this location.*

Each element $t \in T$ *is denoted by* $\ell \xrightarrow{\sigma, \mathfrak{cc}, X} \ell'$*, and it represents a transition from location* ℓ *to location* ℓ' *on the input action* σ*.* $X \subseteq \mathbb{X}$ *is the set of the clocks to be reset with this transition, and* $\mathfrak{cc} \in \mathcal{C}(\mathbb{X})$ *is the enabling condition for* t*.*

The semantics of the discrete timed automaton is defined by associating a transition system with it, which we call a *concrete model*.

Definition 2. *Let* $\mathcal{A} = (Act, Loc, \ell^0, T, \mathbb{X}, Inv, AP, V)$ *be a discrete timed automaton, and* v^0 *a clock valuation such that* $\forall x \in \mathbb{X},\ v^0(x) = 0$*. A* concrete model *for* $\mathcal{A}$ *is*

a tuple $\mathcal{M}_{\mathcal{A}} = (Q, q^0, \longrightarrow, \mathcal{V})$, where $Q = Loc \times \mathbb{N}^n$ is a set of the concrete states, $q^0 = (\ell^0, v^0)$ is the initial state, $\longrightarrow \subseteq Q \times Q$ is a total binary relation on Q defined by action and time transitions as follows. For $\sigma \in Act$ and $\delta \in \mathbb{N}$,

1. *Action transition: $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ iff there is a transition $\ell \xrightarrow{\sigma, \mathfrak{cc}, X} \ell' \in T$ such that $v \models \mathfrak{cc} \wedge Inv(\ell)$ and $v' = v[X := 0]$ and $v' \models Inv(\ell')$,*
2. *Time transition: $(\ell, v) \xrightarrow{\delta} (\ell, v + \delta)$ iff $v \models Inv(\ell)$ and $v + \delta \models Inv(\ell)$.*

$\mathcal{V} : Q \mapsto 2^{AP}$ is a valuation function such that $\mathcal{V}((\ell, v)) = V(\ell)$ for all $(\ell, v) \in Q$.

A *run* ρ of $\mathcal{A}$ is an infinite sequence of concrete states: $q_0 \xrightarrow{\delta_0, \sigma_0} q_1 \xrightarrow{\delta_1, \sigma_1} q_2 \xrightarrow{\delta_2, \sigma_2} \dots$ such that $q_i \in Q$, $\sigma_i \in Act$, and $\delta_i \in \mathbb{N}_+$ for each $i \in \mathbb{N}$. Notice that our runs are *strongly monotonic*. This is because the definition of the run does not permit two consecutive actions to be performed one after the other, i.e., between each two actions some time must pass.

2.2 Metric Temporal Logic (MTL)

Let $p \in AP$, and I be an interval in $\mathbb{N}$ of the form: $[a, b)$ or $[a, \infty)$, for $a, b \in \mathbb{N}$ and $a \neq b$. The MTL in release positive normal form is defined by the following grammar:

$$\alpha := \mathbf{true} \mid \mathbf{false} \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \mathbf{U}_I \alpha \mid \mathbf{G}_I \alpha.$$

Intuitively, $\mathbf{U}_I$ and $\mathbf{G}_I$ are the operators for *bounded until* and for *bounded always*. The formula $\alpha \mathbf{U}_I \beta$ is true in a computation if β is true in the interval I at least in one state and always earlier α holds. The formula $\mathbf{G}_I \alpha$ is true in a computation α is true at all states of the computation that are in the interval I. The derived basic modality is defined as follows: $\mathbf{F}_I \alpha \stackrel{def}{=} \mathbf{true} \mathbf{U}_I \alpha$ (*bounded eventually*).

Let $\mathcal{A}$ be a discrete timed automaton, $\mathcal{M}_{\mathcal{A}} = (Q, q^0, \longrightarrow, \mathcal{V})$ a concrete model for $\mathcal{A}$, $\rho : q_0 \xrightarrow{\delta_0, \sigma_0} q_1 \xrightarrow{\delta_1, \sigma_1} q_2 \xrightarrow{\delta_2, \sigma_2} \dots$ a run of $\mathcal{A}$, and α, β formulae of MTL. In order to define the satisfiability relation for MTL, we need to define the notion of a *discrete path λ_ρ corresponding to run ρ* [6]. This can be done in a unique way because of the assumption that $\delta_i \in \mathbb{N}_+$. First, define the sequence $\Delta_0 = [b_0, b_1), \Delta_1 = [b_1, b_2), \Delta_2 = [b_2, b_3), \dots$ of pairwise disjoint intervals, where: $b_0 = 0$, and $b_i = b_{i-1} + \delta_{i-1}$ if $i > 0$. Now, for each $t \in \mathbb{N}$, let $idx_\rho(t)$ denote the unique index i such that $t \in \Delta_i$. A *discrete path* (or *path*) λ_ρ *corresponding to* ρ is a mapping $\lambda_\rho : \mathbb{N} \mapsto Q$ such that $\lambda_\rho(t) = (\ell_i, v_i + t - b_i)$, where $i = idx_\rho(t)$. Given $t \in \mathbb{N}$, the suffix λ_ρ^t of a path λ_ρ at time t is a path defined as: $\forall i \in \mathbb{N}$, $\lambda_\rho^t(i) = \lambda_\rho(t + i)$.

In order to improve readability, in the following definition we write $\lambda_\rho^t \models_{\mathrm{MTL}} \varphi$ instead of $\mathcal{M}_\varphi, \lambda_\rho^t \models_{\mathrm{MTL}} \varphi$, for any MTL formula φ.

Definition 3. *The* satisfiability *relation $\models_{\mathrm{MTL}}$, which indicates truth of an* MTL *formula in the concrete model $\mathcal{M}_{\mathcal{A}}$ along a path λ_ρ at time $t \in \mathbb{N}$, is defined inductively as follows:*

- $\lambda_\rho^t \models_{\mathrm{MTL}} \mathbf{true}$, $\lambda_\rho^t \not\models_{\mathrm{MTL}} \mathbf{false}$,
- $\lambda_\rho^t \models_{\mathrm{MTL}} p$ *iff* $p \in \mathcal{V}(\lambda_\rho(t))$, $\lambda_\rho^t \models_{\mathrm{MTL}} \neg p$ *iff* $p \notin \mathcal{V}(\lambda_\rho(t))$,

- $\lambda_\rho^t \models_{\mathrm{MTL}} \alpha \wedge \beta$ *iff* $\lambda_\rho^t \models_{\mathrm{MTL}} \alpha$ *and* $\lambda_\rho^t \models_{\mathrm{MTL}} \beta$,
- $\lambda_\rho^t \models_{\mathrm{MTL}} \alpha \vee \beta$ *iff* $\lambda_\rho^t \models_{\mathrm{MTL}} \alpha$ *or* $\lambda_\rho^t \models_{\mathrm{MTL}} \beta$,
- $\lambda_\rho^t \models_{\mathrm{MTL}} \alpha \mathbf{U}_I \beta$ *iff* $(\exists t' \in I)(\lambda_\rho^{t+t'} \models_{\mathrm{MTL}} \beta$ *and* $(\forall 0 \leqslant t'' < t')\lambda_\rho^{t+t''} \models_{\mathrm{MTL}} \alpha)$,
- $\lambda_\rho^t \models_{\mathrm{MTL}} \mathbf{G}_I \beta$ *iff* $(\forall t' \in I)(\lambda_\rho^{t+t'} \models_{\mathrm{MTL}} \beta)$.

As $\lambda_\rho^0 = \lambda_\rho$, we shall write $\mathcal{M}_\mathcal{A}, \lambda_\rho \models_{\mathrm{MTL}} \varphi$ for $\mathcal{M}_\mathcal{A}, \lambda_\rho^0 \models_{\mathrm{MTL}} \varphi$. An MTL formula φ is *existentially valid* in the model $\mathcal{M}_\mathcal{A}$, denoted $\mathcal{M}_\mathcal{A} \models_{\mathrm{MTL}} \mathbf{E}\varphi$, if, and only if $\mathcal{M}_\mathcal{A}, \lambda_\rho \models_{\mathrm{MTL}} \varphi$ for some path λ_ρ starting in the initial state of $\mathcal{M}_\mathcal{A}$. Determining whether an MTL formula φ is existentially valid in a given model is called an *existential model checking problem*.

3 Translation from MTL to LTL$_q$

3.1 Abstract model

Let φ be an MTL formula and $\mathcal{A} = (Act, Loc, \ell^0, T, \mathbb{X}, Inv, AP, V)$ be a discrete timed automaton with $\mathbb{X} = \{x_0, \ldots, x_{n-1}\}$. For each $j \in \{0, \ldots, n-1\}$, let c_j^{max} be the largest constant appearing in intervals of φ and in any enabling condition involving the clock x_j and used in the state invariants and guards of $\mathcal{A}$. For two clock valuations v and v' in $\mathbb{N}^n$, we say that $v \simeq v'$ iff for each $0 \leqslant j < n$ either $v(x_j) > c_j^{max}$ and $v'(x_j) > c_j^{max}$ or $v(x) \leqslant c_j^{max}$ and $v'(x) \leqslant c_j^{max}$ and $v(x) = v'(x)$.

It is well known, that the relation $\simeq$ is an equivalence relation, what gives rise to construct an finite abstract model. To this end we define the set of possible values of the clock x_j in the abstract model as $\mathbb{D}_j = \{0, \ldots, c_j^{max}+1\}$ for $0 \leqslant j < n$. Moreover, for two clock valuations v and v' in $\mathbb{D}_0 \times \ldots \times \mathbb{D}_{n-1}$, we say that v' is the *time successor* of v (denoted $succ(v)$) as follows: for each $0 \leqslant j < n$,

$$succ(v)(x_j) = \begin{cases} v(x_j) + 1, \text{ if } v(x_j) \leqslant c_j^{max}, \\ c_j^{max} + 1, \text{ if } v(x_j) = c_j^{max} + 1. \end{cases}$$

Definition 4. *Let $\mathcal{A} = (Act, Loc, \ell^0, T, \mathbb{X}, Inv, AP, V)$ be a discrete timed automaton, and φ an MTL formula build over the set AP of atomic propositions. An* abstract model *for the automaton $\mathcal{A}$ and the formula φ is a tuple $\mathcal{M}_\varphi = (\widehat{S}, s^0, \hookrightarrow, \widehat{\mathcal{V}})$, where $\widehat{S} = L \times (\mathbb{D}_0 \times \ldots \times \mathbb{D}_{n-1})$ is the set of abstract states, $s^0 = (\ell^0, \{0\}^{n+1})$ is the initial state, $\widehat{\mathcal{V}} : \widehat{S} \to 2^{AP}$ is a valuation function such that for all $p \in AP$, $p \in \widehat{\mathcal{V}}((\ell, v))$ iff $p \in V(\ell)$, and $\hookrightarrow \subseteq S \times Act' \times S$, where $Act' = Act \cup \{\tau\}$, is a transition relation defined by the time and action transitions:*

- *Time transition: $(\ell, v) \overset{\tau}{\hookrightarrow} (\ell, v')$ iff $v \models Inv(\ell)$, $v' = succ(v)$, and $v' \models Inv(\ell)$,*
- *Action transition: for any $\sigma \in Act$, $(\ell, v) \overset{\sigma}{\hookrightarrow} (\ell', v')$ iff there exists a transition $\ell \overset{\sigma, \mathfrak{cc}, X}{\longrightarrow} \ell' \in T$ such that $v \models \mathfrak{cc} \wedge Inv(\ell)$, $v' = v[X := 0]$, and $v' \models Inv(\ell')$.*

Definition 5. *A* path *in $\mathcal{M}_\varphi$ is a sequence $\pi = (s_0, s_1, \ldots)$ of states such that for each $j \in \mathbb{N}$, either $(s_j \overset{\tau}{\hookrightarrow} s_{j+1})$ or $(s_j \overset{\sigma}{\hookrightarrow} s_{j+1})$ for some $\sigma \in Act$, and every action transition is preceded by at least one time transition.*

The above definition of the path ensures that the first transition is the time one, and that between each two action transitions at least one time transition appears.

For a path π, $\pi(j)$ denotes the j-th state s_j of π, $\pi[..j] = (\pi(0), \ldots, \pi(j))$ denotes the j-th prefix of π ending with $\pi(j)$, and $\pi^j = (s_j, s_{j+1}, \ldots)$ denotes the j-th suffix of π starting with $\pi(j)$.

Given a path π one can define a function $\zeta_\pi : \mathbb{N} \mapsto \mathbb{N}$ such that for each $j \geqslant 0$, $\zeta_\pi(j)$ is equal to the number of time transitions on the prefix $\pi[..j]$. Let us note that for each $j \geqslant 0$, $\zeta_\pi(j)$ gives the value of the global time in the j-th state of the path π.

3.2 The logic LTL$_q$

Let $\mathcal{I}$ be the set of all intervals in $\mathbb{N}$. Let $AP_{\mathcal{I}} = \{q_I \mid I \in \mathcal{I}\}$. The LTL$_q$ formulae in the negation normal form are defined by the following grammar:

$$\psi ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg p \mid q_I \mid \neg q_I \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \mathbf{U} \psi \mid \mathbf{G}\psi,$$

where $p \in AP$ and $q_I \in AP_{\mathcal{I}}$. The temporal modalities $\mathbf{U}$ and $\mathbf{G}$ are, respectively, named as the *until* and the *always*. The derived basic temporal modality for *eventually* is defined in the standard way: $\mathbf{F}\psi \stackrel{def}{=} \mathbf{true}\mathbf{U}\psi$.

In order to improve readability, in the following definition we write $\langle \pi, m\rangle \models_k \psi$ instead of $\mathcal{M}_\varphi, \langle \pi, m\rangle \models_k \psi$, for any LTL$_q$ formula ψ.

Definition 6. *The* satisfiability *relation $\models^d$, which indicates truth of an LTL$_q$ formula in the abstract model $\mathcal{M}_\varphi$ along the path π with the starting point m and at the depth $d \geqslant m$, is defined inductively as follows:*

- $\langle \pi, m\rangle \models^d \mathbf{true}$, $\quad \langle \pi, m\rangle \not\models^d \mathbf{false}$,
- $\langle \pi, m\rangle \models^d p$ *iff* $p \in \mathcal{V}(\pi(d))$, $\quad \langle \pi, m\rangle \models^d \neg p$ *iff* $p \notin \mathcal{V}(\pi(d))$,
- $\langle \pi, m\rangle \models^d q_I$ *iff* $\zeta_\pi(d) - \zeta_\pi(m) \in I$,
- $\langle \pi, m\rangle \models^d \neg q_I$ *iff* $\zeta_\pi(d) - \zeta_\pi(m) \notin I$,
- $\langle \pi, m\rangle \models^d \alpha \wedge \beta$ *iff* $\langle \pi, m\rangle \models^d \alpha$ *and* $\langle \pi, m\rangle \models^d \beta$,
- $\langle \pi, m\rangle \models^d \alpha \vee \beta$ *iff* $\langle \pi, m\rangle \models^d \alpha$ *or* $\langle \pi, m\rangle \models^d \beta$,
- $\langle \pi, m\rangle \models^d \alpha \mathbf{U} \beta$ *iff* $(\exists j \geqslant d)(\langle \pi, d\rangle \models^j \beta$ *and* $(\forall d \leqslant i < j)\, \langle \pi, d\rangle \models^i \alpha)$,
- $\langle \pi, m\rangle \models^d \mathbf{G}\beta$ *iff* $(\forall j \geqslant d)\, \langle \pi, d\rangle \models^j \beta$.

An LTL$_q$ formula ψ *existentially holds* in the model $\mathcal{M}_\varphi$, written $\mathcal{M}_\varphi \models \mathbf{E}\psi$, if, and only if $\mathcal{M}_\varphi, \langle \pi, 0\rangle \models^0 \psi$ for some path π starting at the initial state. The *existential model checking problem* asks whether $\mathcal{M}_\varphi \models \mathbf{E}\psi$.

3.3 Translation

Let $p \in AP$, α and β be formulae of MTL. We define the translation from MTL into LTL$_q$ as a function $\mathrm{tr} : \mathrm{MTL} \to \mathrm{LTL}_q$ in the following way:

- $\mathrm{tr}(\mathbf{true}) = \mathbf{true}$, $\mathrm{tr}(\mathbf{false}) = \mathbf{false}$, $\mathrm{tr}(p) = p$, $\mathrm{tr}(\neg p) = \neg p$,
- $\mathrm{tr}(\alpha \wedge \beta) = \mathrm{tr}(\alpha) \wedge \mathrm{tr}(\beta)$, $\mathrm{tr}(\alpha \vee \beta) = \mathrm{tr}(\alpha) \vee \mathrm{tr}(\beta)$,
- $\mathrm{tr}(\alpha \mathbf{U}_I \beta) = \mathrm{tr}(\alpha) \mathbf{U} \mathrm{tr}(q_I \wedge \beta)$, $\mathrm{tr}(\mathbf{G}_I \beta) = \mathbf{G}(\neg q_I \vee \mathrm{tr}(\beta))$

Observe that the translation of literals as well as logical connectives is straightforward. The translation of the $\mathbf{U}_I$ operator ensures that β holds somewhere in the interval I (this is expressed by the requirement $q_I \wedge \mathrm{tr}(\beta)$), and α holds always before β. Similarly, the translation of the $\mathbf{G}_I$ operator ensures that β always holds in the interval I (this is expressed by the requirement $\neg q_I \vee \mathrm{tr}(\beta)$).

Theorem 1. *Let $\mathcal{A}$ be a discrete timed automaton, $\mathcal{M}_\mathcal{A}$ the concrete model for $\mathcal{A}$, φ an MTL formula, and $\mathcal{M}_\varphi$ the abstract model for the automaton $\mathcal{A}$ and the formula φ. Then, $\mathcal{M}_\mathcal{A} \models \mathbf{E}\varphi$ if, and only if $\mathcal{M}_\varphi \models \mathbf{E}\mathrm{tr}(\varphi)$.*

4 Bounded model checking

In this section we define a *bounded semantics* for LTL_q in order to translate the *existential model checking problem* for LTL_q into the satisfiability problem.

4.1 Bounded semantics

To define the bounded semantics one needs to represent infinite paths in a model in a special way. To this aim, we recall the notions of *k-paths* and *loops* [8].

Definition 7. *Let $\mathcal{M}_\varphi$ be a model, $k \in \mathbb{N}$, and $0 \leqslant l \leqslant k$. A* k-path *is a pair (π, l), also denoted by π_l, where π is a finite sequence $\pi = (s_0, \ldots, s_k)$ of states such that for each $0 \leqslant j < k$, either $(s_j \overset{\tau}{\hookrightarrow} s_{j+1})$ or $(s_j \overset{\sigma}{\hookrightarrow} s_{j+1})$ for some $\sigma \in Act$, and every action transition is preceded by at least one time transition. A k-path π_l is a* loop*, written $\circlearrowright(\pi_l)$ for short, if $l < k$ and $\pi(k) = \pi(l)$.*

If a k-path π_l is a loop it represents the infinite path of the form uv^ω, where $u = (\pi(0), \ldots, \pi(l))$ and $v = (\pi(l+1), \ldots, \pi(k))$. We denote this unique path by $\widetilde{\pi}_l$. Note that for each $j \in \mathbb{N}$, $\widetilde{\pi}_l^{\,l+j} = \widetilde{\pi}_l^{\,k+j}$.

In the definition of bounded semantics for variables from $AP_\mathcal{I}$ one needs to use only a finite prefix of the sequence $(\zeta_{\widetilde{\pi}_l}(0), \zeta_{\widetilde{\pi}_l}(1), \ldots)$. Namely, for a k-path π_l that is not a loop the prefix of the length k is needed, and for a k-path π_l that is a loop the prefix of the length $k + k - l$ is needed.

In order to improve readability, in the following definition we write $\langle \pi_l, m\rangle \models_k \psi$ instead of $\mathcal{M}_\varphi, \langle \pi_l, m\rangle \models_k \psi$, for any LTL_q formula ψ.

Definition 8 (Bounded semantics). *Let $\mathcal{M}_\varphi$ be the abstract model, π_l be a k-path in $\mathcal{M}_\varphi$, and $0 \leqslant m, d \leqslant k$. The relation $\models_k^d$ is defined inductively as follows:*

- $\langle \pi_l, m\rangle \models_k^d \mathbf{true}$, $\quad \langle \pi_l, m\rangle \not\models_k^d \mathbf{false}$,
- $\langle \pi_l, m\rangle \models_k^d p$ *iff* $p \in \mathcal{V}(\pi_l(d))$, $\quad \langle \pi_l, m\rangle \models_k^d \neg p$ *iff* $p \notin \mathcal{V}(\pi_l(d))$,
- $\langle \pi_l, m\rangle \models_k^d q_I$ *iff* $\begin{cases} \zeta_{\widetilde{\pi}_l}(d) - \zeta_{\widetilde{\pi}_l}(m) \in I, & \text{if } \pi_l \text{ is not a loop,} \\ \zeta_{\widetilde{\pi}_l}(d) - \zeta_{\widetilde{\pi}_l}(m) \in I, & \text{if } \pi_l \text{ is a loop and } d \geqslant m, \\ \zeta_{\widetilde{\pi}_l}(d + k - l) - \zeta_{\widetilde{\pi}_l}(m) \in I, & \text{if } \pi_l \text{ is a loop and } d < m, \end{cases}$
- $\langle \pi_l, m\rangle \models_k^d \neg q_I$ *iff* $\langle \pi_l, m\rangle \not\models_k^d q_I$

– $\langle \pi_l, m\rangle \models^d_k \alpha \wedge \beta$ *iff* $\langle \pi_l, m\rangle \models^d_k \alpha$ *and* $\langle \pi_l, m\rangle \models^d_k \beta$,
– $\langle \pi_l, m\rangle \models^d_k \alpha \vee \beta$ *iff* $\langle \pi_l, m\rangle \models^d_k \alpha$ *or* $\langle \pi_l, m\rangle \models^d_k \beta$,
– $\langle \pi_l, m\rangle \models^d_k \alpha \mathbf{U} \beta$ *iff* $(\exists_{d \leqslant j \leqslant k})\big(\langle \pi_l, d\rangle \models^j_k \beta$ *and* $(\forall_{d \leqslant i < j})\, \langle \pi_l, d\rangle \models^j_k \alpha\big)$
or $\big(\supset(\pi_l)$ *and* $(\exists_{l<j<d})\, \langle \pi_l, d\rangle \models^j_k \beta$ *and* $(\forall_{l<i<k})\ \langle \pi_l, d\rangle \models^j_k \alpha$
and $(\forall_{d \leqslant i \leqslant k})\ \langle \pi_l, d\rangle \models^j_k \alpha\big)$,
– $\langle \pi_l, m\rangle \models^d_k \mathbf{G}\beta$ *iff* $\supset(\pi_l)$ *and* $(\forall_{j \leqslant k}) j \geqslant \min(d, l)$ *implies* $\langle \pi_l, d\rangle \models^j_k \beta$.

An LTL$_q$ formula ψ *existentially k-holds* in the model $\mathcal{M}_\varphi$, written $\mathcal{M}_\varphi \models_k \mathbf{E}\psi$, if, and only if $\mathcal{M}_\varphi, \langle \pi, 0\rangle \models^0_k \psi$ for some path π starting at the initial state.

Theorem 2. *Let $\mathcal{A}$ be a discrete timed automaton, φ an MTL formula, and $\mathcal{M}_\varphi$ the abstract model for the automaton $\mathcal{A}$ and the formula φ. Moreover, let $\psi = \mathrm{tr}(\varphi)$. Then, $\mathcal{M}_\varphi \models \mathbf{E}\psi$ if, and only if there exists $k \geqslant 0$ such that $\mathcal{M}_\varphi \models_k \mathbf{E}\psi$.*

4.2 Translation to SAT

The last step of our method is the standard one (see [8, 7]). It consists in encoding the transition relation of $\mathcal{M}_\varphi$ and the LTL formula $\mathrm{tr}(\varphi)$. The only novelty lies in encoding of the finite prefix of the sequence $(\zeta_{\widetilde{\pi}_l}(0), \zeta_{\widetilde{\pi}_l}(1), \ldots)$. The translation to SAT results in the propositional formula $[\mathcal{M}_\varphi, \mathrm{tr}(\varphi)]_k$ with the property expressed in the following theorem.

Theorem 3. *Let $\mathcal{M}_\varphi$ be an abstract model. Then, for every $k \in \mathbb{N}$, $\mathcal{M}_\varphi \models^d_k \mathbf{E}\mathrm{tr}(\varphi)$ if, and only if, the propositional formula $[\mathcal{M}_\varphi, \mathrm{tr}(\varphi)]_k$ is satisfiable.*

5 Experimental results

In this section we experimentally evaluate the performance of our new translation. We have conducted the experiments using the slightly modified timed generic pipeline paradigm (TGPP) [7].

5.1 Timed Generic Pipeline Paradigm

The Timed Generic Pipeline Paradigm (TGPP) discrete timed automata model shown in Figure 1 consists of Producer producing data within the certain time interval $([a, b])$ or being inactive, Consumer receiving data within the certain time interval $([c, d])$ or being inactive within the certain time interval $([g, h])$, and a chain of n intermediate Nodes which can be ready for receiving data within the certain time interval $([c, d])$, processing data within the certain time interval $([e, f])$ or sending data. We assume that $a = c = e = g = 1$ and $b = d = f = h = 2 \cdot n + 2$, where n represents number of nodes in the TGPP.

To compare our experimental results with [7], we have tested the TGPP discrete timed automata model, scaled in the number of intermediate nodes on the following MTL formulae that existentially hold in the model of TGPP (n is the number of nodes):

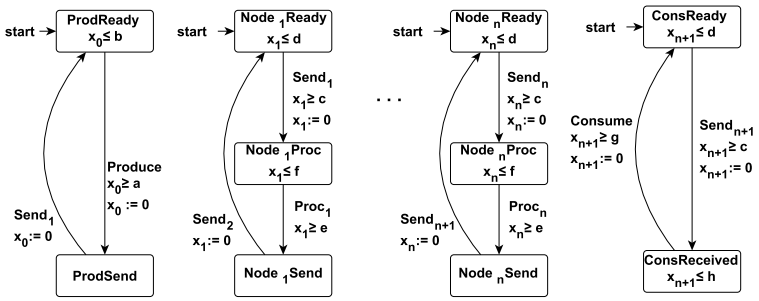


Fig. 1: The TGPP system.

- $\varphi_0 = \mathbf{F}_{[0,2\cdot n+3)}(ConsReceived)$. It expresses that Consumer receives the data in at most $2 \cdot n + 3$ time units.
- $\varphi_1 = \mathbf{G}_{[0,2\cdot n+2)}(ConsReady)$. It states that the Consumer is always forced to receive the data in $2 \cdot n + 2$ time units.
- $\varphi_2 = \mathbf{G}_{[0,\infty)}(ProdReady \vee ConsReady)$. It states that always either the Producer has sent the data or the Consumer has received the data.
- $\varphi_3 = \mathbf{F}_{[0,2\cdot n+3)}(\mathbf{G}_{[0,\infty)}(ProdSend \vee ConsReceived))$. It states that eventually in time less then $2 \cdot n + 3$ it is always the case that the Producer is ready to send the data or the Consumer has received the data.
- $\varphi_4 = \mathbf{G}_{[0,\infty)}(\mathbf{F}_{[0,2\cdot n+3)}(ConsReceived))$. It states that the Consumer infinitely often is receiving the data in time less then $2 \cdot n + 3$.
- $\varphi_5 = \mathbf{G}_{[0,\infty)}(\mathbf{F}_{[0,2\cdot n+3)}(ProdSend) \wedge \mathbf{G}_{[0,\infty)}(\mathbf{F}_{[0,2\cdot n+3)}(ConsReceived))$. It states that the Producer infinitely often is sending the data in the time less then $2 \cdot n + 3$ and the Consumer infinitely often is receiving the data in time less then $2 \cdot n + 3$.

5.2 Performance evaluation

We have performed our experimental results on a computer equipped with I7-3770 processor, 32 GB of RAM, and the operating system Linux with the kernel 4.6.4. Our SAT-based BMC algorithms are implemented as standalone programs written in the programming language `C++`. We used the state of the art SAT-solver CryptoMiniSat5 (`http://www.msoos.org/`).

All the benchmarks together with instructions on how to reproduce our experimental results can be found at the web page `http://ajd.czest.pl/~a.zbrzezny/bmc.html`.

The number of considered k-paths (f_k) for the new translation is always equal to 1 and for the old translation is respectively equal to: $f_k(\varphi_0) = 2$, $f_k(\varphi_1) = 2$, $f_k(\varphi_2) = 2$, $f_k(\varphi_3) = 3$, $f_k(\varphi_4) = 8 \cdot n + 9$, $f_k(\varphi_5) = 16 \cdot n + 17$. The length of the witness for the formula φ_0 is equal to $4 \cdot n + 4$; for the formula φ_1 is equal to $8 \cdot n + 6$; for the formula φ_2 and is equal to $8 \cdot n + 6$; for the formula φ_3 is equal to $8 \cdot n + 15$; for the formula φ_4 is equal to $8 \cdot n + 6$; for the formula φ_5 is equal to $8 \cdot n + 6$.

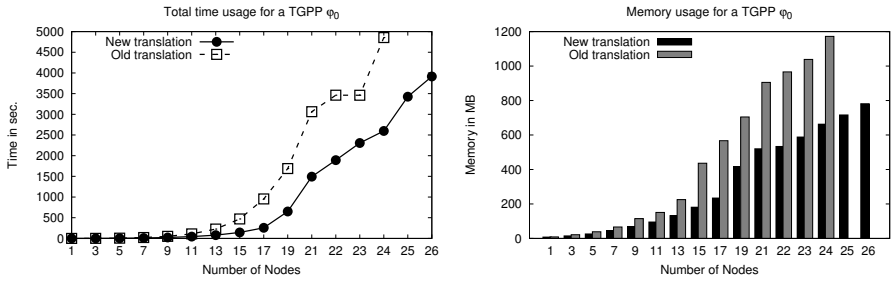


Fig. 2: φ_0: TGPP with n nodes.

From Fig. 2 one can observe that the new method is able to verify the formula φ_0 for TGPP with 27 nodes. The old method is able to verify the formula φ_0 for TGPP with 24 nodes. The memory usage for the old method is 1.72 times higher than for the new method for 24 nodes.

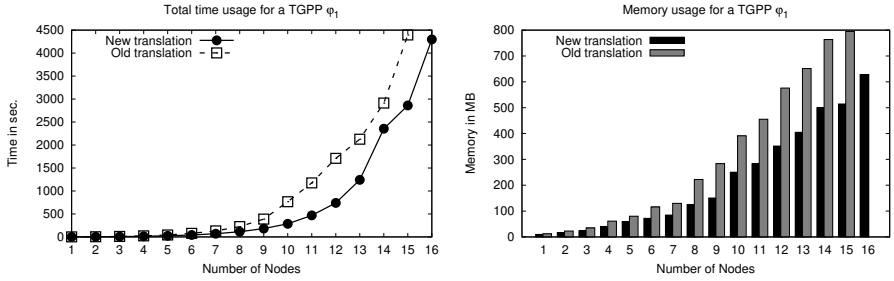


Fig. 3: φ_1: TGPP with n nodes.

From Fig. 3 one can observe that the new method is able to verify the formula φ_1 for TGPP with 17 nodes. The old method is able to verify the formula φ_1 for TGPP with 15 nodes. The memory usage for the old method is 1.50 times higher than for the new method for 15 nodes.

From Fig. 4 one can observe that the new method is able to verify the formula φ_2 for TGPP with 18 nodes. The old method is able to verify the formula φ_2 for TGPP with 15 nodes. The memory usage for the old method is 1.55 times higher than for the new method for 15 nodes.

From Fig. 5 one can observe that the new method is able to verify the formula φ_3 for TGPP with 17 nodes. The old method is able to verify the formula φ_3 for TGPP with 13 nodes. The memory usage for the old method is 2.21 times higher than for the new method for 13 nodes.

From Fig. 6 one can observe that the new method is able to verify the formula φ_4 for TGPP with 13 nodes. The old method is able to verify the formula φ_4 for TGPP

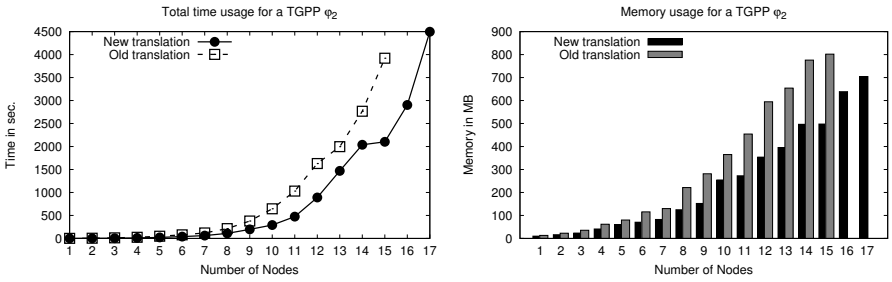


Fig. 4: φ_2: TGPP with n nodes.

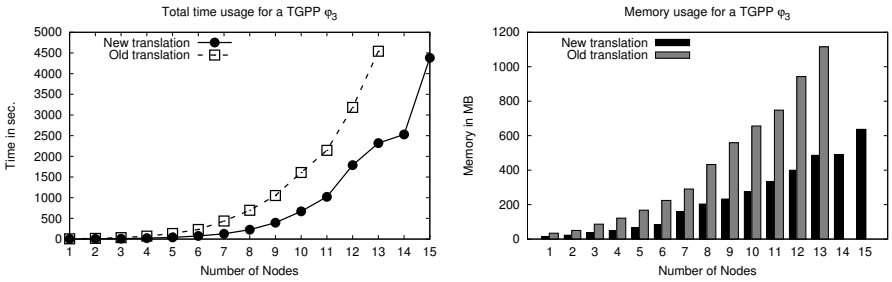


Fig. 5: φ_3: TGPP with n nodes.

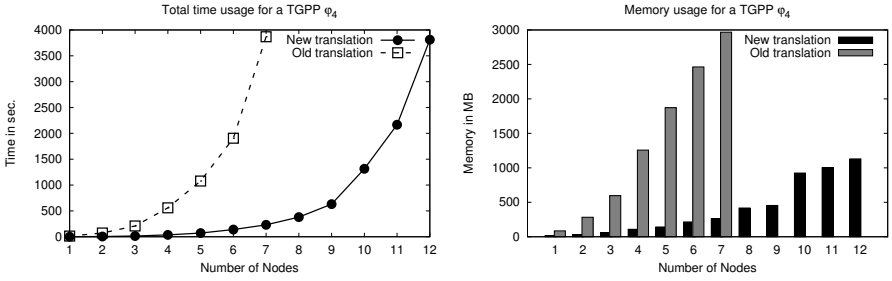


Fig. 6: φ_4: TGPP with n nodes.

with 7 nodes. The memory usage for the old method is 10.32 times higher than for the new method for 7 nodes.

From Fig. 8 one can observe that the new method is able to verify the formula φ_5 for TGPP with 13 nodes. The old method is able to verify the formula φ_5 for TGPP with 6 nodes. The memory usage for the old method is 21.39 times higher than for the new method for 6 nodes.

For the formula φ_4 the new method generates only 395775 variables and 1229009 clauses (Fig. 7) for 7 nodes. The time consumed by BMC to generate the set of clauses is equal 65.35 sec. The memory consumed by BMC to generate the set of clauses is

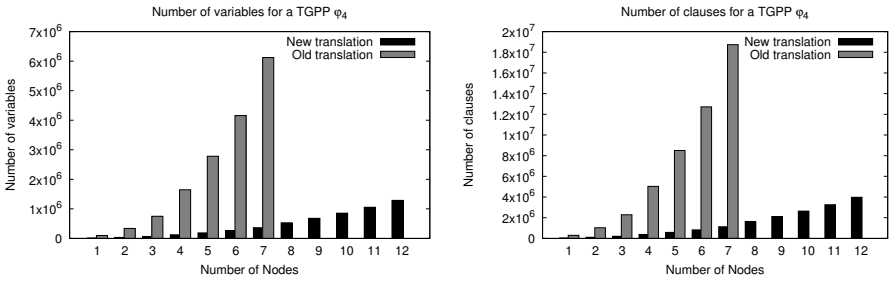


Fig. 7: Number of variables and clauses for φ_4 and TGPP with n nodes.

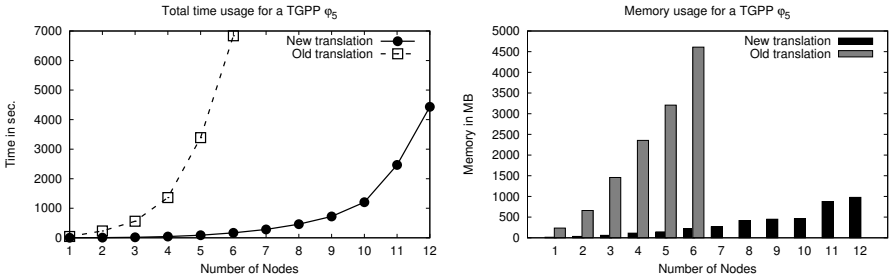


Fig. 8: φ_5: TGPP with n nodes.

equal 100.54 MB. The time and memory consumed by the state of the art SAT solver CryptoMiniSat5, respectively is equal to 112.59 sec. and 287.41 MB.

The old method generates 6122646 variables and 18739998 clauses. The time consumed by BMC to generate the set of clauses is equal 977.18 sec. The memory consumed by BMC to generate the set of clauses is equal 1505.05 MB. The time and memory consumed by the state of the art SAT solver CryptoMiniSat5, respectively is equal to 2892.31 sec. and 2967.82 MB.

The example above shows that our new method results in reducing the overall runtime and memory of BMC to construct a CNF formula, and of SAT solver to check satisfiability of this formula.

6 Conclusions

We have proposed, implemented, and experimentally evaluated SAT-based BMC method for soft real-time systems, which are modelled by discrete timed automata, and for properties expressible in MTL with the semantics over discrete timed automata. The method is based on a translation of the existential model checking for MTL to the existential model checking for LTL_q, and then on the translation of the existential model checking for LTL_q to the propositional satisfiability problem.

In the following table we compare the new BMC method with the old one.

Simple BMC+DTA&MTL	BMC+DTA&MTL[6]
no new clocks	a number of new clocks equal to the number of intervals in the given formula
no new transitions	exponential number of new transitions
only one path	a number of paths depending on the given formula and the length of the k-path
smaller number of propositional variables and clauses	substantially larger number of propositonal variables and clauses
better time and memory usage	worse time and memory usage

Table 1: The comparison of two methods

The experimental results show that our approach is much better than the approach based on translation to HLTL. The reason is that the new method needs only one new path, does not need any new clocks, and does not need any new transitions. The experiments confirm that the improvement in question leads to a reduction of the size of the CNF formulas submitted to the SAT solver, and therefore to a significant reduction both in the time and memory required by the SAT solver to return an answer. The paper presents preliminary experimental results only, but they show that the proposed verification method is quite efficient and worth exploring.

Therefore, in our future work we are going to define the SMT-based BMC encoding for DTA and for LTL_q and compare this encoding with the SAT-based encoding, and we would like to develop SAT-based BMC method for timed automata and properties expressible in TECTL.

References

1. R. Alur and D. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. 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*, pages 193–207. Springer-Verlag, 1999.
3. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.
4. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
5. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
6. Bożena Woźna-Szcześniak and Andrzej Zbrzezny. A translation of the existential model checking problem from MITL to HLTL. *Fundamenta Informaticae*, 122(4):401–420, 2013.
7. Bożena Woźna-Szcześniak and Andrzej Zbrzezny. Checking MTL properties of discrete timed automata via bounded model checking. *Fundam. Inform.*, 135(4):553–568, 2014.
8. A. Zbrzezny. A new translation from ECTL* to SAT. *Fundamenta Informaticae*, 120(3-4):377–397, 2012.

Rough Sets and Sorites Paradox

Andrzej Jankowski[1], Andrzej Skowron[2,3], and Piotr Wasilewski[2]

[1] The Dziubanski Knowledge Technology Foundation
Nowogrodzka 31, 00-511 Warsaw, Poland
andrzej.adgam@gmail.com
[2] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
skowron@mimuw.edu.pl, piotr@mimuw.edu.pl
[3] Systems Research Institute, Polish Academy of Sciences
Newelska 6, 01-447 Warsaw, Poland

Abstract. We discuss the rough set approach to approximation of vague concepts. There are already published several papers on rough sets and vague concepts staring from the seminal papers by Zdzisław Pawlak. However, only a few of them are discussing the relationships of rough sets with the sorites paradox. This paper contains a continuation of discussion on this issue.

Key words: vagueness, vague concept, sorites paradox, (adaptive) rough set.

1 Introduction

The rough set (RS) approach was proposed by Professor Zdzisław Pawlak in 1982 [35, 36, 40] as a tool for dealing with imperfect knowledge, in particular with vague concepts. Over the years many applications of methods based on rough set theory alone or in combination with other approaches have been developed.

The rough set approach seems to be of fundamental importance in artificial intelligence and cognitive sciences, especially in machine learning, data mining and knowledge discovery from databases, pattern recognition, decision support systems, expert systems, intelligent systems, multiagent systems, adaptive systems, autonomous systems, inductive reasoning, commonsense reasoning, adaptive judgement, conflict analysis.

Rough sets have established relationships with many other approaches such as fuzzy set theory, granular computing, evidence theory, formal concept analysis, (approximate) Boolean reasoning, multicriteria decision analysis, statistical methods, decision theory, matroids have been clarified. Despite the overlap with many other theories rough set theory may be considered as an independent discipline in its own right. There are reports on many hybrid methods obtained by combining rough sets with other approaches such as soft computing (fuzzy sets, neural networks, genetic algorithms), statistics, natural computing, mereology, principal component analysis, singular value decomposition or support vector machines.

In particular some relationships of the rough set approach with vague concepts were shown (see, *e.g.*, [2, 3, 5, 13, 29, 31, 32, 37, 38, 41, 45, 49–51, 55]). However, the relationships with sorites paradox are not explored well yet. In this paper, we extend a discussion on this topic, especially presented in [25].

Let us also note that the relationships with vague concepts of other approaches to uncertainty such as fuzzy sets or graded consequence are elaborated in the literature (see, *e.g.*, [7–13, 17, 28, 43, 44]).

This paper is structured as follows. In Sect. 2 we present some preliminaries on vague sets. Rudiments of rough sets, in particular approximations of concepts, are discusses in Sect. 3. The rough set approach to sorites paradox is presented in Sect. 4. The issue of higher order vagueness in rough sets is covered in Sect. 5. In Sect. 6, we present some constraints on induced classifiers for vague concepts related to the sorites paradox. They are making it possible to eliminate the contradiction characteristic to sorites paradox which is related to behaviour of the classifier when it passes through different approximation regions. Sect. 7 emphasizes the need for development the adaptive rough set approach.

2 Vague Sets

Mathematics requires that all mathematical notions (including set) must be exact, otherwise precise reasoning would be impossible. However, philosophers (see, *e.g.*, [26] and recently computer scientists as well as other researchers have become interested in *vague* (imprecise) concepts. Moreover, in the XX century one can observe the drift paradigms in modern science from dealing with precise concepts to vague concepts, especially in the case of complex systems (*e.g.*, in economy, biology, psychology, sociology, quantum mechanics).

Almost all concepts we are using in natural language are vague [1, 6]. Therefore, common sense reasoning based on natural language must be based on vague concepts and not on classical logic. Interesting discussion of this issue can be found in [45]. The idea of vagueness can be traced back to the ancient Greek philosopher Eubulides of Megara (ca. 400BC) who first formulated so called "sorites" (heap) and "falakros" (bald man) paradox (see, *e.g.*, [26]). There is a huge literature on issues related to vagueness and vague concepts in philosophy (see, *e.g.*, [4, 14, 19, 26, 27, 46–48]).

Vagueness is often associated with the boundary region approach (*i.e.*, existence of objects which cannot be uniquely classified relative to a set or its complement) which was first formulated in 1893 by the father of modern logic, German logician, Gottlob Frege (1848-1925) (see [15]). According to Frege (see Grundgesetze der Arithmetik, vol. ii, Sect.56 [15, 16]) the concept must have a sharp boundary:

> *To the concept without a sharp boundary there would correspond an area that would not have any sharp boundary – line all around.*

It means that mathematics must use crisp, not vague concepts, otherwise it would be impossible to reason precisely. However, vagueness in opinion of Ludwig Wittgenstein is an essential feature of language with semantics specified by

'language games'. A language is not a calculus with rigid rules that provide for all possible circumstances. There are many vague concepts in natural languages [1, 6]. One should also note that vagueness also relates to insufficient specificity, as the result of lack of feasible searching methods for sets of features adequately describing concepts.

Discussion on vague (imprecise) concepts in philosophy includes the following characteristic features of them [26]: (i) the presence of borderline cases, (ii) boundary regions of vague concepts are not crisp, (iii) vague concepts are susceptible to sorites paradox. In the sequel we discuss these issues in the RS framework. The reader can find the discussion on application of the RS approach to vagueness in [45].

3 Rough Set Based Concept Approximation

The starting point of rough set theory is the indiscernibility relation, which is generated by information about objects of interest (defined later in this section as signatures of objects). The indiscernibility relation expresses the fact that due to a lack of information (or knowledge) we are unable to discern some objects employing available information (or knowledge). This means that, in general, we are unable to deal with each particular object but we have to consider granules (clusters) of indiscernible objects as a fundamental basis for our theory.

¿From a practical point of view, it is better to define basic concepts of this theory in terms of data. Therefore we will start our considerations from a data set called an *information system*.

Suppose we are given a pair $\mathbb{A} = (U, A)$ of non-empty, finite sets U and A, where U is the *universe* of *objects*, and A – a set consisting of *attributes*, i.e. functions $a : U \longrightarrow V_a$, where V_a is the set of values of attribute a, called the *domain* of a. The pair $\mathbb{A} = (U, A)$ is called an *information system* (see, *e.g.*, [34]). Any information system can be represented by a data table with rows labeled by objects and columns labeled by attributes. Any pair (x, a), where $x \in U$ and $a \in A$ defines the table entry consisting of the value $a(x)$.

Any subset B of A determines a binary relation $\mathcal{IND}_B$ on U, called an *indiscernibility relation*, defined by

$$x \ \mathcal{IND}_B \ y \text{ if and only if } a(x) = a(y) \text{ for every } a \in B, \tag{1}$$

where $a(x)$ denotes the value of attribute a for object x.

Obviously, $\mathcal{IND}_B$ is an equivalence relation. The family of all equivalence classes of $\mathcal{IND}_B$, i.e., the partition determined by B, will be denoted by $U/\mathcal{IND}_B$, or simply U/B; an equivalence class of $\mathcal{IND}_B$, i.e., the block of the partition U/B, containing x will be denoted by $B(x)$ (other notation used: $[x]_B$ or more precisely $[x]_{\mathcal{IND}_B}$). Thus in view of the data we are unable, in general, to observe individual objects but we are forced to reason only about the accessible granules of knowledge (see, *e.g.*, [33, 36, 42]).

If $(x, y) \in \mathcal{IND}_B$ we will say that x and y are *B-indiscernible*. Equivalence classes of the relation $\mathcal{IND}_B$ (or blocks of the partition U/B) are referred to

as *B-elementary sets* or *B-elementary granules*. In the rough set approach the elementary sets are the basic building blocks (concepts) of our knowledge about reality. The unions of *B-elementary sets* are called *B-definable sets*.

For $B \subseteq A$ we denote by $Inf_B(x)$ the *B-signature* of $x \in U$, i.e., the set $\{(a, a(s)) : a \in B\}$. Let $INF(B) = \{Inf_B(s) : s \in U\}$. Then for any objects $x, y \in U$ the following equivalence holds: $x \mathcal{IND}_B y$ if and only if $Inf_B(x) = Inf_B(y)$.

The indiscernibility relation will be further used to define basic concepts of rough set theory. Let us define now the following two operations on sets $X \subseteq U$

$$\mathsf{LOW}_B(X) = \{x \in U : B(x) \subseteq X\}, \tag{2}$$

$$\mathsf{UPP}_B(X) = \{x \in U : B(x) \cap X \neq \varnothing\}, \tag{3}$$

assigning to every subset X of the universe U two sets $\mathsf{LOW}_B(X)$ and $\mathsf{UPP}_B(X)$ called the *B-lower* and the *B-upper approximation* of X, respectively. The set

$$\mathsf{BN}_B(X) = \mathsf{UPP}_B(X) - \mathsf{LOW}_B(X), \tag{4}$$

will be referred to as the *B-boundary region* of X.

If the boundary region of X is the empty set, i.e., $\mathsf{BN}_B(X) = \varnothing$, then the set X is *crisp* (*exact*) with respect to B; in the opposite case, i.e., if $\mathsf{BN}_B(X) \neq \varnothing$, the set X is referred to as *rough* (*inexact*) with respect to B. Thus any rough set, in contrast to a *crisp set*, has a non-empty boundary region.

Thus a set is *rough* (imprecise) if it has nonempty boundary region; otherwise the set is crisp (precise). This is exactly the idea of vagueness proposed by Frege.

Let us observe that the definition of rough sets refers to data (knowledge), and is *subjective*, in contrast to the definition of classical sets, which is in some sense an *objective* one.

Due to the granularity of knowledge, rough sets cannot be characterized by using available knowledge. Therefore with every rough set we associate two *crisp* sets, called *lower* and *upper approximation*. Intuitively, the lower approximation of a set consists of all elements that *surely* belong to the set, whereas the upper approximation of the set constitutes of all elements that *possibly* belong to the set, and the *boundary region* of the set consists of all elements that cannot be classified uniquely to the set or its complement, by employing available knowledge.

4 Approximations of Concepts and Sorites Paradox

Let us consider the *heap paradox*.

1. 10,000 grains of sand is a heap of sand.
2. 10,000 grains of sand is a heap of sand, then 9999 grains of sand is a heap of sand.
3. 9999 grains of sand is a heap of sand, then 9998 grains of sand is a heap of sand.

4. ...
5. *Conclusion*. 1 grain of sand is a heap of sand.

For a given set X by $card(X)$ we denote the cardinality of X. Let us consider the sequence of collections of grains of sand: $x_1, \ldots, x_i, x_{i+1}, \ldots, x_N$, such that $card(x_i) - card(x_{i+1}) = 1$ for $i = 1, \ldots, N-1$.

It is worthwhile mentioning that the concept of *heap* is vague. This concept may be perceived differently by different agents. Now let us consider an agent ag having a decision system $\mathbb{A} = (U, A, d)$, where $U \supseteq \{x_1, \ldots, x_i, x_{i+1}, \ldots, x_N\}$ is a family of collections of grains and A is a set of conditional attributes over U. The decision d assigns to each $x \in U$ the decision $d(x)$ equal to 1 if x is a heap and 0, otherwise. This decision is made, *e.g.*, by another agent ag_{dec} on the basis of some attributes (usually different from attributes from A). We denote by H the decision class $\{x \in U : d(x) = 1\}$ and by $-H$ the decision class $\{x \in U : d(x) = 0\}$. In particular, the decision d is assigned to each x_i from the considered sequence. The agent ag defines a partition of U using the the lower approximation of H, *i.e.*, $\mathsf{LOW}_A(H)$, the boundary region of H, *i.e.*, $\mathsf{BN}_A(H)$, and the lower approximation of $-H$, *i.e.*, $\mathsf{LOW}_A(-H)$.

In our example, we assume that $x_1 \in \mathsf{LOW}_A(H)$ and $x_N \in \mathsf{LOW}_A(-H)$.

By a *bounce* we understand any i such that one of the following conditions is satisfied: (i) $x_i \in \mathsf{LOW}_A(H)$ & $x_{i+1} \in \mathsf{BN}_A(H)$, (ii) $x_i \in \mathsf{LOW}_A(H)$ & $x_{i+1} \in \mathsf{LOW}_A(-H)$, (iii) $x_i \in \mathsf{BN}_A(H)$ & $x_{i+1} \in \mathsf{LOW}_A(-H)$.

Now, we explain why such *bounces* may occur.

Let us consider the first case. The two remaining cases are analogous. One could argue that we have a problem because there exists i such that $x_i \in \mathsf{LOW}_A(H)$ and $x_{i+1} \in \mathsf{BN}_A(H)$ but the difference between cardinalities x_i and x_{i+1} is negligible ($card(x_i) - card(x_{i+1}) = 1$) from the point of view of the concept *heap*. Observe that in the rough set approach the agent ag using the decision system $\mathbb{A}$ is perceiving objects (*i.e.*, in our example collections of grains) by means of attributes from A. Let us assume that $A = \{card\}$ and the conditional attribute $card$ assigns to any collection of grains $x \in U$ its cardinality. The decision d is taken by another agent ag_{dec} and it may be based, *e.g.*, on the basis of a *shape* of collection of grains. For example, $d(x) = 1$ if the shape of x is trapezoidal with sufficiently large ratio of the trapezoid hight to the length of the longest parallel sides of the trapezoid, and 0, otherwise. It may happen in U that the decision made by the agent ag_{dec} for all collections of grains from the elementary granule $A(x_i)$ (with the same cardinality, say n) are equal to 1, *i.e.*, all collections of grains from $A(x_i)$ have the relevant trapezoidal shape accepted by d as the positive examples of the concept *heap*. However, in case of $A(x_{i+1})$ (consisting of collections of grains with the same cardinality equal to $n-1$) there are in U collections x, y of grains such that $d(x) = 1$ (*i.e.*, $x \in H$) and $d(y) = 0$ (*i.e.*, $y \in -H$). This explains that the considered case of bounce is possible despite the fact that the difference between $card(x_i)$ and $card(x_{i+1})$ looks negligible from the point of view of the concept *heap*.

¿From the above considerations, we conclude that in general one can assume that for any i:

$$x_i \in \mathsf{LOW}_A(H) \text{ implies } x_{i+1} \in \mathsf{LOW}_A(H) \cup \mathsf{BN}_A(H) \cup \mathsf{LOW}_A(-H), \quad (5)$$

instead of

$$x_i \in \mathsf{LOW}_A(H) \text{ implies } x_{i+1} \in \mathsf{LOW}_A(H). \quad (6)$$

Analogous conditions may be formulated when we change the condition in the predecessor from the lower approximation of H, to the boundary region of H, or to the lower approximation of the complement of H.

Of course, in some cases some arguments of the alternative on the right hand side may be eliminated. For example, in some cases of decision table $\mathbb{A}$ of agent ag in Eq. 5 may be eliminated on the right had side of implication the third argument of the alternative.

5 Higher Order Vagueness and Rough Sets: Toward Adaptive Rough Sets

In [26], it is stressed that boundaries of vague concepts are not crisp. In the definition presented in this chapter, the notion of boundary region is defined as a crisp set $\mathsf{BN}_B(X)$. However, let us observe that this definition is relative to the subjective knowledge expressed by attributes from B. Different sources of information may use different sets of attributes for concept approximation. Hence, the boundary region can change when we consider these different views. Another reason for boundary change may be related to incomplete information about concepts. They are known only on samples of objects [18]. Hence, when new objects appear again the boundary region may change. ¿From the discussion in the literature it follows that vague concepts cannot be approximated with satisfactory quality by *static* constructs such as induced membership inclusion functions, approximations or models derived, *e.g.*, from a sample. Understanding of vague concepts can be only realized in a process in which the induced models are adaptively matching the concepts in dynamically changing environments. This conclusion seems to have important consequences for further development of rough set theory in combination with fuzzy sets and other soft computing paradigms for adaptive approximate reasoning. For further details the reader is referred, *e.g.*, to [49, 50, 56].

¿From the above considerations it follows that for dealing with higher order vagueness one should consider an extension of the rough set approach to the adaptive rough set approach. In this approach, approximations of a vague concept are considered over a family of decision systems $\{\mathbb{A}_t\}_{t\in T}$, where T is a set of indices, *e.g.*, time points. Hence, we obtain a family of the lower approximations, upper approximations and boundary regions of the considered vague concept which are changing, *e.g.*, over time (see Figure 1).

It is worthwhile mentioning that the elements of this family are obtained through interaction with the environment what is pointing to the necessity of

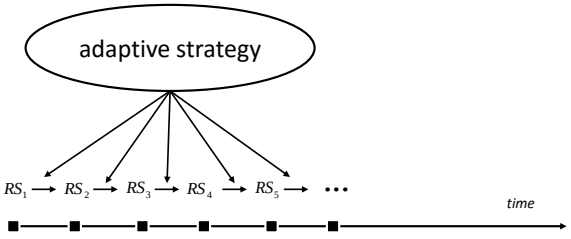


Fig. 1. Adaptive rough sets.

embedding the adaptive rough set approach in the framework of interactive granular computing and WisTech program (see, *e.g.*, [22–24, 21]).

6 Constraints on Bouncing Between Different Approximation Regions

If one would like to obtain some constraints on bouncing collections of sand grains between different approximation regions of the concept 'to be a heap of sand' assuming that succeeding collections are obtained by individually removing one grain from the preceding ones, then more details on rough set based approximations should be considered. For example, one may require that the changes of membership functions on consecutive collections of sand grains are below a given threshold. Let us consider an illustrative example to explain this issue in more detail.

First of all, one should note that usually information about approximated concept is partial, *e.g.*, provided by a sample of cases 'for' and 'against' a given concepts. Hence, in the rough set approach were developed methods for inductive extensions of approximation spaces from samples U of objects represented by decision systems on the universe U^* of all objects [39, 30].

In Figure 2 is presented a simple example of classifier for a concept $C \subseteq U^*$. The classifier is induced from a given partial information about C represented by a decision system $\mathbb{A}_d = (U, A, d)$ with the set of objects $U \subseteq U^*$ and the decision d equal (or almost equal) on U to the restriction to U of the characteristic function of C. The classifier represents an approximation of the characteristic function of the concept $C \subseteq U^*$.

The procedure of conflict resolution shown in Figure 2 between induced decision rules matching a given new case x (belonging to an extension U^* of U, *i.e.*, $U \subseteq U^*$) (and perceived as a signature of x, *i.e.*, $Inf_A(x)$) may be realized using arguments 'for' and 'against' membership in C determined by these rules. The arguments are aggregated using weights w_k as it is presented in Figure 3 what finally leads to the classifier computing the membership function μ_C for a given concept C.

Now, one can consider the membership function μ_{H^*} as an approximation of the characteristic function of the vague concept 'to be a heap of sand' $H^* \subseteq U^*$

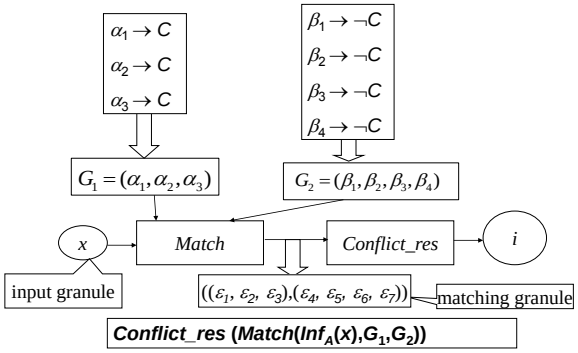


Fig. 2. Rough set-based rule classifier for a concept C (partially) specified by a decision system $\mathbb{A}_d = (U, A, d)$, where d is a characteristic function of a vague concept $C \subseteq U^*$ restricted to the sample $U \subseteq U^*$.

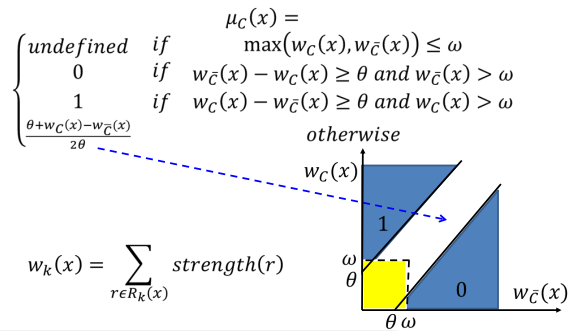


Fig. 3. Rough set based rule classifier for a concept C partially specified by a decision system, where Θ, ω are thresholds specified by the user, $strength(r)$ denotes the strength of the rule r (*e.g.*, defined by the support of the rule r [30]), and $R_k(x)$ denotes the set of decision rules induced from a given decision system for the decision $k \in \{C, \overline{C}\}$ ($\overline{C} = U^* \setminus C$) matching the case x.

induced (analogously as above) from a partial information represented by a decision system $\mathbb{A}_d = (U, A, d)$, where d is a characteristic function of a vague concept $H^* \subseteq U^*$ restricted to the sample $U \subseteq U^*$. We use μ_{H^*} in considerations concerning the paradox of heap of sand (Figure 4). Note that the induced approximations of the concept H^* are now defined as follows. The lower approximation of H^* is defined by $\mathsf{LOW}_A(H^*) = \{x \in U^* : \mu_{H^*}(x) = 1\}$, the upper approximation of H^* is defined by $\mathsf{UPP}_A(H^*) = \{x \in U^* : \mu_{H^*}(x) > 0 \vee \mu_{H^*}(x) = undefined\}$ and the boundary region of H^*: $\mathsf{BN}_A(H^*) = \mathsf{UPP}_A(H^*) \setminus \mathsf{LOW}_A(H^*)$.

In the considered example, we assume that the induced approximation of H^* represented by μ_{H^*} is consistent with a given sequence $x_1, \ldots, x_N$, *i.e.*, for any x_i and x_{i+1} representing consecutive collections of sand grains after individually removing one grain in each step, we have $\mu_{H^*}(x_i) = 1$ if $x_i \in H^*$ and

$\mu_{H^*}(x_i) = 0$ if $x_i \notin H^*$. In Figure 4 is presented a simple property of behavior of the model of H^* on elements of a sequence $x_1, \ldots, x_N$. If some additional constraints concerning weights are satisfied than one can see that the boundary region cannot be omitted. Moreover, using the assumption about a 'bounce size' of the membership values in passing from x_i to x_{i+1}, one can see that it can be necessary for the considered sequence to 'spend more time' in the boundary region before going out of it. One can specify such assumptions about 'bounce size' using the following constraints for induced classifiers: $w_C(x_i) - w_C(x_{i+1}) \leq \delta$ and $w_{\overline{C}}(x_{i+1}) - w_{\overline{C}}(x_i) \leq \delta$, where δ is a given threshold bounding bounces in degrees of memberships of x_i and x_{i+1}.

$x_1, \ldots, x_i, x_{i+1}, \ldots, x_N;\ \mu_{H^*}(x_1) = 1,\ \mu_{H^*}(x_N) = 0$

If

$x_i \in \mathrm{LOW}_A(H^*) \rightarrow x_{i+1} \in \mathrm{UPP}_A(H^*)$

$w_C(x_{i+1}) \leq w_C(x_i), w_{\overline{C}}(x_{i+1}) \geq w_{\overline{C}}(x_i)$

then

there exists $i_0 :\ x_{i_0} \in \mathrm{BN}_A(H^*)$

$x_i \in \mathrm{BN}_A(H^*) \rightarrow x_{i+1} \in \mathrm{UPP}_A(U^* \setminus H^*)$

$x_i \in \mathrm{LOW}_A(U^* \setminus H^*) \rightarrow x_{i+1} \in \mathrm{LOW}_A(U^* \setminus H^*)$

Fig. 4. Heap of sand - additional constraints on weights and their consequences.

7 Conclusions

We discussed the sorites paradox in the rough set approach. We have added a discussion on possible new constraints which should be added and preserved by approximations of vague concepts. These constraints are related to behavior of induced classifiers approximating vague concepts on sequences of objects considered in the sorites paradox for these vague concepts. We have also pointed the necessity of development of the adaptive rough set approach.

There are numerous logical approaches to vagueness (see, *e.g.*, [20, 52, 54]). However, from the above considerations it follows that adaptive logic based on rough sets can be relevant for the outlined approach. It is worthwhile mentioning here the following sentences from [53]:

> *Aristotle's man of practical wisdom, the phronimos, does not ignore rules and models, or dispense justice without criteria. He is observant of principles and, at the same time, open to their modification. He begins*

with nomoi established law and employs practical wisdom to determine how it should be applied in particular situations and when departures are warranted. Rules provide the guideposts for inquiry and critical reflection.

We plan to develop a logical approach to vagueness based on rough sets and adaptive judgement [21–24].

Acknowledgments. This work was partially supported by the Polish National Science Centre (NCN) grants DEC-2011/01/D /ST6/06981, DEC-2013/09/B/ST6/01568 as well as by the Polish National Centre for Research and Development (NCBiR) under the grant DZP/RID-I-44 / 8 /NCBR/2016.

References

1. G. P. Baker, P. M. S. Hacker (Eds.), *Wittgenstein: Understanding and Meaning: Volume 1 of an Analytical Commentary on the Philosophical Investigations, Part II: Exegesis §§1-184*, Blackwell, 2005. (2nd Edition).
2. M. Banerjee, M. K. Chakraborty, Foundations of vagueness: A category-theoretic approach, *Electronic Notes in Theoretical Computer Science* **82**(4) (2003) 10–19.
3. J. G. Bazan, A. Skowron, R. Swiniarski, Rough sets and vague concept approximation: From sample approximation to adaptive learning, *Transactions on Rough Sets* **V** (2006) 39–62.
4. M. Black, Vagueness: An exercise in logical analysis, *Philosophy of Science* **4**(4) (1937) 427–455.
5. Z. Bonikowski, U. Wybraniec-Skardowska, Vagueness and roughness. *Transactions on Rough Sets* **IX** (2008) 1–13.
6. L. C. Bums, *Vagueness. An Investigation into Natural Languages and the Sorites Paradox*, Kluwer, 1991.
7. D. Dubois, F. Esteva, L. Godo, H. Prade, An information-based discussion of vagueness, In: *Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Melbourne, Australia, December 2-5, 2001, IEEE*, IEEE Computer Science Press, 2001, 781–784.
8. D. Dubois, L. Godo, H. Prade, F.Esteva, An information-based discussion of vagueness, In: H. Cohen, C. Lefebvre (Eds.), *Handbook of Categorization in Cognitive Science*, Elsevier, Amsterdam, 2005, pp. 892–913.
9. D. Dubois, J. Lang, H. Prade, Handling uncertainty, context, vague predicates, and partial inconsistency in possibilistic logic, In: D. Driankov, P. W. Eklund, A. L. Ralescu (Eds.), *Fuzzy Logic and Fuzzy Control, IJCAI '91, Workshops on Fuzzy Logic and Fuzzy Control, Sydney, Australia, August 24, 1991, Proceedings*, LNCS **833**, Springer 1991, pp. 45–55.
10. D. Dubois, H. Prade, Modeling uncertain and vague knowledge in possibility and evidence theories, In: R. D. Shachter, T. S. Levitt, L. N. Kanal, J. F. Lemmer (Eds.), *UAI '88: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence, Minneapolis, MN, USA, July 10-12, 1988*, North-Holland, 1988, pp. 303–318.
11. D. Dubois, H. Prade, Fuzzy sets – a convenient fiction for modeling vagueness and possibility, *IEEE Transactions on Fuzzy Systems* **2**(1) (1994) 16–21.

12. D. Dubois, H. Prade, Modeling uncertain and vague knowledge in possibility and evidence theories, CoRR abs/1304.2349.
URL http://arxiv.org/abs/1304.2349
13. S. Dutta, S. Basu, M. K. Chakraborty, Many-valued logics, fuzzy logics and graded consequence: A comparative appraisal, In: K. Lodaya (Ed.), *Logic and Its Applications, 5th Indian Conference, ICLA 2013, Chennai, India, January 10-12, 2013, Proceedings*, Springer, LNCS **7750**, Springer, 2013, 197–209.
14. K. Fine, Vagueness, truth and logic, *Synthese* **30** (1975) 265–300.
15. G. Frege, *Grundgesetzen der Arithmetik* **2**, Verlag von Hermann Pohle, 1903.
16. P. Geach, M. Black (Eds.), *Translations from the Philosophical Writings of Gottlob Frege*, Blackwell, 1960.
17. J. A. Goguen, The logic of inexact concepts, *Synthese* **19** (1968-69) 325–373.
18. T. Hastie, R. Tibshirani, J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag, 2001.
19. C. G. Hempel, Vagueness and logic, *Philosophy of Science* **6** (1939) 163–180.
20. D. Hyde, Vagueness, Logic, and Ontology, Ashgate, 2008.
21. A. Jankowski, *Complex Systems Engineering: Wisdom for Saving Billions Based on Interactive Granular Computing* (in preparation), Springer, 2016
22. A. Jankowski, A. Skowron, A Wistech paradigm for intelligent systems, *Transactions on Rough Sets* **VI** (2007) 94–132.
23. A. Jankowski, A. Skowron, Wisdom technology: A rough-granular approach, In: M. Marciniak, A. Mykowiecka (Eds.), *Bolc Festschrift*, Springer, Heidelberg, LNCS **5070**, Springer, 2009, 3–41.
24. A. Jankowski, A. Skowron, R. W. Swiniarski, Interactive complex granules, *Fundamenta Informaticae* **133**(2-3) (2014) 181–196.
25. A. Jankowski, A. Skowron, Rough sets and vague concepts, *Analele Universitatii din Bucuresti. Seria Informatica* **LXII**(3) (2015) 119–133.
26. R. Keefe, *Theories of Vagueness*, Cambridge Studies in Philosophy, Cambridge, UK, 2000.
27. R. Keefe, P. Smith, *Vagueness: A Reader*, MIT Press, Massachusetts, MA, 1997.
28. J. Lawry, D. Dubois, A bipolar framework for combining beliefs about vague propositions, In: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of KR 2012, Rome, Italy, June 10-14, 2012*, AAAI Press, 2012.
29. S. Marcus, The paradox of the heap of grains, in respect to roughness, fuzziness and negligibility, In: L. Polkowski, A. Skowron (Eds.), *Proceedings of RSCTC'1998, Warsaw, June 22-26, 1998*, LNAI **1424**, Springer-Verlag, 1998, 19–23.
30. H.S. Nguyen, A. Skowron, Rough Sets: From Rudiments to Challenges. In: A. Skowron, Z. Suraj (eds.), *Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam*, Intelligent Systems Reference Library **42**, Springer 2013, 75-173.
31. E. Orłowska, Semantics of vague concepts, In: G. Dorn, P. Weingartner (Eds.), *Foundation of Logic and Linguistics*, Plenum Press, 1984, 465–482.
32. E. Orłowska, Reasoning about vague concepts, *Bulletin of the Polish Academy of Sciences, Mathematics* **35** (1987) 643–652.
33. S. K. Pal, L. Polkowski, A. Skowron (Eds.), *Rough-Neural Computing: Techniques for Computing with Words*, Cognitive Technologies, Springer-Verlag, 2004.
34. Z. Pawlak, Information systems - Theoretical foundations, *Information Systems* **6** (1981) 205–218.
35. Z. Pawlak, Rough sets, *International Journal of Computer and Information Sciences* **11** (1982) 341–356.

36. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data, System Theory*, Knowledge Engineering and Problem Solving **9**, Kluwer Academic Publishers, 1991.
37. Z. Pawlak, Vagueness and uncertainty: A rough set perspective, *Computational Intelligence: An International Journal* **11** (1995) 217–232.
38. Z. Pawlak, Vagueness - A rough set view. In: J. Mycielski, G. Rozenberg, A. Salomaa (Eds.), *Structures in Logic and Computer Science*, Springer, LNCS **1261**, Springer, 1997, 106–117.
39. Z. Pawlak, A. Skowron, Rough sets: Some extensions, *Information Sciences* **177**(1) (2007) 28–40.
40. Z. Pawlak, A. Skowron, Rudiments of rough sets, *Information Sciences* **177**(1) (2007) 3–27.
41. L. Polkowski, M. Semeniuk-Polkowska: Boundries, borders, fences, hedges. *Fundamenta Informaticae* **129**(1-2) (2014) 149–159.
42. L. Polkowski, A. Skowron, Rough mereological calculi of granules: A rough set approach to computation, *Computational Intelligence: An International Journal* **17**(3) (2001) 472–492.
43. H. Prade, A two-layer fuzzy pattern matching procedure for the evaluation of conditions involving vague quantifiers, *Journal of Intelligent and Robotic Systems* **3**(2) (1990) 93–101.
44. H. Prade, C. Testemale, Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries, *Information Sciences* **34**(2) (1984) 115–143.
45. S. Read, *Thinking about Logic: An Introduction to the Philosophy of Logic*, Oxford University Press, 1994.
46. G. Ronzitti (Ed.), *Vagueness: A Guide*, Logic, Epistemology, and The Unity of Science Series, vol. 19, Springer, 2011.
47. B. Russell, Vagueness, *The Australian Journal of Psychology and Philosophy* **1** (1923) 84–92.
48. S. Shapiro (Ed.), *Vagueness in Context*, Clarendon Press, 2006.
49. A. Skowron, Rough sets and vague concepts, *Fundamenta Informaticae* **64**(1-4) (2005) 417–431.
50. A. Skowron, R. Swiniarski, Rough sets and higher order vagueness, In: D. Ślęzak, G. Wang, M. Szczuka, I. Düntsch, Y. Yao (Eds.), *Proceedings of RSFDGrC'2005, Regina, Canada, August 31-September 3, 2005*, Part I, LNAI **3641**, Springer-Verlag, 2005, 33–42.
51. D. Ślęzak, P. Wasilewski, Foundations of rough sets from vagueness perspective, In: A. E. Hassanien, Z. Suraj, D. Ślęzak, P. Lingras (Eds.), *Rough Computing: Theories, Technologies and Applications*, IGI Global, 2007, 1–37.
52. K. Tanaka, F. Berto, E. Mares, F. Paoli (Eds.), Paraconsistency: Logic and Applications, Springer, 2013.
53. L. P. Thiele, The Heart of Judgment: Practical Wisdom, Neuroscience, and Narrative, Cambridge University Press, 2010.
54. P. Verdée, S. van der Waart van Gulik, A generic framework for adaptive vague logics, *Studia Logica* **90** (2008) 385–405.
55. M. Wolski, Science and semantics: A note on vagueness, In: A. Skowron, Z. Suraj (Eds.), *Rough Sets and Intelligent Systems. Professor Zdzisław Pawlak in Memoriam*, Series Intelligent Systems Reference Library **43**, Springer 2013, 623–643.
56. Q. Zhang, J. Wang, G. Wang, H. Yu, The approximation set of a vague set in rough approximation space, *Information Sciences* **300** (2015) 1–19.

A version of rough mereology suitable for rough sets

Lech T. Polkowski

Polish-Japanese Academy IT
Koszykowa str. 86, 02-008 Warszawa, Poland
email: lech.polkowski@pja.edu.pl;polkow@pjwstk.edu.pl

Abstract. We address in principle the notion of a boundary and we propose a version of mereology better adapted to rough set theory than the original version. We discuss the motivation and differences between the original and proposed now versions of rough mereology and we show that the Pawlak notion of a boundary in rough set theory is a particular case of the more general notion of a boundary in the rough mereological theory proposed in this work.

Keywords: rough set theory, rough mereology, truly rough inclusion, boundary

1 Introduction: the idea of a boundary, in general and in Pawlak's theory

It is evident to all who study rough set idea that the most important notion is the notion of a boundary and most important things that conform to that notion are boundaries of concepts as they witness the uncertainty of the concept. The notion of a boundary has been the subject of investigation by philosophers, logicians, topologists from ancient times to now. The basic problem with the notion of a boundary stems of course from the fact that our perception of the world is continuous whereas the world has a discrete structure. Whence follow the philosophical dilemmas like Bolzano's paradox of the two touching balls A and B. The question is where A ends and B begins? If q is the last point on A which must exists by closedness of A, then if p is the first point on B and p is not q then A and B do not touch and there is no boundary between them but we know they touch as we are not able to push A or B any further. Similar are problems by Leonardo of air and water: what is the boundary between water in the river and air? Those and other similar questions have occupied philosophers since times of antiquity. One needs not to reach for real physical phenomena in order to find problems and difficulties with boundaries, e.g., time imagined continuity have caused similar problems: what is the last moment an event occurs? See Varzi [9] and Smith [8] for a discussion of philosophical aspects of the notion of a boundary.

Mathematicians resolved the problem by postulating the completeness of the real line which cannot be dissected into two open disjoint non–empty sets; returning to philosophical aspects of the notion of a boundary, this implies that when an event has the last moment p in which it happens, we are not able to point to the first moment in which it does not happen. They also approached the problem of a boundary from a local point of view with the idea of a neighborhood and closeness: the boundary of a set is the set of points 'infinitely close' to the set in the sense that each neighborhood of a boundary point must needs intersect the set and its complement, see Engelking [1].

This point of view prevails in Pawlak's idea of a boundary see Pawlak [3] but the implementation of it proceeds in a distinct way. In the first place, one needs the data about reality in the form of an *information system* i.e. a tuple (U, A, val, V) where U is a set of *objects* representing physical entities, processes, moments of time etc., A is a set of *attributes* each of which maps objects in U into the *set of values* V by means of a mapping $val : U \times A \to V$. The value $val(u, a)$ is often written down in a simpler form of $a(u)$.

Objects are then coded as *information sets* of the form $Inf(u) = \{a(u) : a \in A\}$. The crux of the rough set approach is in identification of objects having the same information sets: $Ind(u, v)$ if and only if $Inf(u) = Inf(v)$, where $Ind(u, v)$ is the *indiscernibility relation* . From that moment on, objects loose their real names and become visible only by their information sets which allows for making some of them indiscernible. The equivalence relation of indiscernibility partitions the universe U into classes which are also regarded as primitive *granules of knowledge* $[u]$ or *black boxes*.

One addresses the problem of *concepts* understood as subsets of the universe U and distinguishes among them *certain* ones as those which can be assembled from granules by the union of sets operator i.e. a concept $C \subseteq U$ is certain if and only if $C = \bigcup\{[u] : [u] \subseteq C\}$. Other concepts are not certain, commonly called *rough*. A rough concept R then must have an object u such that the class $[u]$ is not contained in it but it does intersect the complement $U \setminus R$. Such objects in R constitute the *boundary* of R, BdR, i.e. $BdR = \{u \in U : [u] \cap (U \setminus R) \neq \emptyset \neq [u] \cap R\}$. One may say that the boundary of R consists of objects which have their copies both in R and in $U \setminus R$. This is clearly a topological approach as the class $[u]$ is the least neighborhood of u in the partition topology induced by the indiscernibility relation Ind.

Let us point to advantages of this approach: first it is objective as the shape of the boundary follows by the data without any intervention of subjective factors which are so essential in e.g. fuzzy set theory see Zadeh [12]; next, it relies solely on data without resorting to e.g. real numbers or other auxiliary external to data factors.

2 Mereology and its extensions into domain of uncertainty

Mereology due to Stanisław Leśniewski, see Leśniewski [2], Sinisi [7], is based on the notion of a part relation, $part(x, y)$ (*'x is a part to y'*) which satisfies over a universe U conditions:

M1 For each $x \in U$ it is not true that $part(x, x)$;

M2 For each triple x, y, z of things in U if $part(x, y)$ and $part(y, z)$, then $part(x, z)$.

The notion of an *element* is defined as the relation $el(x, y)$ which holds true if $part(x, y)$ or $x = y$. It follows that the relation of being an element is a partial order on U and $el(x, y)$ and $el(y, x)$ are true simultaneously if and only if $x = y$. Clearly, $part(x, y)$ if and only if $el(x, y)$ and $x \neq y$.

The last important notion is that of the *class* understood as the object which represents a collective entity i.e. a *property*: for a property Ψ on U which is not void, the class pf Ψ, $Cls\Psi$, is the object such that:

C1 If $\Psi(u)$ holds true, then $el(u, Cls\Psi)$;

C2 For each u, if $el(u, Cls\Psi)$ then there are objects p, q such that $el(p, u)$, $el(p, q)$, $\Psi(q)$ hold true.

2.1 Our rough mereology as up to now

We proposed a scheme which extended mereology based on the part relation to the version based on the *'part to a degree'* relation, see Polkowski [6], written down as the relation $\mu(x, y, r)$ (*x is a part of y to a degree of at least of r*) on the universe U endowed with the mereological notion of the element el. The assumptions abut μ reflected the basic true properties of the partial containment:

RM1 $\mu(x, x, 1)$ for each $x \in U$;

RM2 $\mu(x, y, 1)$ if and only if $el(x, y)$ holds true;

RM3 If $\mu(x, y, 1)$ and $\mu(z, x, r)$ hold true then $\mu(z, y, r)$ holds true;

RM4 If $\mu(x, y, r)$ and $s < r$ hold true then $\mu(x, y, s)$ holds true.

This approach has required the a priori mereology on the universe U of which the relation μ was a diffusion or fuzzification, cf. Varzi [10]. Assume that $f : [0, 1]^2 \to [0, 1]$ is a continuous in each coordinate and symmetric function such that $f(1, r) = r$ for each $r \in [0, 1]$. We will call any such f a *pre–norm*.

We say for a pre–norm f that relation μ is f–transitive if and only if from true conditions $\mu(x, y, r)$ and $\mu(y, z, s)$ the true condition $\mu(x, z, f(r, s))$ follows.

Proposition 1. *If the relation μ on the universe U is f–transitive then the relation $\pi(x, y)$ which holds true if and only if $\mu(x, y, 1)$ and $\mu(y, x, 1)$ hold true is an equivalence relation on U.*

Indeed, $\pi(x, x1)$ holds true by RM1; symmetry is evident by definition; transitivity follows by $f-$transitivity of μ.

An attempt to apply this version of μ in the rough set context is handicapped by the fact that by RM2, the relation π should be the identity which does not capture the full scope of rough set cases. We need something more flexible to accomodate equivalence relations of indiscernibility.

2.2 A rough mereology for rough set theory

As stated below, we need a mereology which may account for rough set theoretic contexts. We assume an information system (U, A, val, V) as our playground.

We define a *truly rough inclusion*, $\mu_{tr}(x, y, r)$ as a relation which satisfies on U the following assumptions:

TRM1 $\mu_{tr}(x, x, 1)$;

TRM2 There is a partition P on U such that $\mu_{tr}(x, y, 1)$ if and only if x and y are in the same partition class $[x]_P$;

TRM3 If $\mu_{tr}(x, y, 1)$ and $\mu_{tr}(z, x, r)$ then $\mu_{tr}(z, y, r)$;

TRM4 If $\mu_{tr}(x, y, r)$ and $s < r$ then $\mu_{tr}(x, y, s)$.

TRM5 The truly rough inclusion $\mu_{tr}(x, y, r)$ is f–transitive for some pre–norm f.

The predicate $el(x, y)$ if $\mu_{tr}(x, y, 1)$ defines x as an *element of* y.

We sum up basic consequences of our assumptions.

Proposition 2. *The following are true by conditions TRM1-TRM5.*

1. $\mu_{tr}(x, y, 1)$ *implies* $\mu_{tr}(y, x, 1)$ *i.e.* μ_{tr} *is symmetric.*

2. The relation $el(x, y)$ *is symmetric and* $el(x, y)$ *and* $el(y, x)$ *imply that* $[x]_P = [y]_P$.

3. $\{y : \mu_{tr}(y, x, 1)\}$=$[x]_P$.

4. The relation $\mu_{tr}(x, y, 1)$ *is transitive in the sense that* $\mu_{tr}(z, x, 1)$ *and* $\mu_{tr}(x, y, 1)$ *imply* $\mu_{tr}(z, y, 1)$.

3 Boundaries in rough mereology for rough sets

We use the language of predicates on the universe U in definitions of boundaries by means of a truly rough inclusions μ_{tr}.

3.1 A general scheme for boundaries

For a truly rough inclusion μ_{tr}, and $x \in U$, $r \in [0, 1]$, we define a new predicate $N(x, r)(z)$ if there exists an $s \geq r$ such that $\mu(z, x, s)$. $N(x, r)$ is the *neighborhood granular predicate about* x *of radius* r.

Consider a predicate Ψ on U having a non–empty meaning $[\Psi]$. The *complement to* Ψ is the predicate $-\Psi$ such that $-\Psi(x)$ if and only if not $\Psi(x)$. We define the *upper extension of* Ψ *of radius* r, denoted Ψ_r^+ by letting $\Psi_r^+(x)$ if there exists z such that $\Psi(z)$ and $N(x,r)(z)$. Similarly, we define the *lower restriction of* Ψ *of radius* r, denoted Ψ_r^- by letting $\Psi_r^-(x)$ if and only if not $(-\Psi)_r^+(x)$.

Proposition 3. *1. Predicates* Ψ_r^+ *and* Ψ_r^- *are disjoint in the sense that there is no* $z \in U$ *such that* $\Psi_r^+(z)$ *and* $\Psi_r^-(z)$ *hold true. 2. If* $\Psi_r^+(x)$ *holds true then* $\Psi_r^+(y)$ *holds true for each* y *such that* $\mu_{tr}(y,x,1)$*. 3. If* $\Psi_r^-(x)$ *holds true then* $\Psi_r^-(y)$ *holds true for each* y *such that* $\mu_{tr}(y,x,1)$*.*

Proof. Claim 1 follows by definitions of the two predicates. For Claim 2, consider x, y such that $\Psi_r^+(x)$ and $\mu_{tr}(y,x,1)$. There exists z such that $\Psi(z)$, $N(x,s)(z)$ hold true with some $s \geq r$ so $\mu_{tr}(z,x,s)$ holds true. By symmetry of μ_{tr}, we have $\mu_{tr}(x,y,1)$ true and f–transitivity of μ_{tr} for an adequate pre–norm f implies that $\mu_{tr}(z,y,f(1,s))$ holds true i.e. $\mu_{tr}(z,y,s)$ holds true which means that $N(y,r)(z)$ holds true and finally $\Psi_r^+(y)$ holds true. For Claim 3, assume that $\Psi_r^-(x)$ and $\mu_{tr}(y,x,1)$ hold true i.e.

$$\neg\exists z, s \geq r.\mu_{tr}(z,x,s) \wedge \neg\Psi(z) \tag{1}$$

which is equvalent to

$$\mu_{tr}(z,x,s) \rightarrow \Psi(z). \tag{2}$$

As $\mu_{tr}(y,x,1)$ is equivalent to $\mu_{tr}(x,y,1)$, we have by f–transitivity of μ_{tr} that

$$\mu_{tr}(z,y,s) \rightarrow \Psi(z), \tag{3}$$

which is equivalent to the thesis $\Psi_r^-(y)$.

We will say that a predicate Ψ is *el–saturated* if and only if true formulas $\Psi(x)$ and $el(y,x)$ imply that $\Psi(y)$. A corollary to Claim 3 in Proposition 3 says that for each $r \in [0,1]$, predicates Ψ_r^+ and Ψ_r^- are el–saturated.

A global and local definition of the boundary For a predicate Ψ, we define the predicate *boundary of* Ψ *with respect to a truly rough inclusion* μ_{tr}, denoted $Bd_{\mu_{tr}}\Psi$ as follows:

$$Bd_{\mu_{tr}}\Psi \leftrightarrow (\neg\Psi_1^+) \wedge (\neg\Psi_1^-). \tag{4}$$

Arguing like in proof of Proposition 3, we prove the following

Proposition 4. *1.* $Bd_{\mu_{tr}}\Psi$ *is el–saturated 2. For no* $z \in U$*,* $Bd_{\mu_{tr}}\Psi(z) \wedge \Psi_1^+(z)$ *is true and for no* $z \in U$*,* $Bd_{\mu_{tr}}\Psi(z) \wedge \Psi_1^-(z)$ *is true.*

Proposition 5. *For each* $x \in U$*,* $Bd_{\mu_{tr}}\Psi(x)$ *holds true if and only if there exist* $z, y \in U$ *such that* $\Psi(z)$*,* $-\Psi(y)$*,* $\mu_{tr}(z,x,1)$*,* $\mu_{tr}(y,x,1)$*.*

A predicate *Open* is defined on predicates on U and a predicate Φ on U is *open*, $Open(\Phi)$ in symbols if and only if it is el–saturated.

Corollary 1. *$Open(\Psi_r^+)$ and $Open(\Psi_r^-)$ hold true for each $r \in [0,1]$.*
$Open(Bd_{\mu_{tr}}\Psi)$ holds true.

Proposition 6. *For a finite collection of predicates $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ if $Open(\Psi_i)$ holds true for each $i \leq k$, then $Open(\bigvee_i \Psi_i)$ holds true.*

A predicate *Closed* holds true for a predicate Ψ if and only if $Open(-\Psi)$ holds true.

Corollary 2. *$Closed(\Psi_r^+)$ and $Closed(\Psi_r^-)$ hold true for each $r \in [0,1]$.*
$Closed(Bd_{\mu_{tr}}\Psi)$ holds true.

For the mereotopological notion of boundary see also Polkowski and Semeniuk–Polkowska [4], [5] and Varzi [11].

3.2 The Pawlak notion of a boundary is a special case of truly rough mereological notion of a boundary

We return to an information system (U, A, val, V). We derive a truly rough inclusion from any Archimedean t–norm. There exist two non–isomorphic Archimedean t–norms:

- the Łukasiewicz t–norm $L(x,y) = max\{0, x+y-1\}$;
- the product t–norm $P(x,y) = x \cdot y$.

Both these t–norms admit a Hilbert–style representation

$$t(x,y) = g(f(x) + f(y)),$$

where $f : [0,1] \to [0,1]$ is a continuous decreasing function with $f(0) = 1$, and $g : [0,1] \to [0,1]$ is the inverse to f. In case of the t–norm L, $f(x) = 1 - x$ and $g(y) = 1 - y$. We let for an Archimedean t–norm t:

$$\mu^t(x,y,r) \text{ if and only if } g(\frac{card(Dis(x,y))}{card(A)}) \geq r, \tag{5}$$

where $Dis(x,y) = \{a \in A : a(x) \neq a(y)\}$.

In particular, as for the Łukasiewicz t–norm we have $g(y) = 1 - y$, the *Lukasiewicz truly rough inclusion* can be defined as

$$\mu^L_{tr}(x,y,r) \text{ if and only if } \frac{card(Ind(x,y))}{card)A)} \geq r, \tag{6}$$

where $Ind(x,y) = A \setminus Dis(x,y)$. In particular, μ^L is L–transitive.

The predicate of element $el(x,y)$ holds true if and only if $\mu^L_{tr}(x,y,1)$ holds true if and only if $Ind(x,y)$ i.e. x, y are indiscernible. Hence, a predicate is el–saturated if and only if its meaning is the union of a family of indiscernibility classes and rough mereological notions of Ψ_1^+ and Ψ_1^- become, respectively, the notions of the upper and the lower approximations of the meaning of Ψ and the meaning of the boundary predicate $Bd_{\mu^L}\Psi$ is the boundary of the meaning of Ψ.

4 Conclusions

We have proposed a new version of rough mereology suitable for rough set theory and we show that the rough set theory is a particular case of an abstract rough mereotopological theory.

5 Acknowledgement

The author would like to dedicate this work to the memory of Professor Zdzisław Pawlak (1926-2006).

References

1. Engelking, R.: General Topology. Heldermann Verlag, Berlin, 1989.
2. Leśniewski, S.: Podstawy Oglnej Teoryi Mnogoci, I. (Foundations of General SetTheory,I, in Polish). Prace Polskiego Koa Naukowego w Moskwie. Sekcja Matematyczno-Przyrodnicza, 1916, nr.2.
3. Pawlak, Z.: Rough Sets. Theoretical aspects of Reasoning about Data. Kluwer, Dordrecht, 1991.
4. Polkowski, L., Semeniuk-Polkowska, M.: Granular Mereotopology: A First Sketch. http://ceur-ws.org/Vol-1032/paper-28.pdf
5. Polkowski, L., Semeniuk-Polkowska, M.: Boundaries, borders, fences, hedges. Fundamenta Informaticae - Dedicated to the Memory of Professor Manfred Kudlek 129(1-2), 2014, 149-159.
6. Polkowski, L.: Approximate Reasoning by Parts. An Introduction to Rough Mereology. ISRL Series No. 20. Springer International Switzerland, Cham, 2012.
7. Sinisi, V.: Leśniewski's Foundations of Mathematics. Topoi 2 (1), 1983, 3-52.
8. Smith, B.: Boundaries: An essay in mereotopology. In: Hahn, L. (ed.): The Philosophy of Roderick Chisholm (Library of Living Philosophers). La Salle: Open Court, 1997, 534-561.
9. Varzi, A. C.: Boundary. In Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/boundary/
10. Varzi, A. C.: Mereology. In Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/mereology/
11. Varzi, A. C.: Basic problems of mereotopology. In: Guarino, N. (ed.): Formal Ontology in Information Systems. IOS Press, Amsterdam, 1998, 29-38.
12. Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338-353, 1965.

Rough Set Based Approximations of Classes in the OWL Ontology of Places in Poland

Extended Abstract

Piotr Grochowalski[1], Krzysztof Pancerz[1] and Tomasz Szul[2]

[1] Chair of Computer Science, Faculty of Mathematics and Natural Sciences
University of Rzeszów
Prof. S. Pigonia Str. 1, 35-310 Rzeszów, Poland
{piotrg,kpancerz}@ur.edu.pl
[2] Department of Power Engineering and Automation of Agricultural Processes
Faculty of Production and Power Engineering
University of Agriculture in Krakow, Poland
Balicka Str. 116B, 30-149 Kraków
tomasz.szul@ur.krakow.pl

Abstract. The main goal of our research is to build the ontology of places in Poland covering a variety of aspects of places, mainly administrative and socio-economic. The ontology is being implemented using the OWL 2 Web Ontology Language. In the created OWL ontology, we can distinguish two kinds of classes, primary classes exactly defined in the ontology as well as secondary classes defined over the ontology on the basis of primary classes and properties of individuals considered in the ontology. We show how to use rough sets to approximate secondary classes by means of primary classes in the created ontology. Rough set approximations enable us to extract some useful knowledge about places.

Keywords: rough sets, approximation, ontology, OWL 2.

1 Ontologies and Semantic Relations

Ontologies, as formal representations of knowledge, have recently gained a significant popularity. They are currently used in knowledge engineering and data mining to capture knowledge about some domain of interest. Two reasons seem to be the main source of this popularity. Firstly, there exist well-defined standards of languages for the ontology representation. Secondly, ontologies cover various semantic aspects of information which are useful in data mining processes.

One of the key decisions to take in the ontology development process is to select the language in which the ontology will be implemented. Our ontology of places is built in accordance with the OWL 2 Web Ontology Language (shortly OWL 2). OWL 2 is the most recent development in standard languages defined by the World Wide Web Consortium (W3C) [2]. An OWL ontology consists of three components: classes, individuals, and properties. Classes are representations of concepts in a given domain of interest. Classes are interpreted as sets

that contain individuals. Individuals (also known as instances) represent objects in the domain of interest. Individuals can be referred to as being instances of classes. Properties (also known as roles or attributes) are binary relations on individuals. Properties link two individuals together. There are two main types of properties in OWL 2: object properties linking an individual to an individual and data properties linking an individual to a data value.

Semantic relations are very important components of ontologies as they describe the relationships that can be established between concepts. In the presented approach to rough set based approximations of classes in the OWL ontology, we are interested in the INSTANCE-OF relation as well as specific semantic relations describing relationships covering economic and social aspects of places. Such relations are represented in the OWL ontology by object and data properties. If i INSTANCE-OF c holds, it means that the individual i is an instance (example) of a given class c. It is worth recalling that i is also an instance of all superclasses of c.

2 The OWL Ontology of Places in Poland

In [9], we showed selected parts of the ontology of places in Poland (defined classes and class hierarchies, identified individuals, identified properties linking individuals). The ontology is being implemented using the OWL 2 Web Ontology Language. In general, the main goal of our research is to build the ontology of places in Poland covering a variety of aspects of places, mainly administrative and socio-economic. In the next section, we show how to use rough sets to approximate secondary classes by means of primary classes in the created ontology of places. Rough set approximations enable us to extract some useful knowledge about places. The ontology built by us can be used in various socio-economic research as the knowledge base. Moreover, it may constitute the basis for search engines and other computer tools used in the real-estate market.

3 Approximations of Classes in OWL Ontologies

Rough sets proposed by Z. Pawlak [10] are an appropriate tool to deal with rough (ambiguous, imprecise) concepts in the universe of discourse. There are various approaches to applying rough sets for a representation of vague knowledge and reasoning over it in ontologies (e.g., [4], [6], [7]). For example, in [4], a rough set approach to vague concept approximations was presented. The concept approximations were constructed on the basis of data sets (decision tables with condition attributes representing, e.g., sensory measurements) and an additional domain knowledge (the so-called concept ontology) using approximate reasoning schemes. In the current paper, we consider a situation where the whole knowledge is included in a domain ontology (implemented using the OWL 2 Web Ontology Language). Following the approach presented in [7], we propose to apply rough sets to approximate secondary classes by means of primary classes in the created OWL ontology of places.

Let us consider, as an example, a part of the ontology of places in Poland devoted to administrative districts. In our ontology, we have distinguished three administrative types of communes:

- urban commune,
- rural commune
- urban-rural commune.

Moreover, according to [3], where functional structures of communes in Poland were considered, we have distinguished eight basic functional types of communes:

- urban commune,
- urbanized commune,
- multifunctional transitional commune,
- overwhelmingly agricultural commune,
- prevalently agricultural commune,
- tourism and recreational function commune,
- forestry function commune,
- mixed function commune.

In our ontology of places, all of the types of communes shown above are represented by primary classes, i.e., classes exactly defined in the ontology.

Some of the socio-economic aspects of places considered in our ontology are issues related to waste management. They are expressed especially by means of data properties of individuals, for example, the rate $MAHW$ of mass accumulation of household waste. Using this rate, we can define a secondary class representing the concept "administrative district with the rate of mass accumulation of household waste greater than or equal to 100 $\frac{kg}{person \cdot year}$". One can see that such a class is not exactly defined in the ontology, but it can be derived from the primary class and one of the properties of individuals.

The semantics of the OWL 2 Web Ontology Language is complex (see [1]). Therefore, we omit the formal description of the considered problem of rough set based approximations of classes in the OWL ontology and give only its brief review, rather informal. Let C be a set of classes and I be a set of individuals in a given OWL ontology $\mathcal{O}$. For a given class $c \in C$, we consider a set $INST(c)$ of all individuals from I that are instances of c.

Analogously to rough approximation of sets defined in rough set theory [10], we can define rough approximation of a given secondary class c^* by means of primary classes. The lower approximation $lower(c^*)$ of c^* is given by:

$$lower(c^*) = \{c \in C : \underset{i \in INST(c)}{\forall} i \in INST(c^*)\}.$$

The lower approximation of a secondary class c^* consists of each primary class c such that all individuals being instances of c are also instances of c^*.

The upper approximation $upper(c^*)$ of c^* is given by:

$$upper(c^*) = \{c \in C : \underset{i \in INST(c)}{\exists} i \in INST(c^*)\}.$$

The upper approximation of a secondary class c^* consists of each primary class c such that there exists at least one individual being an instance of c which is also an instance of c^*.

Conventionally, the boundary region $bound(c^*)$ is defined as:

$$bound(c^*) = upper(c^*) - lower(c^*).$$

Let us consider a part of individuals in our ontology which are communes in the Lubelskie Voivodship. In this voivodship, we have 213 communes distributed into administrative types of communes as follows: 20 urban communes, 169 rural communes, and 24 urban-rural communes. In case of functional types of communes, we have the following distribution: 20 urban communes, 5 urbanized communes, 7 multifunctional transitional communes, 57 overwhelmingly agricultural communes, 102 prevalently agricultural communes, 11 tourism and recreational function communes, 2 forestry function communes, and 9 mixed function communes.

On the basis of data included in Tables 1 and 2, we obtain the following approximations of the class representing the concept "administrative district with the rate of mass accumulation of household waste greater than or equal to 100 $\frac{kg}{person \cdot year}$":

- the lower approximation consists of a class representing the concept "urban commune" (as a functional type), only,
- the upper approximation consists of classes representing the concepts "urban commune" (as an administrative type), "rural commune", "urban-rural commune", "urban commune" (as a functional type), "urbanized commune", "multifunctional transitional commune", "prevalently agricultural commune", "tourism and recreational function commune", and "mixed function commune".

Table 1. Results of approximation for administrative types of communes

Administrative type	#Communes with $MAHW \geq 100$	#Communes with $MAHW < 100$
urban	19	1
rural	11	158
urban-rural communes	8	16

The obtained approximations enable us, for example, to make the following generalizations for communes in the Lubelskie Voivodship:

- an urban commune (as a functional type) is an administrative district with the rate of mass accumulation of household waste greater than or equal to 100 $\frac{kg}{person \cdot year}$ in the Lubelskie Voivodship (according to the lower approximation),

Table 2. Results of approximation for functional types of communes

Functional type	#Communes with $MAHW \geq 100$	#Communes with $MAHW < 100$
urban	20	0
urbanised	4	1
multifunctional transitional	2	5
overwhelmingly agricultural	0	57
prevalently agricultural	4	98
tourism and recreational function	7	4
forestry function	0	2
mixed function	1	8

– an urbanised commune may be an administrative district with the rate of mass accumulation of household waste greater than or equal to 100 $\frac{kg}{person \cdot year}$ in the Lubelskie Voivodship (according to the boundary region).

Such generalizations are useful knowledge derived from the ontology of places. One can see that the presented approach can be used in search engines for ontologies.

A valuable way of developing further research is to consider various approaches for determining approximations, for example, the Variable Precision Rough Set Model (VPRSM) [11] or those based on combined rough sets and fuzzy sets (cf. [5]).

References

1. OWL 2 Web Ontology Language direct semantics. W3C Recommendation. Tech. rep., W3C (2012)
2. OWL 2 Web Ontology Language structural specification and functional-style syntax. W3C Recommendation. Tech. rep., W3C (2012)
3. Bański, J.: Współczesne typologie obszarów wiejskich w Polsce - przegląd podejść metodologicznych (Contemporary typologies of rural areas in Poland - an overview of methodological approaches). Przegląd Geograficzny 86(4), 441–470 (2014)
4. Bazan, J., Skowron, A., Swiniarski, R.: Rough sets and vague concept approximation: From sample approximation to adaptive learning. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets V, Lecture Notes in Computer Science, vol. 4100, pp. 39–62. Springer-Verlag, Berlin Heidelberg (2006)
5. Dubois, D., Prade, H.: Rough fuzzy sets and fuzzy rough sets. International Journal of General Systems 17(2-3), 191–209 (1990)
6. Ishizu, S., Gehrmann, A., Nagai, Y., Inukai, Y.: Rough ontology: Extension of ontologies by rough sets. In: Smith, M.J., Salvendy, G. (eds.) Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design, Lecture Notes in Computer Science, vol. 4557, pp. 456–462. Springer-Verlag, Berlin Heidelberg (2007)
7. Keet, C.M.: Ontology engineering with rough concepts and instances. In: Cimiano, P., Pinto, H.S. (eds.) Knowledge Engineering and Management by the Masses,

Lecture Notes in Artificial Intelligence, vol. 6317, pp. 503–513. Springer-Verlag, Berlin Heidelberg (2010)
8. Nguyen, L.A.: Paraconsistent and approximate semantics for the OWL 2 Web Ontology Language. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) Rough Sets and Current Trends in Computing, Lecture Notes in Artificial Intelligence, vol. 6086, pp. 710–720. Springer-Verlag, Berlin Heidelberg (2010)
9. Pancerz, K., Grochowalski, P., Derkacz, A.: Towards the ontology of places in Poland: an example of the Mazowieckie voivodship. Barometr Regionalny. Analizy i Prognozy 14(3) (2016)
10. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht (1991)
11. Ziarko, W.: Variable precision rough set model. Journal of Computer and System Sciences 46(1), 39–59 (1993)

Reversing Transitions in Bounded Petri Nets*

Kamila Barylska[1], Evgeny Erofeev[2], Maciej Koutny[3],
Łukasz Mikulski[1], and Marcin Piątkowski[1]

[1] Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University
Toruń, Chopina 12/18, Poland
{kamila.barylska,lukasz.mikulski,marcin.piatkowski}@mat.umk.pl

[2] Parallel Systems, Department of Computing Science
Carl von Ossietzky Universität
D-26111 Oldenburg, Germany
evgeny.erofeev@informatik.uni-oldenburg.de

[3] School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@newcastle.ac.uk

Abstract. Reversible computation deals with mechanisms for undoing the effects of actions executed by a dynamic system. This paper is concerned with reversibility in the context of Petri nets which are a general formal model of concurrent systems. A key construction we investigate amounts to adding 'reverse' versions of selected net transitions. Such a static modification can severely impact on the behaviour of the system, e.g., the problem of establishing whether the modified net has the same states as the original one is undecidable. We therefore concentrate on nets with finite state spaces and show, in particular, that every transition in such nets can be reversed using a suitable set of new transitions.

Keywords: Petri net, reversibility, reversible computation

1 Introduction

Reversible computation deals with (typically local) mechanisms for undoing the effects of actions executed by a dynamic system. Such an approach has been applied, in particular, to various kinds of process calculi and event structures (see, e.g., [3–6, 8, 11, 12, 10]), and to a category theory based setting [7].

* This research has been partially supported by the Polish grant No.2013/09/D/ST6/03928, and by the EU COST Action IC1405, and by DFG (German Research Foundation) through grant Be 1267/14-1 CAVER (Design and Analysis Methods for Real-Time Systems) and Graduiertenkolleg GRK-1765 SCARE (System Correctness under Adverse Conditions).

This paper is concerned with reversibility in the context of Petri nets which are a general formal model of concurrent systems. A key construction we investigate amounts to adding 'reverse' versions of selected net transitions, e.g., a 'straightforward' reverse simply changes the directions of arcs adjacent to a transition being reversed. As shown in [2], such a static modification can severely impact on the behaviour of the system, e.g., the problem of establishing whether the modified net has the same states as the original one is undecidable.

We therefore concentrate in this paper on Petri nets with finite state spaces, more precisely bounded Place/Transition-nets (PT-nets). The state spaces of such nets can be represented by finite labelled transition systems (flts's) which are a convenient tool for specifying different variants of reversibility. One can therefore aim at synthesising a PT-net with 'reversed' behaviour given by an flts.

In this paper we show that it is, in general, impossible to reverse a transition using its straightforward reverse. What is more, the situation does not change if we relax the notion of a reverse by only requiring that the effect of its execution is opposite to that of the original transition. We therefore relax the requirement further, by allowing several reverses for a single transition. This leads to our main result that every transition in a bounded PT-net can be reversed using a suitable set of new transitions.

2 Preliminaries

Transition systems
A *finite labelled transition system* (or, simply, *flts*) is a tuple $TS = (S, T, \rightarrow, s_0)$ with a finite set of *states* S, a finite set of *labels* T, a finite set of *arcs* $\rightarrow \subseteq (S \times T \times S)$, and an *initial state* $s_0 \in S$.[4] A label t is *fireable* at $s \in S$, denoted by $s[t\rangle$, if $(s, t, s') \in \rightarrow$, for some $s' \in S$. A state s' is *reachable* from s through the execution of $\sigma \in T^*$, denoted by $s[\sigma\rangle s'$, if there is a directed path from s to s' whose arcs are labelled consecutively by σ. The set of states reachable from s is denoted by $[s\rangle$. A sequence $\sigma \in T^*$ is *fireable*, from a state s, denoted by $s[\sigma\rangle$, if there is some state s' such that $s[\sigma\rangle s'$.

Let $t^\bullet_{TS} = \{s \in S \mid (s', t, s) \in \rightarrow, \text{ for some } s' \in S\}$ and ${}^\bullet t_{TS} = \{s \in S \mid (s, t, s') \in \rightarrow, \text{ for some } s' \in S\}$ be respectively the sets of all states having an incoming arc labeled with t, and an outgoing arc labeled with t. The set of all arcs labelled by t is denoted by $\overrightarrow{t}$. We assume that each $\overrightarrow{t}$ is nonempty.

Two flts's, $TS_1 = (S_1, T, \rightarrow_1, s_{0_1})$ and $TS_2 = (S_2, T, \rightarrow_2, s_{0_2})$, are *isomorphic* if there is a bijection $\zeta\colon S_1 \rightarrow S_2$ with $\zeta(s_{0_1}) = s_{0_2}$ and $(s, t, s') \in \rightarrow_1 \Leftrightarrow (\zeta(s), t, \zeta(s')) \in \rightarrow_2$, for all $s, s' \in S_1$.

Petri nets
A *Place/Transition Petri net* (or, simply, *net*) is a tuple $N = (P, T, F, M_0)$,

[4] An flts may be considered as a finite automaton with no accepting states.

where P is a finite set of *places*, T is a finite set of *transitions* (or actions), F is the *flow function* $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ specifying the arc weights, and M_0 is the *initial marking* (where a marking is a mapping $M\colon P \to \mathbb{N}$, indicating the number of tokens in each place). A transition $t \in T$ is *enabled* at a marking M, denoted by $M[t\rangle$, if $M(p) \geq F(p,t)$, for all $p \in P$. The *effect* of a transition t on a place p is $\mathit{eff}_p(t) = F(t,p) - F(p,t)$. The firing of t at marking M leads to M', denoted by $M[t\rangle M'$, if $M[t\rangle$ and $M'(p) = M(p) + \mathit{eff}_p(t)$ for every $p \in P$.

The notions of enabledness and firing, $M[\sigma\rangle$ and $M[\sigma\rangle M'$, are extended in the usual way to sequences $\sigma \in T^*$, and $[M\rangle$ denotes the set of all markings reachable from M. We assume that each transition is enabled in at least one reachable marking. There is a partial order relation $<$ on the markings of a Petri net defined so that $M \leq M'$ if $M(p) \leq M'(p)$, for every place $p \in P$. It is easy to observe that transition enabledness is *monotonic*, which means that if a transition t is enabled at a marking M and $M \leq M'$, then t is also enabled at M'.

A Petri net $N = (P, T, F, M_0)$ net is *bounded* if $[M_0\rangle$ is finite, and its *reachability graph* is then defined as an flts

$$RG(N) = ([M_0\rangle, T, \{(M, t, M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}, M_0).$$

If a labelled transition system TS is isomorphic to the reachability graph of a Petri net N, then we say that N *solves* TS, and TS is *synthesisable* to N.

Definition 1 (transition reverse). *A* (strict) reverse *of a transition $t \in T$ in a net $N = (P, T, F, M_0)$ is a new transition $\underline{t}$ such that $F(p, \underline{t}) = F(t, p)$ and $F(\underline{t}, p) = F(p, t)$. An* effect-reverse *of a transition $t \in T$ is a new transition $\overline{t}$ such that $\mathit{eff}_p(\overline{t}) = -\mathit{eff}_p(t)$, for all places $p \in P$.*

To improve readability, we depict newly created reverses and adjacent arcs by dashed (or dotted) lines. Clearly, for a given transition t, its strict reverse $\underline{t}$ is unique and, at the same time, it is an effect-reverse of t. However, an effect-reverse $\overline{t}$ is not necessarily a strict reverse (see Figure 1).

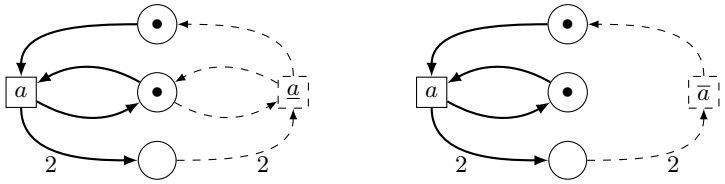


Fig. 1. A transition a and its (strict) reverse $\underline{a}$ (lhs), and an effect-reverse $\overline{a}$, which is not a strict reverse (rhs).

(Un)solvable words

A word $w = t_1 t_2 \dots t_n$ of length $n \in \mathbb{N}$ uniquely corresponds to a labelled transition system $TS(w) = (\{0, \dots, n\}, T, \{(i-1, t_i, i) \mid 0 < i \leq n \wedge t_i \in T\}, 0)$.

We say that a net N solves a word w if it solves $TS(w)$. A word w is then called *solvable*, and otherwise *unsolvable*.

If a word w is solvable, then so are all its factors (where a *factor* w' satisfies $w = vw'u$, for some v and u). Thus, the unsolvability of any proper factor of w entails the unsolvability of w. For this reason, the notion of a *minimal unsolvable word*, defined as an unsolvable word with all proper factors being solvable, is well-defined (see [1] for details).

The *mirror image* w^R of a word w is w written from right to left.

3 Solvability of flts's with reverses

We now define reverses for labelled transition systems, and investigate how they affect the solvability of the resulting flts's. We first introduce the notions of reduction and extension of an flts.

Definition 2 (flts reduction and extension). *Let $TS = (S, T, \rightarrow, s_0)$ be a solvable flts.*

- *The* reduction *of TS by deleting $t \in T$ is an flts $TS^{[-t]} = (S', T \setminus \{t\}, \rightarrow', s_0)$ such that:*
  - *$S' \subseteq S$ are all the states reachable in TS without using $\overrightarrow{t}$;*
  - *$(s_1, a, s_2) \in \rightarrow'$ if $(s_1, a, s_2) \in \rightarrow$, for all $a \neq t$ and $s_1, s_2 \in S'$.*
- *The* extension *of TS by reversing $t \in T$ is an flts $TS^{[+\overline{t}]} = (S, T \cup \{\overline{t}\}, \rightarrow', s_0)$ such that, for all $s_1, s_2 \in S$:*
  - *$(s_1, a, s_2) \in \rightarrow'$ if $(s_1, a, s_2) \in \rightarrow$, for all $a \in T$;*
  - *$(s_1, \overline{t}, s_2) \in \rightarrow'$ if $(s_2, t, s_1) \in \rightarrow$.*

These above notions can be extended to finite sets of transitions, by setting $TS^{[-t_1,t_2 \ldots t_n]} = TS^{[-t_1][-t_2]\ldots[-t_n]}$ and $TS^{[+\overline{t_1},t_2 \ldots t_n]} = TS^{[+\overline{t_1}][+\overline{t_2}]\ldots[+\overline{t_n}]}$.

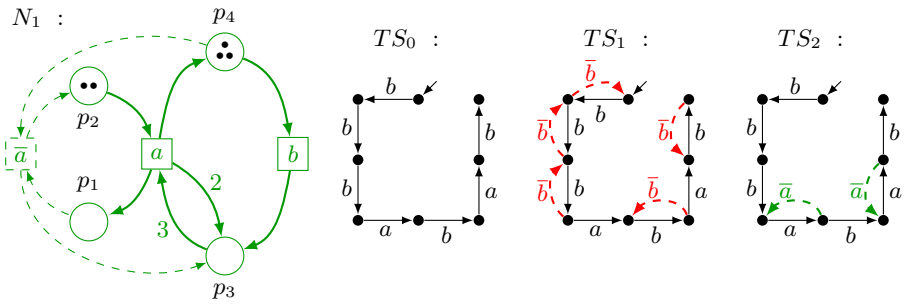


Fig. 2. TS_0 and $TS_2 = TS_0^{[+\overline{a}]}$ are solvable by net N_1 (respectively without and with the dashed part), while $TS_1 = TS_0^{[+\overline{b}]}$ is unsolvable.

Consider a word $w = bbbabab$ which, in Figure 2, corresponds to a solvable flts TS_0. If we add a reverse of transition a, we obtain TS_2 which is solvable by N_1. We will later show that reversing transition b leads to an unsolvable flts TS_1.

The $\overline{a}$ in Figure 2 is an effect-reverse but not a strict reverse of a. We will now show that if a label a can be effect-reversed, i.e., $TS^{[+\overline{a}]}$ is solvable, then there exists a solution in which transition $\overline{a}$ is a strict reverse of a.

Proposition 1. *Let $TS = (S, T, \to, s_0)$ be a solvable flts and $a \in T$. If $TS^{[+\overline{a}]}$ is solvable then there exists its solution such that $\overline{a}$ is a strict reverse of a.*

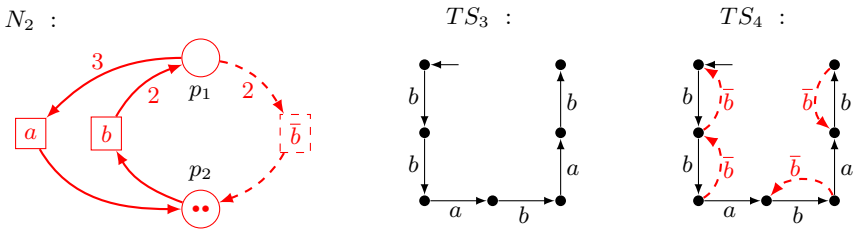


Fig. 3. Adding a reverse $\overline{b}$ in $TS_4 = TS_3^{[+\overline{b}]}$ does not violate solvability.

Consider N_2 of Figure 3 without the dashed part. It solves the word *bbabab*, and so its reachability graph is isomorphic to TS_3. Unlike the case with the reverse of b in TS_1, TS_4 obtained from TS_3 by adding a reverse for transition b is solvable by N_2 with dashed part. Note that, in N_2, $\overline{b}$ is a strict reverse of b.
Similarly, we may reverse a in TS_3, obtaining TS_5 of Figure 4. This flts is solvable by the net N_3 with the dashed part.

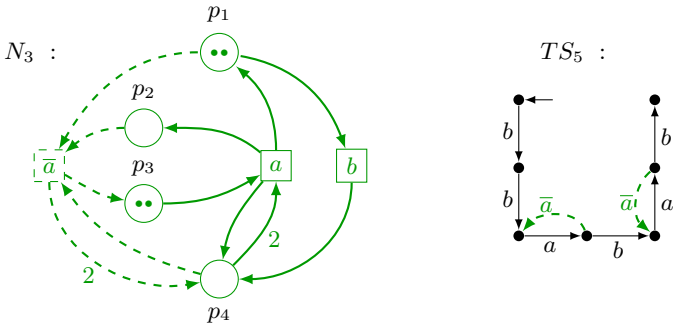


Fig. 4. $TS_5 = TS_3^{[+\overline{a}]}$ is solvable (e.g. by N_3).

The next result states that for a given flts and two of its transitions, if adding a reverse for each of them separately yields solvable flts's, then the flts with both reverses is also solvable.

Proposition 2. *Let $TS = (S, T, \rightarrow, s_0)$ be a solvable flts and $a \neq b \in T$. If both $TS^{[+\overline{a}]}$ and $TS^{[+\overline{b}]}$ are solvable, then so is $TS^{[+\overline{a,b}]}$.*

For $TS = TS_3$ of Figure 3, by Proposition 2, starting from the solutions for $TS_4 = TS^{[+\overline{a}]}$ and $TS_5 = TS^{[+\overline{b}]}$, we can construct a solution N_4 for $TS_6 = TS^{[+\overline{a,b}]}$ depicted in Figure 5.

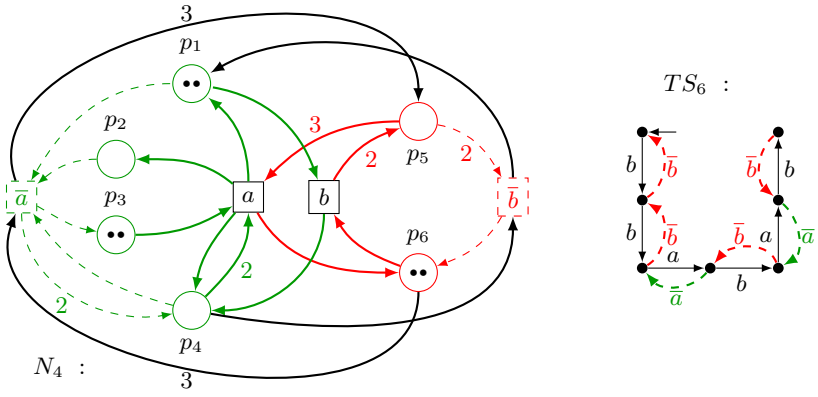


Fig. 5. N_4 solving $TS_6 = TS_3^{[+\overline{a,b}]}$ derived by synchronising the transitions of N_2 and N_3.

We end this section looking at the solvability of words over a two-letter alphabet.

Proposition 3. *Let $w \in \{a, b\}^*$ be a minimal unsolvable word. Then $TS(w^R)$ is solvable.*

Due to Propositions 2 and 3, reversing of both transitions in the mirror image w^R of some minimal unsolvable word w over $\{a, b\}$ yields solvability of w, which is a contradiction. Hence, the following corollary holds

Corollary 1. *Let $w \in \{a, b\}^*$ be a minimal unsolvable word and $TS = TS(w^R)$. Then $TS^{[+\overline{a}]}$ or $TS^{[+\overline{b}]}$ is unsolvable.*

The above result explains why b in TS_1 of Figure 2 cannot be reversed. All we need to observe is that $w = bbbabab$ is the mirror image of a minimal unsolvable word $bababbb$, and then recall that a can be reversed in TS_1.

4 Splitting reverses

In this section we discuss the possibility of "splitting" reverses. More specifically, we investigate flts's in which more than one reverse to a given transition can exist.

Consider N_5 of Figure 6, together with its reachability graph TS_7. First, we observe that $\mathit{eff}_{\overline{b}_1}(p) = \mathit{eff}_{\overline{b}_2}(p) = -\mathit{eff}_b(p)$, for every place p. Hence, transitions $\overline{b}_1$ and $\overline{b}_2$ are both effect-reverses for b. We have already seen that it is impossible to synthesise an flts with just one reverse of b (i.e., TS_1 of Figure 2), but the behaviour of N_5 is exactly what one might indeed want to obtain. The only difference is that N_5 has more than one reverse for b. In what follows, we show that every action of a bounded net can be reversed using finitely many effect-reverses.

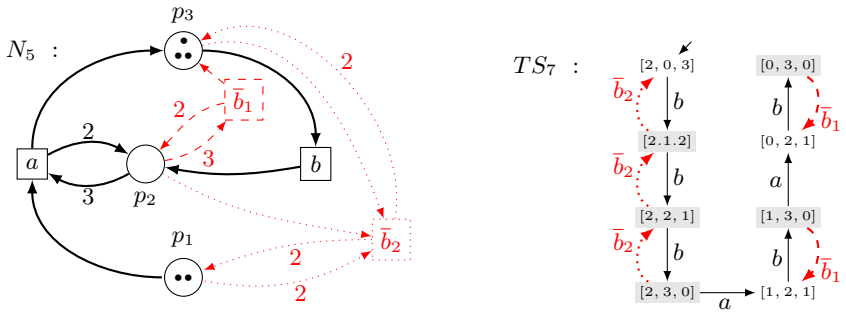


Fig. 6. Splitting reverses in TS_7 results in solvability.

Definition 3 (splitting reverse). *Let $TS = (S, T, \to, s_0)$ be a solvable flts. The* extension *of TS by a set $\overline{T}$ of reverses of $t \in T$ is an flts $TS^{[+t\phi]} = (S, T', \to', s_0)$ such that:*

- $\phi : \overrightarrow{t} \to 2^{\overline{T}} \setminus \{\emptyset\}$ *is a mapping specifying all possible ways in which each of t-labelled arcs can be reversed;*
- $T' = T \cup \overline{T}$*;*
- $(s_1, a, s_2) \in \to'$ *if* $(s_1, a, s_2) \in \to$*, for any $a \in T$;*
- $(s_1, t', s_2) \in \to'$ *if* $(s_2, t, s_1) \in \to$ *and* $t' \in \phi((s_1, t, s_2))$.

We also extend the above notion in the usual way to $TS^{[+t_1\phi_1, t_2\phi_2, \ldots, t_n\phi_n]}$.

Lemma 1. *Let $N = (P, T, F, M_0)$ be a bounded net, $TS = ([M_0\rangle, T, \to, M_0)$ be its reachability graph, and $t \notin T$ be a new transition symbol. If a reachable marking M is $\leq$-maximal in $[M_0\rangle$ and $M' \in [M_0\rangle$, then*

$$TS' = ([M_0\rangle, T \cup \{t\}, \to \cup \{(M, t, M')\}, M_0)$$

is a solvable flts.

Proof. Let $N' = (P, T \cup \{t\}, F', M_0)$, where:

$$\begin{array}{ll} F'(p,a) = F(p,a) & \text{for all } p \in P \text{ and } a \in T \\ F'(a,p) = F(a,p) & \text{for all } p \in P \text{ and } a \in T \\ F'(p,t) = M(p) & \text{for every } p \in P \\ F'(t,p) = M'(p) & \text{for every } p \in P. \end{array}$$

We then obtain that:

(1) t is not enabled at any marking $M'' \neq M$ reachable in N. Indeed, suppose that there exists such a marking M''. Then, by the definition of enabledness, $M''(p) \geq F'(p,t) = M(p)$, for every $p \in P$. Hence $M'' \geq M$, which contradicts the $\leq$-maximality of M.
(2) $M[t\rangle M'$. This follows directly from the definition of F'.

We then observe that, by (1) and (2), the sets of reachable markings of the nets N and N' are equal, and $RG(N') = TS'$. □

Lemma 1 states that to a given solvable flts (with a solution $N = (P, T, F, M_0)$) one can always add a new edge $(s, t_{(s,s')}, s')$, obtaining another solvable flts, provided that s is a state corresponding to some marking M, which is $\leq$-maximal in $[M_0\rangle$, and $t_{(s,s')}$ denotes the label of the edge from s to s', such that $t_{(s,s')} \notin T$. We will use this fact to prove the following theorem

Theorem 1. *Let $TS = (S, T, \rightarrow, s_0)$ be a solvable flts. Then, for every $t \in T$, there exists a finite set $\overline{T}$ and a function $\phi : \overrightarrow{t} \rightarrow 2^{\overline{T}} \setminus \{\emptyset\}$ such that $TS^{[+t\phi]}$ is solvable.*

Proof. Let $N = (P, T, F, M_0)$ be a net solving TS. As TS is finite, N is bounded, and so we can calculate a common bound n on the tokens in the reachable markings for all the places, $n = \max(M(p) \mid M \in [M_0\rangle, p \in P)$.

We extend N to $N' = (P \cup P', T, F', M_0')$ by adding complement places [9] $P' = \{p' \mid p \in P\}$ in such a way that, for all $M \in [M_0\rangle$ and $p \in P$, we define M', such that $M'(p) = M(p)$ and $M'(p') = n - M(p)$. This can be done by inserting in the initial marking $n - M_0(p)$ tokens into each $p' \in P'$, and setting $F'(p', a) = F(a, p)$ as well as $F'(a, p') = F(p, a)$, for all $p' \in P'$ and $a \in T$.

Since, for distict markings $M_1, M_2 \in [M_0'\rangle$, there exists a place $p \in P$ (in which they differ) such that $M_1(p) > M_2(p)$ and $M_1(p') < M_2(p')$, or $M_2(p) > M_1(p)$ and $M_2(p') < M_1(p')$, all distinct markings reachable in N' are $\leq$-incomparable. Hence all markings reachable in N' are $\leq$-maximal in $[M_0'\rangle$. By the construction, the reachability graph of N' is isomorphic to TS.

We then construct TS' by adding to TS a set $\overline{T}$ of $|\overrightarrow{t}|$ new transitions in such a way that, for every $(p, t, q) \in \rightarrow$, we also add $(q, t_{(q,p)}, p) \in \rightarrow$. We then define a function $\phi : \overrightarrow{t} \rightarrow 2^{\overline{T}} \setminus \{\emptyset\}$ in such a way that $\phi((p, t, q)) = \{t_{(q,p)}\}$.

Finally, by repeatedly using Lemma 1 for the net N', we obtain that $TS' = TS^{[+t\phi]}$ is solvable. □

The construction described in the proof of Theorem 1 will in most cases lead to a substantial enlargement of the net, as the size of places is doubled, and the number of newly created transitions is bounded by the size of the reachability graph of the initial net. However, as illustrated by the example depicted in Figure 6, there may also exist solutions that are much smaller. Hence, there is a room for improvement of the suggested constructive technique.

5 Infeasibility for reversing

To draw attention to another important issue, which becomes relevant during the analysis of flts's from the viewpoint of reversibility of transitions, let us consider the following example.

Suppose that one attempted to introduce a reverse for a in TS_8 of Figure 7, which can be solved by N_6. Although there exists a (strict) reverse $\overline{a}$ in N_6, depicted in Figure 7, the meaning of $\overline{a}$ may be confusing. We cannot regard it as an undoing of the executing of action a, since N_6 can fire $bc\overline{a}$ where a does not occur at all. What is more, we can keep repeating the firing of $bc\overline{a}$ indefinitely, without executing a even once. To address this situation, we introduce the notion of *infeasibility for reversing*.

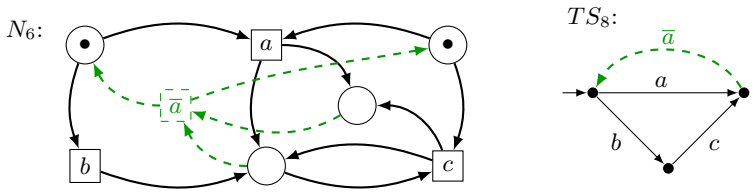


Fig. 7. $TS^{[+\overline{a}]}$ allows execution of $\overline{a}$ without executing of a.

Definition 4. *Let $TS = (S, T, \to, s_0)$ be an flts. Then $a \in T$ is* infeasible (for reversing)*, if $TS^{[+\overline{a}]}$ has a path starting from s_0 with more occurrences of $\overline{a}$ than a. Otherwise, a is* feasible (for reversing)*.*

There is a straightforward necessary condition for being feasible for reversing.

Proposition 4. *Let $TS = (S, T, \to, s_0)$ be an flts and $t \in T$. If $TS^{[-t]}$ has a path from ${}^\bullet t_{TS} \cup \{s_0\}$ to $t^\bullet_{TS}$ then t is infeasible for reversing.*

In general, the reversed implication does not hold. Take, for example, $TS_{10} = TS_9^{[+\overline{a}]}$ of Figure 8. It has a path labelled $acd\overline{a}\overline{a}$, with more $\overline{a}$'s than a's, implying

the infeasibility for reversing of transition a in TS_9. However, the reduction of TS_9 by deleting a, namely $TS_9^{[-a]}$ has no path starting from ${}^\bullet a_{TS_9} \cup \{s_0\}$ to $a_{TS}^\bullet$. Note that TS_{10} and TS_9 are both solvable (see N_7 of Figure 8 with or without dashed arcs, respectively). We will now show that one can always

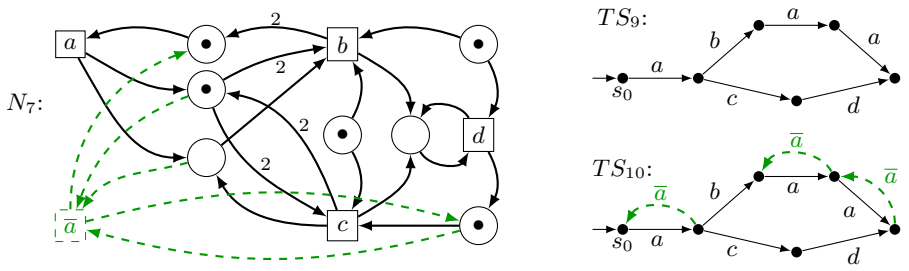


Fig. 8. a is infeasible for reversing in TS_9, even though $TS_9^{[-a]}$ has no path from ${}^\bullet a_{TS_9} \cup \{s_0\}$ to $a_{TS_9}^\bullet$.

establish whether a label of an flts is (in)feasible for reversing. To this end, formulate the following decision problem:

Feasibility for Reversing Problem
Instance: An flts $TS = (S, T, \rightarrow, s_0)$ and $t \in T$.
Question: Is t feasible for reversing in TS?

Proposition 5. *The Feasibility for Reversing Problem is decidable.*

Proof (Sketch of the algorithm.).

The following algorithm reduces the problem of checking the feasibility of a transition for reversing to the problem of finding shortest paths in a weighted digraph.

Input: An flts $TS = (S, T, \rightarrow, s_0)$ and $t \in T$.
Output: YES if t is feasible for reversing in TS; otherwise NO.

Procedure:

1. Compute a weighted graph $G = (V, E, w)$ on the basis of the extension $TS^{[+\overline{t}]} = (S, T \cup \{\overline{t}\}, \rightarrow', s_0)$ of TS, in the following way (for all $s, s' \in S$, $a \in T \cup \{\overline{t}\}$):
   - $V = S$;
   - $(s, s') \in E$ if $(s, a, s') \in \rightarrow'$;
   - $w((s, s')) = \begin{cases} 1 & \text{if } (s, t, s') \in \rightarrow' \\ -1 & \text{if } (s, \overline{t}, s') \in \rightarrow' \\ 0 & \text{otherwise.} \end{cases}$

2. Use, e.g., Bellman-Ford algorithm, to search for a state s_{wit}, such that the distance between s_0 and s_{wit} is negative.
3. If s_{wit} exists, return NO and otherwise YES. □

For a transition system consisting of n states the preprocessing phase (step 1) can be done in time $O(n^2)$. The computation of step 2 can be performed in time $O(n^3)$ (basing on Bellman-Ford algorithm). Therefore the overall complexity of the algorithm is $O(n^3)$.

6 Concluding remarks

In this paper, we have investigated reversibility of transitions in bounded nets. In particular, we have shown that each transition in such nets can be reversed using a suitable set of new transitions, but not necessarily a single reverse transition. In future, we plan to investigate ways in which the generation of sets of reverses could be optimised.

Acknowledgements

We thank Uli Schlachter for assistance with preliminary tests and for adding new modules to APT. We are also grateful to all anonymous reviewers for their constructive comments.

References

1. Kamila Barylska, Eike Best, Evgeny Erofeev, Łukasz Mikulski, and Marcin Piątkowski. On binary words being Petri net solvable. *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015*, 1371:1–15, 2015.
2. Kamila Barylska, Maciej Koutny, Łukasz Mikulski, and Marcin Piątkowski. Reversible computation vs. reversibility in Petri nets. *Reversible Computation - 8th International Conference, RC 2016, Proceedings*, 9720:105–118, 2016.
3. Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.
4. Luca Cardelli and Cosimo Laneve. Reversible structures. In François Fages, editor, *Proceedings of 9th International Computational Methods in Systems Biology (CMSB'11)*, pages 131–140. ACM, 2011.
5. Vincent Danos and Jean Krivine. Reversible communicating systems. In *Proceedings of 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science (LNCS)*, pages 292–307. Springer-Verlag (New York), 2004.
6. Vincent Danos and Jean Krivine. Transactions in RCCS. In *Proceedings of 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science (LNCS)*, pages 398–412. Springer-Verlag (New York), 2005.

7. Vincent Danos, Jean Krivine, and Pawel Sobocinski. General reversibility. *Electronic Notes Theoretical Computer Science*, 175(3):75–86, 2007.
8. Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversing higher-order Pi. In *Proceedings of 21th International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010.
9. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541 – 580, 1989.
10. Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming*, 73(1-2):70–96, 2007.
11. Iain Phillips and Irek Ulidowski. Reversibility and asymmetric conflict in event structures. *Journal of Logic and Algebraic Methods in Programming*, 84(6):781–805, 2015.
12. Iain Phillips, Irek Ulidowski, and Shoji Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Proceedings of 4th Workshop on Reversible Computation (RC'12)*, volume 7581 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2012.

Towards Efficient Verification of Elementary Object Systems

Ismaila Jihad Abdullahi, and Berndt Müller
University of South Wales
(Ismaila.abdullahi, bertie.muller)@southwales.ac.uk

Abstract. Elementary Object Systems (EOS) is a class of Object Petri Nets that follows the "*nets-within-nets*" paradigm. It combines several practical as well as theoretical properties for the needs of multi-agent-systems. However, it comes with some constraints that limit their expressiveness for automatic verification purposes due to the highly expressive nature of the underlying class of Petri nets. In this paper, we proposed a set of transformation rules from EOS to basic Petri nets nets and show isomorphism of the state spaces in order to make verification feasible.

Keywords: Elementary Reference-net System, nets-within-nets, Petri nets, isomorphic property, computational complexity

1 Introduction

Elementary Object Systems (EOS for short) are based on the *nets-within-nets* paradigm of (Valk, 1991,2003) in which the nesting of nets involved in the model is restricted to two levels and are generalised in (Köhler and Heitmann, 2009) for arbitrary nesting structure. This formalism provides a modelling technique that allows tokens of Petri nets to be Petri nets themselves, called *object nets*. Object nets are tokens with internal structure and inner activity and have been applied in a variety of scenarios, e.g., multi-agent systems.

We aim to provide a path to verification of properties of a slightly modified version of EOS, called *elementary reference-net systems (ERS)*, with reference semantics that is practically relevant and overcomes fundamental decidability issues with other formalists as shown in (Köhler and Rölke, 2004) and (Lomazova, and Schnoebelen, 1999). As in similar formalists, we have to distinguish autonomous and synchronous transitions. The need for application of a *partial order (unfolding)* approach for dynamic analysis of EOS have encouraged and driven the development of this new formalism. We refer the reader to (Valk, 1991) for an introduction to the nets-within-nets.

Compared to EOS, two main additions are introduced for ERS: Firstly, we provide each marked object net located in places of the system net with a *unique name* so that object nets with the same marking can be distinguished. Secondly, we use variables to label arcs of the system net. So that when firing transitions, variables are bound to

object nets names instead of statically *typing* system net places allowing dynamic use of net-tokens without fixing *types* for places of the system net.

We extend the notion of 1-safe P/T nets to ERS to guarantee that the state space is finite and markings are bounded. Further to the definition of ERS, we propose a set of transformation rules from 1-safe ERS into P/T nets and show isomorphism of the state spaces of ERS with its generated P/T net.

In Section 2 we review some preliminaries from Petri net theory. Section 3 gives an introduction to ERS. Section 4 presents the set of transformation rules from 1-safe ERS to 1-safe P/T nets. Section 5 proves the isomorphism of the state spaces of 1-safe ERS and of the transformed 1-safe P/T net.

2 Fundamentals of Petri nets

Here we give some definitions from theory of P/T-net, Relevant for our study.

Definition 2.1(P/T net) *A place/transition is a tuple* $N = (P, T, F, W)$*where* P is a finite set of places, T is a finite set of transitions, disjoint from P, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and $W: F \longrightarrow \mathbb{N}\backslash\{0\}$ is the arc weight function. *The pre-set of a node* $x \in P \cup T$*, denoted* $\bullet x$*, is the set containing the elements that immediately precede* x *in the net i.e.:* $\bullet x = \{y \in P \cup T | (y, x) \in F\}$*. Analogously, the postset of a node is denoted* $x\bullet$.

Definition 2.2 (Marking and Enabled transition). A marking of a P/T-net $N = (P, T, F, W)$ is a function $m: P \longrightarrow \mathbb{N}$. *A P/T net system* $\Sigma = (N, m_0)$ *is a net* $N = (P, T, F)$ *together with an initial marking* m_0*. Let* $\Sigma = (N, m_0)$ *be a net system. A transition* $t \in T$ *is enabled in a marking* m *iff* $m(p) \geq W(p, t)$*for all* $p \in \bullet t$*. An enabled transition* t *in marking* m *is denoted by* $m[t >$*. A transition that is enabled in a marking may or may not fire.* Firing of transition removes tokens from input places of t and puts new tokens onto output places of t. *The successor marking* m' *is defined as* $m'(p) = m(p) - W(p, t) + W(t, p)$*. We denote this by* $m[t > m'$*. For a finite sequence of transition* $\sigma = t_1, \ldots t_k$*, we write* $m[\sigma > m'$ *if there are markings* $m_1, \ldots m_{k+1}$ *such that* $m_1 = m, m_{k+1} = m'$ *and* $m_i[t_i > m_{i+1}$*, for all* $i = 1, \ldots, k$. The set of reachable markings of Σ is the set of all markings reachable from the initial marking. Σ is k-bounded if, for every reachable marking m and every place p ∈ P, m(p) ≤ k, and Σ is safe if it is 1-bounded. Moreover, Σ is bounded if it is k-bounded, for some k ∈ N. One can show that the set RM(Σ) is finite if Σ is bounded i.e. if $|\mathrm{RM}(\Sigma)| < \infty$.

3 Elementary Reference-net System (ERS)

By convention, the components of the system net will carry a hat: $\hat{P}, \hat{T}, \hat{p}, \hat{t}$, ... etc.

Definition 3.1 *Let the triple* $\eta_i = (i, N_i, m_i)$ *be a named marked object net, where* i*, is a unique name of an object net;* N_i *is a structure of the object net, and* m_i *is a marking in* N_i*. (Let* $\Sigma = \{(i_1, N_1, m_1), \ldots, (i_k, N_k, m_k)\}$ *be a finite set of unique*

marked named object nets). The structure of an object net with a unique name $i \in \Sigma$ is a P/T- net $N_i = (P_i, T_i, F_i)$, where P_i, is the set of places of the object net, T_i is the set of its transitions and $F_i \subseteq (P_i \times T_i) \cup (T_i \times P_i)$ is the flow relation. We assumed that all sets of nodes are pairwise disjoint and set $P_\Sigma := \bigcup_{\eta_i \in \Sigma} P_{\eta_i}$ and $T_\Sigma := \bigcup_{\eta_i \in \Sigma} T_{\eta_i}$. By $N_\bullet$ we denote the name of ordinary black tokens.

Definition 3.2 (ERS) *Let Var be a finite set of named variables. An elementary reference-net system is a tuple $RS = (\widehat{N}, \Sigma_{m^0}, \ell, \omega, \boldsymbol{R^0})$ where*

- $\widehat{N} = (\hat{P}, \hat{T}, \hat{F})$ is a p/t net called a system net, where $\hat{P}$ is its set of places, $\hat{T}$is its set of transitions and $\hat{F} \subseteq (\hat{P} \times \hat{T}) \cup (\hat{T} \times \hat{P})$ is the flow relation.
- $\Sigma_{m^0} := \{(i_1, N_1, m_1^0), \dots, (i_k, N_k, m_k^0)\}$, is a finite set of marked named object nets.
- $\ell \subseteq (\hat{T} \cup \{\hat{\tau}\}) \times (T_{1\mathrm{i}} \cup \{\tau\}) \times, \dots, (T_k \cup \{\tau\}) \backslash \{\hat{\tau}, \tau, \dots, \tau)\}$, is the synchronisation relation, where $\hat{\tau}$ and τ are special symbols intended to denote inactions at the system and the object net levels respectively. If $\boldsymbol{t} = (\hat{t}, t_1, \dots, t_k)$ and $\hat{t}, \neq \tau$ and $\exists i \in \{1, \dots, k\}$ such that $t_i \neq \tau$, then we say that $\widehat{N} and\ N_i \in \Sigma$ for every $i \in \{1, \dots, k\}$ with $k = |\Sigma|$, participate in $\boldsymbol{t}$. This is the reason why $(\hat{t}, \tau, \dots, \tau)$ is excluded from the set of synchronisation relation: at least one object net must participate in every synchronisation action with the system net.
- $\omega: \hat{F} \longrightarrow Var \cup \{N_\bullet\}$ is an arc labelling function such that for an arc $\hat{a} \in (\hat{F})$ adjacent to a place $\hat{p}$ the inscription of $\omega(\hat{a})$ matches the name of object net in $\hat{p}$
- $\boldsymbol{R^0}$ specifies the initial making, where $\boldsymbol{R^0}: \hat{P} \to \mathbb{N} \cup MS(\Sigma)\ with\ \Sigma = \{(i_1, N_1, m_1), \dots, (i_k, N_k, m_k\}$. It has to satisfy the condition $\boldsymbol{R^0}(\hat{p}) \in \mathbb{N} \Leftrightarrow \boldsymbol{R^0}(\hat{p}) \in \{N_\bullet\}$.

In the example of Fig. 1 an $RS = (\widehat{N}, \Sigma, \ell, \omega, \boldsymbol{M^0})$ is shown, where $\Sigma = \{N_1, N_2\}$. Arcs of $\widehat{N}$ can be identified by their labelling from $\omega(\hat{t})$. Hence {x, y,} can be bound to marked named object nets in places $\hat{p}_1\ and\ \hat{p}_2$ adjacent to transition $\hat{t}'$ to enable it. In the initial marking, places $\hat{p}_1\ and\ \hat{p}_2$ contain references to the marked named object nets $N_1 and\ N_2$ respectively.

We denote by $\mathcal{N} = \{i | (i, N_{i,}, m_i) \in \Sigma\}$, a finite set of object nets names.

Moreover, variables appearing on arcs adjacent to a transition $\hat{t}$ of the system net must satisfy the following four conditions:

$$\forall \hat{t} \in \hat{T}\ and\ \forall \hat{p} \in {}^\bullet\hat{t}, \exists \hat{p}' \in \hat{t}^\bullet, such\ that\ \omega(\hat{p}, \hat{t}) = \omega(\hat{t}, \hat{p}')\ or\ \omega(\hat{p}, \hat{t}) = N_\bullet \quad (1)$$

$$\forall \hat{t} \in \hat{T}\ and\ \forall \hat{p} \in {}^\bullet\hat{t}, \exists \hat{p}' \in \hat{t}^\bullet, such\ that\ \omega(\hat{p}', \hat{t}) = \omega(\hat{t}, \hat{p})\ or\ \omega(\hat{p}, \hat{t}) = N_\bullet \quad (2)$$

$$\forall \hat{t} \in \hat{T} \text{ and for any two places } \hat{p}_1, \hat{p}_2, \in {}^\bullet\hat{t}, if\ \hat{p}_1 \neq \hat{p}_2\ then\ \omega(\hat{p}_1, \hat{t}) \neq \omega(\hat{p}_2, \hat{t}). \quad (3)$$

$$\forall \hat{t} \in \hat{T} \text{ and } \hat{p}'_1, \hat{p}'_2, \in \hat{t}^\bullet, if\ \hat{p}_1 \neq \hat{p}_2\ then\ \omega(\hat{t}, \hat{p}'_1) \neq \omega(\hat{t}, \hat{p}'_2). \quad (4)$$

Condition (1) says that each variable appearing in the incoming arc of a system net transition $\hat{t}$ also has to appear in the outgoing arc of $\hat{t}$ or no such variable exist. Condition (2) says that each variable appearing in the outgoing arc of a system net transition $\hat{t}$ also has to appear in the incoming arc of $\hat{t}$ or no such variable exist. These two

conditions means that no new object net is created and no destroyed after a transition firing in the system net. Condition (3) prevents the ability to join two object nets, and (4) prevents the splitting of an object net. This is because in reality, complex physical entities cannot be cloned at run time. With these restrictions, ERS still retain the ability to describe nesting of object nets, synchronisation, and mobility, but does not allow splitting of the inner marking of an object net or joining the inner marking of several object nets. Assuming these inner markings as modelling the inner state of an agent, this is a reasonable restriction and ERSs are then well suitable to model physical entities

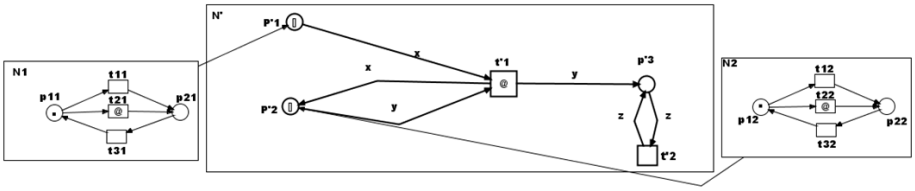


Fig. 1. An example of an ERS

For its behaviour, we introduce the notion of marking for elementary reference-net system ERS under *reference semantics.* Hence in general a marking is given by

1. a distribution of object nets or black tokens $\boldsymbol{R}: \hat{P} \to \mathbb{N} \cup MS(\Sigma)$ and
2. The vector $\boldsymbol{M} = (m_1, \dots, m_k)$ with the current marking of each N_i $(1 \leq i \leq k)$.

$\boldsymbol{R}$ specifies for each system net place $\hat{p}$ a number of black tokens or a multiset of marked named object nets (if $\hat{p}$ contain reference(s) to marked named object nets). If we abbreviate $(m_1, \dots, m_k)$ by $\boldsymbol{M}$ and the set of all such vectors by $\mathcal{M}$, we obtain the following Definition 3.3. By $\Pi_i(\boldsymbol{M})$ we denote the $i-th$ component m_i of $\boldsymbol{M}$ and by $\boldsymbol{M}_{\boldsymbol{i}\to\boldsymbol{m_i}}$ the tuple, where the $i-th$ component is substituted by m_i, $M \in \mathbb{N}^k$.

In what follows a marked named object net is referred to as *net-token.* For a given ERS, by $\sum_{nt} = \Sigma \cup \{N_\bullet\}$ we denote the set of all marked named net-tokens. Only when not introduced in the marking! Sometimes by abuse of notation, for a named object net (i, N_i, m_i) in a place $\hat{p}$ of a marking $\boldsymbol{R}$ of the system net we write $\boldsymbol{R}(\hat{p}) = i$

Definition 3.3 Given *an elementary reference-net system* $RS = \left(\hat{N}, \Sigma_{nt}, \ell, \omega, \boldsymbol{R^0}\right)$ *we define* $\mathcal{M} \coloneqq \{M | M = (m_1, \dots, m_k) \wedge m_i \in MS(P_i)\}$. *Then a marking of an elementary reference-net system is a pair* (R, M) *where* $M \in \mathcal{M}$ *and* $\boldsymbol{R}: \hat{P} \to MS(\Sigma_{nt})$. *Specifying* M^0 *by the initial markings of the marked named object nets* $M^0 = (m_1^0, \dots, m_k^0)$ *we obtain the initial marking* $(\boldsymbol{R^0}, \boldsymbol{M^0})$ *of RS. The set of all markings of RS is denoted by* $\mathcal{M}_\mathrm{r}$.

Let $\hat{t} \in \hat{T}$ be a transition in the system net $\hat{N}$, then $\bullet\hat{t} = \left\{\hat{p} | (\hat{p}, \hat{t}) \in \hat{F}\right\}$, and $\hat{t}\bullet = \{\hat{p} | (\hat{t}, \hat{p}) \in \hat{F}\}$ are sets of its pre- and post-conditions. We denote by $\omega(\hat{t}) \coloneqq \left\{\omega(\hat{p}, \hat{t}) \middle| (\hat{p}, \hat{t}) \in \hat{F}\right\} \cup \left\{\omega(\hat{t}, \hat{p}) \middle| (\hat{t}, \hat{p}) \in \hat{F}\right\} = \bullet\hat{t} \times \{\hat{t}\} \cup \{t\} \times \hat{t}\bullet$ the set of all variables on arcs adjacent to $\hat{t}$. A binding β *specifie*s which variables are bound to names, *where* $\beta: \omega(\hat{t}) \cup \{\bullet\} \longrightarrow \mathcal{N} \cup \{N_\bullet\}$*with* $\mathcal{N} = \left\{i \middle| \left(i, N_{i,}, m_i\right) \in \Sigma\right\}$ *satisfying the condi-*

tions: for each $x \in \mathcal{w}(\hat{t}) \cup \{\bullet\}$, there exist $i \in \mathcal{N}$ *such that* $\beta(x) = i$ *and if* $x = \bullet$ *then* $\beta(x) = N_{\bullet}$.

The *firing rule* will be introduced in three modes.

Definition 3.4 (synchronisation firing mode) *Let* $(\boldsymbol{R}, \boldsymbol{M})$*be a marking of an elementary reference-net system,* $\hat{t} \in \hat{T}$ *a transition of* $\hat{N}$*, and let* β *be a variable binding defined for all* $x \in \mathcal{w}(\hat{t}) \cup \{\bullet\}$*. Let* $\alpha_1, \ldots, \alpha_k \in \Sigma_{nt}$ *be object nets involved in the firing of* $\hat{t}$*. Then* $\hat{t}$ *can fire provided that in each* $\alpha_i \in \Sigma_{nt}$ *for every* $i \in \{1, \ldots, k\}$ *a transition* $t_i \in T_\Sigma$ *such that*$(\hat{t}, t_1, \ldots, t_k) \in \ell$*. Then* $(\hat{t}, t_1, \ldots, t_k)$ *is enabled in* $(\boldsymbol{R}, \boldsymbol{M})$ *if:* $\forall \hat{p} \in \hat{P}$: $(\beta(\mathcal{w}(\hat{p}, \hat{t})), N_{\beta(\mathcal{w}(\hat{p},\hat{t}))}, m_{\beta(\mathcal{w}(\hat{p},\hat{t}))}) \in \boldsymbol{R}(\hat{p})$ and

$$\forall p \in P_i : \Pi_i(\boldsymbol{M}) \geq F_i(p, t_i), \quad (5)$$

This is denoted by $(\boldsymbol{R}, \boldsymbol{M})[\hat{t}, t_i >$ *Let be* $m_i[t_i > m'_i$ *(w.r.t* N_i*). The successor marking* $(\boldsymbol{R}', \boldsymbol{M}')$ *is defined by*

$$\boldsymbol{R}'^{(\hat{p})} = \boldsymbol{R}(\hat{p}) \setminus \left(\beta\left(\mathcal{w}(\hat{p}, \hat{t})\right), N_{\beta(\mathcal{w}(\hat{p},\hat{t}))}, m_{\beta(\mathcal{w}(\hat{p},\hat{t}))}\right) \cup$$
$$\left(\beta\left(\mathcal{w}(\hat{t}, \hat{p})\right), N_{\beta(\mathcal{w}(\hat{t},\hat{p}))}, m_{\beta(\mathcal{w}(\hat{t},\hat{p}))}\right) : \forall \hat{p} \in \hat{P} \text{ and}$$

$$\boldsymbol{M}' = \boldsymbol{M}_{\boldsymbol{i} \to \boldsymbol{m_i}}. \quad (6)$$

This is denoted by $(R, M)[\hat{\boldsymbol{t}}, \boldsymbol{t_i} > (\boldsymbol{R}', \boldsymbol{M}')$.

Definition 3.*5(system-autonomous firing mode)* *Let* $(\boldsymbol{R}, \boldsymbol{M})$ *be a marking of an elementary reference-net system RS* $= \left(\hat{N}, \Sigma_{nt}, \ell, \mathcal{w}, \boldsymbol{R^0}\right)$*and* $\hat{t} \in \hat{T}$ *a transition of* $\hat{N}$ *with a binding* β *such that* $\nexists(\hat{t}, x_i, \ldots, x_k) \in \ell : \exists\, i \in \{1, \ldots, k\} : x_i \neq \tau$*. Then* $\hat{t}$ *is activated in* $(\boldsymbol{R}, \boldsymbol{M})$ *if there is a net token such that:*

$$(\beta(\mathcal{w}(\hat{p}, \hat{t})), N_{\beta(\mathcal{w}(\hat{p},\hat{t}))}, m_{\beta(\mathcal{w}(\hat{p},\hat{t}))}) \in \boldsymbol{R}(\hat{p}) \forall \hat{p} \in \hat{P}. \quad (7)$$

Since we use τ*, for in action, this is denoted by* $(\boldsymbol{R}, \boldsymbol{M})[(\hat{t}, \tau) >$*. The successor marking* $(\boldsymbol{R}', \boldsymbol{M}')$*is defined by*

$$\forall \hat{p} \in \hat{P} : \boldsymbol{R}'(\hat{p}) = \boldsymbol{R}(\hat{p}) \setminus \left(\beta\left(\mathcal{w}(\hat{p}, \hat{t})\right), N_{\beta(\mathcal{w}(\hat{p},\hat{t}))}, m_{\beta(\mathcal{w}(\hat{p},\hat{t}))}\right) \cup \left(\beta\left(\mathcal{w}(\hat{t}, \hat{p})\right), N_{\beta(\mathcal{w}(\hat{t},\hat{p}))}, m_{\beta(\mathcal{w}(\hat{t},\hat{p}))}\right)$$

$$\boldsymbol{M}' = \boldsymbol{M}. \quad (8)$$

This is denoted by $(\boldsymbol{R}.\boldsymbol{M})[(\hat{t}_1, \tau) > (\boldsymbol{R}', \boldsymbol{M}')$.

***Definition 3.6*(object –autonomous firing mode)** *Let* $(\boldsymbol{R}, \boldsymbol{M})$ *be a marking of an elementary reference-net system RS* $= \left(\hat{N}, \Sigma_{nt}, \ell, \mathcal{w}, \boldsymbol{R^0}\right)$*and* $t_i \in T_i$ *a transition of a net-token* $i = (i, N_i, m_i) \in \boldsymbol{R}(\hat{p})$ *for some* $\hat{p} \in \hat{P}$*, such that* $\nexists(\hat{t}, x_i, \ldots, t_i, \ldots, x_k) \in \ell$*, and* t_i *is activated in* N_i*. Then we say that* $(\hat{\tau}, t_i)$ *is activated in* $(\boldsymbol{R}, \boldsymbol{M})$ *(denoted*$(\boldsymbol{R}, \boldsymbol{M})[(\hat{\tau}, t_i) >]$*. The successor marking* $(\boldsymbol{R}', \boldsymbol{M}')$ *of RS is defined by*

$\boldsymbol{R}' = \boldsymbol{R}$ and

$$\boldsymbol{M}' = \boldsymbol{M}_{1 \to \mathrm{m_i}} \text{if } m_i[\, t_i > m'_i \text{ for } \Pi_i(\boldsymbol{M}) = m_{\boldsymbol{i}}. \quad (9)$$

We denote this by $(\boldsymbol{R},\boldsymbol{M})[(\hat{\tau},\ t_i) > (\boldsymbol{R'},\boldsymbol{M'})$.

To introduce the occurrence sequences for ERS we assume an ERS as defined in Definition 3.2. Let RS be an ERS and $(\boldsymbol{R},\boldsymbol{M}),(\boldsymbol{R'},\boldsymbol{M'}) \in \mathcal{M}_r$.

Definition 3.7 *For a new alphabet* $\Gamma \coloneqq (\hat{T} \cup \{\hat{\tau}\}) \times (T_1 \cup \{\tau\}) \times, \dots, (T_k \cup \{\tau\}) \backslash (\hat{\tau},\tau,\dots,\tau)$ *where* $(\hat{\tau},\tau,\dots,\tau)$ *denotes the neutral element of* Γ^**, we define:*

$$(\boldsymbol{R},\boldsymbol{M})[(\hat{\tau},\tau,\dots,\tau) > (\boldsymbol{R'},\boldsymbol{M'}) \text{ if } (\boldsymbol{R},\boldsymbol{M}) = (\boldsymbol{R'},\boldsymbol{M'}) \text{ and}$$

$$(\boldsymbol{R},\boldsymbol{M})[\breve{w}(\hat{t},\alpha) > (\boldsymbol{R'},\boldsymbol{M'}) \text{ if } \exists(\boldsymbol{R''},\boldsymbol{M''}) : (\boldsymbol{R},\boldsymbol{M})[\breve{w} > (\boldsymbol{R''},\boldsymbol{M''}) \text{ and}$$

$$(\boldsymbol{R''},\boldsymbol{M''})[(\hat{t},\alpha) > (R',M') \text{ for some } \breve{w} \in \Gamma^*, \hat{t}, \in \hat{T} \cup \{\hat{\tau}\} \text{ and } \alpha \in ((T_1 \cup \{\tau\}) \times, \dots, (T_k \cup \{\tau\}). \quad (10)$$

To denote that $(\boldsymbol{R'},\boldsymbol{M'})$ *is reachable from* $(\boldsymbol{R},\boldsymbol{M})$ *by some occurrence sequence of actions we write* $(\boldsymbol{R},\boldsymbol{M}) \overset{*}{\rightarrow} (\boldsymbol{R'},\boldsymbol{M'})$.

The set of reachable markings of a reference system RS from a marking $(\boldsymbol{R},\boldsymbol{M})$ *is denoted by* $R(RS,(\boldsymbol{R},\boldsymbol{M}))$. $R(RS)$, *is the set of markings reachable from the initial marking* $(\boldsymbol{R^0},\boldsymbol{M^0})$. *The reachability graph* $(RG(RS)$ *is obtain as for P/T-net systems, which is a digraph whose nodes is the set of reachable markings and edges are the tuples* $((\boldsymbol{R},\boldsymbol{M}),(\hat{t},\alpha),(\boldsymbol{R'},\boldsymbol{M'})) \in \mathcal{M}_r \times (\hat{t},\alpha) \times \mathcal{M}_r$ *where* $(\boldsymbol{R},\boldsymbol{M}) \overset{(\hat{t},\alpha)}{\longrightarrow} (\boldsymbol{R'},\boldsymbol{M'})$.

We now extend the definition of 1-safe P/T-net to ERS. We introduce two conditions for safeness of ERS as a generalisation of the safeness notion for P/T-nets.

Definition 3.8 (1-safe ERS) *Let* $RS = (\hat{N},\Sigma,\ell,\omega,\ \boldsymbol{R^0})$*be an ERS. RS is 1-safe if and only if all reachable markings are 1-safe and if and only if in all reachable markings there is at most one net-token on each system net place and each net-token is 1-safe i.e.,:*

- $\forall(\boldsymbol{R},\boldsymbol{M}) \in R(RS), \forall \hat{p} \in \hat{P}$: $(R(\hat{p}),) \leq 1$ and
- $\forall(i,N_i,m_i) \in \boldsymbol{R}(\hat{p})$:$\forall p_i \in P_i : \forall \hat{p} \in \hat{P}\ (\boldsymbol{R}(\hat{p}),\Pi_i(\boldsymbol{M}(p_i)) > 0 \Longrightarrow \Pi_i(\boldsymbol{M}(p_i)) \leq 1.$

Observation 3.9: *Given an ERS if for all reachable markings there is at most one token on each system net place and each net-token is 1-safe, then all reachable markings are 1-safe.*

Theorem 3.10 *If an ERS is safe, then its set of reachable markings is finite. The proof to this theorem is presented in appendix A.*

4 Transformation of ERS into P/T- nets

We construct a behaviorally equivalent finite P/T-net model for the entire ERS model and show this by strong bisimulation equivalence between states of the two models. By doing so, we develop a set of transformation rules that provide the same behavioral properties as the original one for formal verification and analysis.

Related work can be found in (Miyamoto & Horiguchi, 2013; Lomazova & Ermakova, 2016). We highlight the similarities and differences between the proposed ap-

proach and these related studies. Miyamoto and Horiguchi present a translation technique for transforming classical Multi-Agent nets (MANs) into Modular Nets (MNs) and show isomorphism of state spaces of both nets including the computational complexity for transforming MAN into MNs. The major similarities between our work and that of (Lomazova&Ermakova, 2016) is that they developed a set of rules for translating a safe conservative nested Petri net (NP-net) into an equivalent P/T net. The main differences are that we established clearly an important relation between the isomorphic properties of state space of safe-ERS and a 1-safe P/T net. Among such results are the establish Lemmas, and proof of a theorem for the isomorphism. Moreover, we adopt a different way of introducing the procedure for transforming nets-within-nets into 1-safe P/T net, which consequently give a neater and easier-to-understand presentation.

4.1 Transformation Rules

This subsection gives a set of transformation rules for transforming Elementary Reference-net system (Section 3) into P/T-net. There exist five rules and they must be applied in sequence from Rule 1 to Rule 5. With these rules ERS can be translated into a P/T net system N^*.

Let $RS = (\widehat{N}, \Sigma, \ell, \omega, \boldsymbol{R^0})$*be an ERS with a set* Σ_{nt} *of all marked named net tokens in the initial marking. By* $\mathbb{R}$ *we denote the set of all names used in* Σ_{nt}*. The net will be translated into a P/T-net system* $N^* = (P^*_{N^*}, T^*_{N^*}, F^*_{N^*}, M^*_0)$

Rule 1: *Generate the set* $P^*_{N^*}$ *of places of a P/T-net* N^**. The first, is the set* P'_{N^*} *of places from the system net* $\widehat{N}$*, and the second the set* P_{N^*} *of all places of each net-token in the initial marking of the system net. Finally, we take the union of these set as the set* $P^*_{N^*}$ *of a target P/T-net* N^**, with the assumption that* $P'_{N^*} \cap P^*_{N^*} = \emptyset$*.*

P'_{N^*} *is generated by duplicating all places of the system net for each net-token name* i *used in the initial marking of the system net and labelled it with a pair* (p', i) *where* p'*is a place in* $\hat{P}$*. Thus the set is defined as follows:*

$$P'_{N^*} := \bigcup_{p' \in \hat{P}} \{(p', i) | i \in \mathbb{R}, i \geq 1\}. \quad (11)$$

P_{N^*} *is generated by taking a copy of each place in the set* P_i *for each net-token and labelled it with a pair* (p_i, i) *where* p_i *is a place in* P_i*. It is defined as follows:*

$$P_{N^*} := \bigcup_{i \in \Sigma_{nt}} \{(p_i, i) | p_i \in P_i, i \in \mathbb{R}, i \geq 1\}. \quad (12)$$

Therefore the set $P^*_{N^*}$ *of a target P/T-net* N^* *as shown in Fig.2 is the union of these set, namely*

$$P^*_{N^*} := P'_{N^*} \cup P_{N^*}. \quad (13)$$

Rule 2: *Define the initial marking for* N^**. For a P/T-net* N^* *we define an encoding of markings on places from the set of places* $\hat{P}$ *in an ERS by markings on the generated places from* $P^*_{N^*}$*. If a net-token with name* $i \in \mathbb{R}_i$ *resides in a place* $\hat{p}$ *in an initial*

marking $R^0(\hat{p})$ *of the system net, then a black token in placed on* $(\hat{p}, i) \in P^*_{N^*}$ *as the initial marking* M^*_0 *of the constructed, namely*

$$M^*_0(\hat{p}, i) = R^0(\hat{p}). \tag{14}$$

Also, we define an encoding of markings on places from the set of places P_i *on the generated places from* $P^*_{N^*}$. *If all places* (p, i) *for all* p *such that* $(p, i) \in P^*_{N^*}$ *is marked in the initial marking* M^0 *of the net-token* $i \in \mathbb{R}_i$, *then of black token is placed on* $(\hat{p}, i) \in P^*_{N^*}$ *in* M^*_0 ,*namely*

$$M^*_0(p, i) = M^0(p). \tag{15}$$

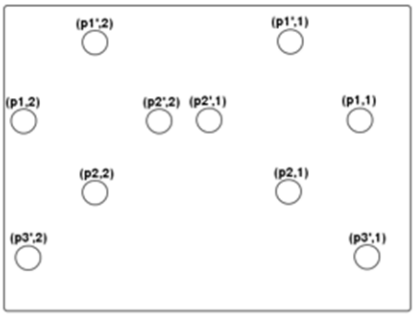


Fig. 2. Set of places of P/T net

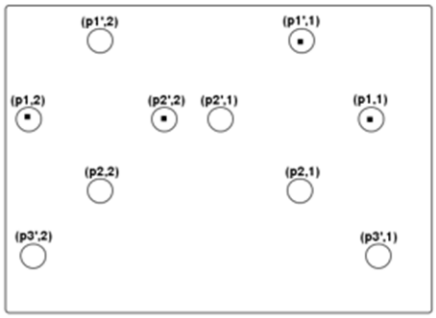


Fig 3: initial marking

If *a place in the system net is a place that contains a black token, then the unique copy corresponding to the place in* N^* *is also marked with a black token. In the given ERS, reference to the net-token* N_1 *resides in* $\hat{p}_1$, *and reference to the net-token resides in* $\hat{p}_2$. *Hence, we have tokens in* $(p'_1, 1)$ *and*$(p'_2, 2)$ *for* N^*. *Likewise, we define the markings for places* $(p_1, 1)$ *and* $(p_1, 2)$. *This is illustrated in Fig.3 above.*

Rule 3: *Generate a family of P/T-net transitions from a system net. We define a set* T^*_{sat} *of transitions of* N^* *obtained from each autonomous transition of the system net* $\widehat{N}$ *by duplicating each autonomous transition for each input arc variable of* $\hat{t}$ *that may be bound to any of the named net-token name in each place adjacent to* $\hat{t}$ *with appropriate input and output arcs, in* N^*.

$$T^*_{sat} := \bigcup_{\hat{t} \in \hat{T}} \{t'_{\beta_i(x)} | x \in w(\hat{t}) : \hat{t} \text{ is a system autonomous transition}\}. \tag{16}$$

In the example ERS, the set $w(\hat{t})$ *of input arc variables that can be bound to a named net-token for* t'_2 *is as follows:*

$$\beta(w(t'_2)) = \{\beta_1 = (z = 1) \quad \beta_2 = (z = 2)\}. \tag{17}$$

Where β_1 *and* β_2 *are bound to the input arc variable* z, *respectively. Therefore, two transitions* t'_{21} *and* t'_{22} *are generated for transition* t'_2 *from Rule 3.*

We define a set F^*_{sat} *of arcs for system autonomous transitions in* N^**as follows:*

$$F^*_{sat} = \bigcup_{\hat{a}\in\hat{F}}\{(x', y'|(x,y) = w(\hat{a}), x' \in P'_{N^*}(x) \cup T^*_{sat}(x), y' \in P'_{N^*}(y) \cup T^*_{sat}(y)\}. \quad (18)$$

Rule 4: *Generate a family of transitions representing autonomous transitions in each net-token. For a set T^*_{nat} of transitions of N^* we define a set of similar autonomous transitions as follows.*

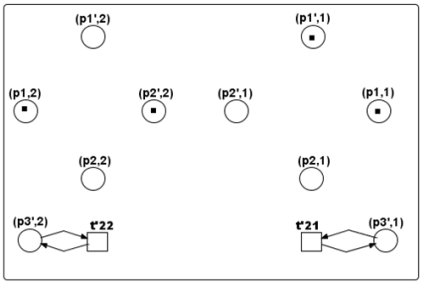


Fig. 4. Transitions and arcs from Rule 3

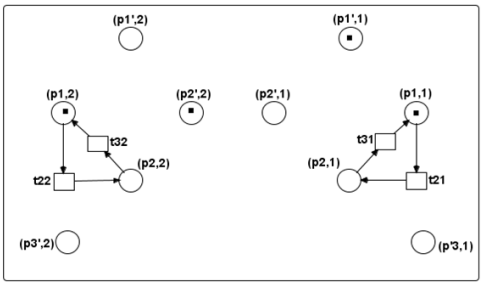


Fig. 5, Transitions and arcs after Rule 4

$$T^*_{nat} \coloneqq \bigcup_{i\in\Sigma_{nt}}\{t|t_i \in T_i \wedge t_i is\ an\ object\ autonomous\ transition\}. \quad (19)$$

We define a set F^*_{nat} of arcs of net-token autonomous transitions in N^*as follows:

$$F^*_{nat} = \{(p,i),t) \in P_{N^*} \times T^*_{nat}|(p,t) \in F_i > 0\} \cup$$
$$\{(t,(p.i) \in T^*_{nat} \times P_{N^*}|(t,p) \in F_i > 0\}. \quad (20)$$

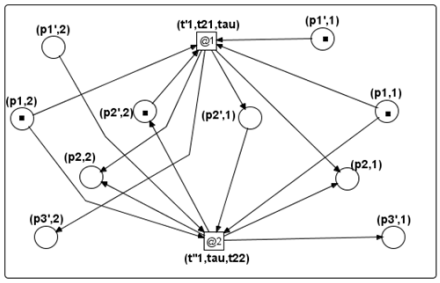
This is depicted in Fig.5.

Rule 5: *Generate a family of transitions representing synchronisation transitions obtained from the system net and net-tokens. An occurrence of a synchronous firing presumes simultaneous occurrence of a transition $\hat{t} \in \hat{T}$ with a set of transitions given by a binding β in system net, and some net-tokens transitions$(t_1, \dots, t_k) \in \ell$. This can be viewed as a combination of Rule 3 and Rule 4 with the condition that all involved transitions must be elements in the transition relation ℓ of an ERS.*

Transitions $(t_1, \dots, t_k)$ occur simultaneously with $\hat{t} \in \hat{T}$ of a system net, if $(\hat{t}, (t_i, \dots, t_k)) \in \boldsymbol{\ell}$. We generate synchronisation transitions from an ERS in a P/T-net N^* accordingly. This implies that we will have $|\ell|$ such transitions in N^*. Each of these transitions is composed of a system net transition $\hat{t} \in \hat{T}$, and some transitions of net-tokens that participate in synchronous firing of $\hat{t}$. They are defined as follows.

$$T^*_{sync_i} \coloneqq \bigcup_{i=1}^{k}\{t_{i.\beta_i(x)} = \{\hat{t}, t_1, \dots, t_k\}|x \in w(\hat{t}), \hat{t} \in \hat{T}, t_1 \in T_1, \dots, t_k \in T_k\}. \quad (21)$$

In our example two places $\hat{p}_1$ and $\hat{p}_2$ are marked with one net-token each in the initial marking. We add two transitions $t_1 = \{\{\hat{t}_1, t_{21}, \tau\}$ and $t_2 = \{\hat{t}_1, \tau, t_{22}\}$ annotated with @1 and @2, which is shown in Fig.6. The result of transforming ERS into P/T-net is shown in Fig. 7.



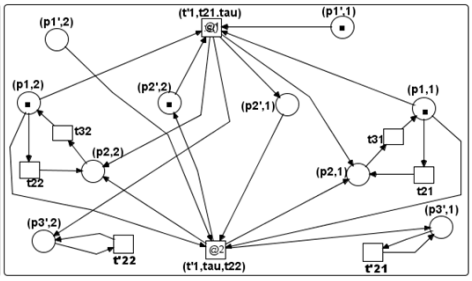


Fig. 6. Synchronous firing transitions and arcs

Fig. 7. Result of transforming ERS

5 Isomorphic Properties of the State Spaces

We establish an isomorphism between the states of an ERS and the generated 1-safe P/T-net. Recall that in Rule 2 we defined two separate initial markings for the P/T-net N^*: $M_0^*(\hat{p}, i)$ and $M_0^*(p, i)$. The former is an encoding of markings from the set of places $\hat{P}$ of the system net in an ERS and the latter is an encoding of markings from the set of places P_i of a net-token i. Likewise, we defined three sets of transitions: T_{sat}^*, T_{nat}^*, and $T_{sync_i}^*$ from Rule 3, Rule 4 and Rule 5 respectively in N^*. In the following, we define some mappings from the P/T-net to and ERS.

Definition 5.1 *A mapping $\hat{f}$ maps a marking M^* of a P/T-net N^* from the set of places $\hat{P}$ to markings R of a system net of an ERS as follows:*

$$\hat{f}(M^*)(\hat{p}, i) = R(\hat{p}) \; such \; that \; (\hat{p}, i) \in P_{N^*}^* : \hat{p} \in \hat{P} : i \in \mathbb{R} . \tag{22}$$

Definition 5.2 *A mapping f maps a marking M^* of a P/T-net N^* from the set of places P_i of net-token i of ERS to a marking M of a net-token of ERS as follows:*

$$f(M^*)(p, i) = M(p) \; such \; that \; (p, i) \in P_{N^*}^* : p \in P_i : i \in \mathbb{R} . \tag{23}$$

Definition 5.3 *$\hat{g}$ is a mapping that maps a transition $t'_{\beta_i(x)} \in T_{sat}^*$ of P/T-net N^* to a system-autonomous firing mode $(\hat{t}, \tau) \notin dom(\ell)$ of an ERS as follows:*

$$\hat{g}\left(t'_{\beta_i(x)}\right) = (\hat{t}, \tau) . \tag{24}$$

where $\beta_i(x)$ is a binding function that binds a variable $x \in w(\hat{t})$ on arcs adjacent to $\hat{t}$ to an object net name.

Definition 5.4 *g is a function that maps a transition $t \in T_{nat}^*$ of P/T-net N^* to an object-autonomous firing mode $(\tau, t_i) \notin dom(\ell)$ of an ERS as follows:*

$$g(t) = (\tau, t_i) . \tag{25}$$

Definition 5.5 *g_s is a mapping function that maps a transition $t_{i.\beta_i(x)} \in T_{sync_i}^*$ of P/T-net N^* to a synchronisation firing mode $(\hat{t}, t_1, \ldots, t_k) \in \ell$ of an ERS as follows:*

$$g_s\left(t_{i.\beta_i(x)}\right) = \{(\hat{t}, t_1, \ldots, t_k)\}\,. \tag{26}$$

The following lemmas related to $\widehat{N}$ and N^*constructed by Rules 1 to 5, hold.

Lemma 5.6 *For the initial marking at $\widehat{N}$ level, the following equality holds:*

$$R^0(\hat{p}) = \hat{f}(M_0^*)(\hat{p}, i)\,. \tag{27}$$

Lemma 5.7 *Suppose that $R = \hat{f}(M^*)$ and$(\hat{t}, \tau) = \hat{g}(t'_{\beta_i(x)})$. The following proposition holds:*

$$M^*[t'_{\beta_i(x)} > \Leftrightarrow R[(\hat{t}, \tau) >\,. \tag{28}$$

Lemma 5.8 *Suppose that $R_1 = \hat{f}(M_1^*)$, $M_1^*[t'_{\beta_i(x)} > M_2^*$, and $R_1[\hat{g}\left(t'_{\beta_i(x)}\right) > R_2$. The following equality holds: $R_2 = \hat{f}(M_2^*)$.* (29)

Lemma 5.9 *For the initial marking of the object net, the following holds:*

$$M^0(p) = f(M_0^*)(p, i)\,. \tag{30}$$

Lemma 5.10 *Suppose that $M = f(M^*)$ and$(\tau, t_i) = g(t)$. The following proposition holds:*

$$M^*[g(t) > \Leftrightarrow M[\left((\tau, t_i)\right) >\,. \tag{31}$$

Lemma 5.11 *Suppose that $M_1 = f(M_1^*)$, $M_1^*[t > M_2^*$, and $M_1[g(t) > M_2$. The following equality holds:*

$$M_2 = f(M_2^*)\,. \tag{32}$$

Lemma 5.12 *Suppose that $(R_1, M_1) = f_s(M_1^*)$ and $t_s = g_s\left(t_{i.\beta_i(x)}\right)$. The following proposition holds:*

$$M_1^*[g_s\left(t_{i.\beta_i(x)}\right) > \Leftrightarrow (R_1, M_1)[t_s >\,. \tag{33}$$

Lemma 5.13 *Suppose$(R_1, M_1) = f_s(M_1^*)$, $M_1^*[t_{i.\beta_i(x)} > M_2^*$ and $(R_1, M_1)[g_s\left(t_{i.\beta_i(x)}\right) > (R_2, M_2)$.*

The following equality holds:

$$(R_2, M_2) = f_s(M_2^*)\,. \tag{34}$$

From the above Lemmas, the following theorem holds.

Theorem 5.14 Let RS be a 1-safe ERS. Let also N*be a 1-safe P/T-net obtained from RS by the set of transformation Rules 1 to 5 above. Then state spaces of RS and N* are isomorphic.

Proof: Lemmas5.6 and 5.9 defines a one-to-one mapping between the initial markings of the 1-safe P/T-net $N^*$and the initial marking in RS. From Lemma 5.7 a system-autonomous firing mode $(\hat{t}, \tau)$ is enabled in a marking (R, M) if, and only if, the corresponding transition $t'_{\beta_i(x)}$ is enabled in the corresponding marking M^*. Also from Lemma 5.10 an object-autonomous firing mode (τ, t_i) is enabled in a marking

(R, M) *if, and only if, the corresponding transition* t *is enabled in the corresponding marking* M^*. *Again, from Lemma 5.12 a synchronous firing mode* $(\hat{t}, t_1, \ldots, t_k)$ *is enabled in a marking* (R, M) *if, and only if, the corresponding transition* $t_{i.\beta_i(x)}$ *is enabled in the corresponding* M^*. *Finally from Lemmas 5.8, 5.11 and 5.13, the generated markings in the 1-safe P/T-net can be mapped to the generated markings in the RS.* □

Thus we have shown that every ERS can be transformed to behaviourally equivalent 1-safe P/T-net. Hence the standard analysis techniques for 1-safe P/T-net can be applied for ERS.

6 Conclusion

While general elementary object systems (EOS) come with some constraints that limit their expressiveness for automatic verification purposes, in this paper a modification that relaxes these constraints was given: *elementary reference-net systems, ERS*. Also, we proposed a set of rules for transforming ERS to behaviourally equivalent 1-safe P/T nest. Furthermore, we established an important relationship between the isomorphic properties of state spaces of 1-safe ERS and 1-safe P/T net. Among such results are the established Lemmas, and the proof of a theorem which relates the state space of 1-safe P/T nets 1-safe ERS. The definition of elementary reference-net system, ERS, targets practical relevance and the use of a partial order (unfolding) approach for dynamic analysis of EOS. In future work, we aim to compare an unfolding of the transformed 1-safe P/T to a direct unfolding of a 1-safe ERS without computing an intermediate expansion.

References

Köhler, M. and Heitmann, F., 2009. On the expressiveness of communication channels for object nets. *Fundamenta Informaticae*, *93*(1-3), pp.205-219.

Köhler, M. and Rölke, H., 2004. Properties of object Petri nets. In Applications and Theory of Petri Nets 2004 (pp. 278-297). Springer Berlin Heidelberg.

Lomazova, I.A. and Schnoebelen, P., 1999, July. Some decidability results for nested Petri nets. In Perspectives of System Informatics (pp. 208-220). Springer Berlin Heidelberg.

Lomazova, I.A. and Ermakova, V.O., 2016 Verification of Nested Petri Nets Using an Unfolding Approach.

Miyamoto, T. and Horiguchi, K., 2013. Modular reachability analysis of Petri nets for multiagent systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 43(6), pp.1411-1423.

Valk, R., 1991. Modelling concurrency by task/flow EN systems. In 3rd Workshop on Concurrency and Compositionality (Vol. 191).

Valk, R., 2003. Object Petri nets: Using the nets-within-nets paradigm, Advanced Course on Petri Nets 2003 (J. Desel, W. Reisig, G. Rozenberg, Eds.), 3098.

Appendix A: Proof of Theorem 3.10

Proof. Let RS be a safe ERS. Let$m := |\hat{P}|$and $n := max\{|P_i| \mid (i, (P_i.T_i, F_i), m_i) \in \mathcal{N}\}$ be the number of system net places and the maximum number of places present in an object net, respectively.

By definition of safe ERS each net token is 1-safe and hence there are at most 2^n different markings a net-token may have. By definition of safe ERS each system net place is either marked or unmarked with a net-token with one of these markings, thus there are up to $(1 + 2^n)^m$ different markings of RS, i.e. $|R(RS)| \leq (1 + 2^n)^m$. □

Appendix B: Proof of Lemma 5.6

Proof: An initial marking of a system net in an ERS can be expressed by $R^0 = R^0(\hat{p}), \forall \hat{p} \in \hat{P}$. *By Rule 2,* $(\hat{p}, i) \in P^*_{N^*}$ *in the P/T-net has one token in the corresponding initial marking* $M^*_0(\hat{p}, i)$, *therefore* $M^*_0(\hat{p}, i) = R^0(\hat{p})$.

From Def. 5.1, $\hat{f}(M^*_0)(\hat{p}, i)$ *becomes* $\hat{f}(M^*_0)(\hat{p}, i) = R^0(\hat{p}) = R^0(\hat{p})$ □

Appendix C: Proof of Lemma 5.7

Proof: (⇒) *Suppose that* $t'_{\beta_i(x)} \in T^*_{sat}$ *is a transition that represents an autonomous transition in the P/T- net then* $(\hat{t}, \tau) \in \hat{T}$ *is a corresponding transition in the system net. From* $M^*[t'_{\beta_i(x)} >$ *and Def. 2.3, each place has at least* $W^*_{sat}((\hat{p}, i), t'_{\beta_i(x)})$ *tokens namely for each place* $(\hat{p}, i) \in P^*_{N^*}$, *the following inequality holds:*

$$M^*\big((\hat{p}, i)\big) \geq W^*_{sat}\left((\hat{p}, i), t'_{\beta_i(x)}\right). \tag{35}$$

Since $R = \hat{f}(M^*)$, *the number of token in place* $(\hat{p}, i)$ *equals the number of tokens in place* $\hat{p} \in \hat{P}$ *of a system net* $\hat{N}$:

$$M^*\big((\hat{p}, i)\big) = R(\hat{p}). \tag{36}$$

From Rule 3, the weight of the arc from $(\hat{p}, i)$ *to* $t'_{\beta_i(x)}$ *equals number of variables on the arc from* $\hat{p}$ *to* $\hat{t}$ *under the binding* β:

$$W^*_{sat}\left((\hat{p}, i), t'_{\beta_i(x)}\right) = \beta\big(w(\hat{p}, \hat{t})\big). \tag{37}$$

From (35), (36) & (37), for each place $\hat{p} \in \hat{P}$ *the following holds:*

$$R(\hat{p}) \geq \beta\big(w(\hat{p}, \hat{t})\big). \tag{38}$$

From Def. 3.5, $R[(\hat{t}, \tau) >$.

(⇐)*(38) holds since* $R[(\hat{t}, \tau) >$; *(36) & (37) also hold. Therefore, (35) holds. From Def. 2.3,* $M^*[t'_{\beta_i(x)} >$

Appendix D: Proof of Lemma 5.8

Proof: From Def. 2.3, the number of tokens in place $(\hat{p}, i)$ *in a successor marking* M_2^* *is expressed as follows:*

$$M_2^*(\hat{p}, i) = M_1^*(\hat{p}, i) - W_{sat}^*\left((\hat{p}, i), t'_{\beta_i(x)}\right) + W_{sat}^*\left(t'_{\beta_i(x)}, (\hat{p}, i)\right). \quad (39)$$

Since $R_1 = \hat{f}(M_1^*)$*, (30) holds. Similarly to (31), it holds that*

$$W_{sat}^*\left(t'_{\beta_i(x)}, (\hat{p}, i)\right) = \beta\big(w(\hat{t}, \hat{p})\big). \quad (40)$$

Therefore: $M_2^*(\hat{p}, i) = R_1(\hat{p}) - \beta\big(w(\hat{p}, \hat{t})\big) + \beta\big(w(\hat{t}, \hat{p})\big)$ *. (See Def. 3.5&* 36) (41)

Finally it holds that $R_2 = \hat{f}(M_2^*)$ *because (41) holds for each place.* □

Appendix E: Proof of Lemma 5.9

Proof: An initial marking of an object net in an ERS can be expressed by $M^0 = M^0(p), \forall p \in P_i,\ i \in \mathbb{R}$ *hold. Rule 2 says that place* $(p, i) \in P_{N^*}^*$ *in the P/T-net has one token in the corresponding initial marking* $M_0^*(p, i)$*, therefore* $M_0^*(p, i) = M^0(p)$.

From Def. 5.2, $f(M_0^*)(p, i)$ *becomes* $f(M_0^*)(p, i) = M^0(p)$ □

Appendix F: Proof of Lemma 5.10

Proof: (⇒) *Suppose that* $t \in T_{nat}^*$ *is a transition that represents an autonomous transition in the P/T- net then* $(\tau, t_i) \in T_i$ *is a corresponding transition in the object net. From* $M^*[t >$ *and the Def. 2.3, each place has at least* $W_{nat}^*((p, i), t)$ *tokens namely for each place* $(p, i) \in P_{N^*}^*$*, the following inequality holds:*

$$M^*\big((p, i)\big) \geq W_{nat}^*\big((p, i), t\big). \quad (42)$$

Since $M = f(M^*)$*, the number of tokens in*(p, i) *equals the number of tokens in* $p \in P_i$ *of an object net* N_i*:*

$$M^*\big((p, i)\big) = M(p). \quad (43)$$

From Rule 4, the weight of the arc from (p, i) *to* t *equals the weight of the arc from* p_i *to* t_i

$$W_{nat}^*\big((p, i), t\big) = W_i(p_i, t_i). \quad (44)$$

From (40) and (41), for each place $p \in P_i$ *the following inequality holds:*

$$M(p) \geq W_i(p_i, t_i). \quad (45)$$

From Def. 4.6, $M[(\tau, t_i) >$.

(⇐)*(45) holds since* $M[(\tau, t_i) >$*; (43) & (44) also hold. Therefore, (42) holds. From Def.2.3,* $M^*[t >$. □

Appendix G: Proof of Lemma 5.11

Proof: From Def. 2.3.2, the number of tokens in place (p, i) *in a successor marking* M_2^* *is expressed as follows:*

$$M_2^*(p,i) = M_1^*(p,i) - W_{nat}^*\big((p,i),t\big) + W_{nat}^*\big(t,(p,i)\big)\,. \tag{46}$$

Since $M_1 = f(M_1^*)$*, (43) holds. Similarly to (44), it holds that*

$$W_{nat}^*\big(t,(p,i)\big) = W_i(p_i,t_i)\,. \tag{47}$$

Therefore, the following equation holds:

$$M_2^*(p,i) = M_1(p_i) - W_i(p_i,t_i + W_i(t_i,p_i) = M_2^*(p,i) \text{ (See Def. 3.6)} \tag{48}$$

Finally it holds that $M_2 = f(M_2^*)$ *because (46) holds for each place.* □

Appendix H: Proof of Lemma 5.12

Proof: (⇒) *For* $\hat{t}$*, it can be proved in a similar way to Lemma 5.7 that*

$$\forall \hat{p} \in {}^\bullet\hat{t}\colon R(\hat{p}) \geq \beta\big(w(\hat{t},\hat{p})\big)\,. \tag{49}$$

For $(t_1,\dots,t_k)$ *it can be proven in a similar to Lemma 5.10 for each net-token transition* $t_i \in T_i$ *that*

$$\forall p_i \in {}^\bullet t_i\colon M_1(p_i) \geq W_i(p_i,t_i)\,. \tag{50}$$

From Rule 5, and equations (48) and (49) it holds that $(R_1, M_1)[t_s >$.

(⇐) *For* $t_{i.\beta_i(x)} \in T_{sync_i}^*$ *which is added in Rule 5, it can be shown that in a similar way to Lemma 5.7 that*

$$\forall (\hat{p},i) \in {P'}_{N^*}\colon M_1^*\big((\hat{p},i)\big) \geq W^*\big((\hat{p},i),\hat{t}\big)\,. \tag{51}$$

Similarly, it can be shown from Lemma 5.10 for $t_i \in T_i$ *that participate in* $t_{i.\beta_i(x)} \in T_{sync_i}^*$ *that*

$$\forall (p_i,i) \in P_{N^*}\colon M_1^*(p_i,i) \geq W_{nat}^*(p_i,t_i). \tag{52}$$

The action $(\hat{t},t_1,\dots,t_k)$ *share no input places by assumption in Rule 1. From Def. 2.3, (51) & (52):* $M_1^*[t_{i.\beta_i(x)} >$. □

Appendix I: Proof of Lemma 5.13

Proof: It can be proved in a similar way to Lemma 5.8 and 5.11 by Def. 2.3, and Rules 3 & 4. □

Context-dependent Lexical and Syntactic Disambiguation in Ontology Population *

Natalia Garanina and Elena Sidorova

A.P. Ershov Institute of Informatics Systems,
Lavrent'ev av., 6, Novosibirsk 630090, Russia
{garanina,lsidorova}@iis.nsk.su

Abstract. We suggest an approach to resolution of context-dependent lexical and syntactic ambiguity in a framework of ontology population from natural language texts. We show that a set of maximally determined ontology instances can be represented as a Scott information system with an entailment relation as a collection of information connections. Moreover, consistent primary lexical instances form FCA-concepts. These representations are used to justify correctness of lexical disambiguation and to define syntactic ambiguity and its resolution. This information system generates a multi-agent system in which agents resolve the ambiguity of both types.

1 Introduction

Ontological databases are currently widely used for storing information obtained from a great number of sources. To complete such ontologies, formalisms and methods that allow one to automate the process are developed. Features of automatic information retrieval cause ontology population ambiguities. In linguistics several kinds of ambiguities are considered: lexical, syntactic, semantic, and pragmatic [2]. In a process of ontology population from natural language texts we use our algorithms [5] in which the following ambiguity types appear: (1) several ontology instances or data attributes correspond to the same text fragment, (2) some value is incorrectly assigned to some attribute of some instance, (3) some value is incorrectly assigned to attributes of several instances, (4) some value is incorrectly assigned to several attributes of some instance, (5) several values are assigned to one-valued attribute of some instance. The first type corresponds to lexical ambiguity, and other types are syntactic ambiguity. An algorithm for lexical disambiguation was represented in [6]. In this work we suggest the modified algorithm for resolving lexical ambiguity, a new algorithm for syntactic disambiguation, and we justify the correctness of both of them.

In [6] we demonstrated that the process of retrieval of information in a form of a set of ontology instances can be presented as a Scott information system [13].

* The research has been supported by Russian Foundation for Basic Research (grant 15-07-04144) and Siberian Branch of Russian Academy of Science (Integration Grant n.15/10 "Mathematical and Methodological Aspects of Intellectual Information Systems").

This process produces maximally determined instances for ontology population. In this paper we prove that consistent sets of instances and lexical objects which values assign attributes of these instances form FCA concepts [3]. This fact grantees that information states of ambiguous conflicting agents do not intersect. This implies correctness of lexical disambiguation.

Besides, now we use a representation of ontologies which does not consider ontology relations as special structures. Only classes are allowed in these ontologies, and relations are represented as special attributes of classes. Well-known ontology representation language OWL uses the notation of this kind. This representation is a good solution for specification of polyadic relations. Our algorithms for ontology population are simpler with this representation because class and relation instances are packed in the same item.

Automatic techniques of disambiguation usually do not use an input data context in full. This can lead to incomplete and incorrect ambiguity resolution [1, 9, 8, 7]. Our approach tries to ease these drawbacks. For disambiguation we use a distributed approach. Every retrieved instance is related to agent. These agents detect and resolve ambiguities with help of a special master agent. This approach takes polynomial time for disambiguation.

The rest of the paper is organized as follows. In Section 2, an approach to ontology population in the framework of information systems is discussed. Section 3 describes lexical and syntactic disambiguation in terms of the system defined in the previous section. The next Section 4, gives definitions for a multi-agent system of context-dependent ambiguity resolution. Section 5 informally describes agents of our systems, their action protocols, and the main conflict resolution algorithm. In the concluding Section 6, directions of future researches are discussed.

2 Scott Information Systems in Ontology Population

Let we be given an ontology of a subject domain, the ontology population rules, semantic and syntactic model for a sublanguage of the subject domain and a data format, and input data as a finite natural language text with information for population of the ontology. We consider *ontology O of a subject domain* which includes (1) finite nonempty set C_O of classes for concepts of the subject domain, (2) a finite set of attributes with names in $Dat_O \cup Rel_O$, each of which has values in some data domain (data attributes in Dat_O) or is some instance of the ontology (relation attributes in Rel_O, which model relations), and (3) finite set D_O of data types. Every class $c \in C_O$ is defined by a tuple of typed attributes: $c = (Dat_c, Rel_c)$, where every data attribute $\alpha \in Dat_c \subseteq Dat_O$ has type $d_\alpha \in D_O$ with values in V_{d_α} and every relation attribute $\rho \in Rel_c \subseteq Rel_O$ is of class $c_\rho \in C_O$. Let a set of all values of all attribute be $V_O = \cup_{d_\alpha \in D_O} V_{d_\alpha}$. *Information content* IC_O of ontology O is a set of class instances, where every instance $a \in IC_O$ is of form (c_a, Dat_a, Rel_a), where c_a is a class of the instance, every data attribute in Dat_a has name $\alpha \in Dat_{c_a}$ with value(s) in V_{d_α} and every relation attribute in Rel_a has name $\rho \in Rel_{c_a}$ with a value as an instance

of class c_ρ. *Ontology population problem* is to compute an information content for a given ontology from given input data. Input data for the ontology population process are natural language texts. These data are finite and our algorithms of ontology-oriented text analysis can generate a finite set of ontology instances [5]. Finiteness of the set is guaranteed by prohibition for the rules from generating infinite information items by one position. We suggest to consider this process of forming ontology instances as work with Scott information systems. A *Scott information system* T is a triple $(T, Con, \vdash)$, where

– T is a set of tokens and $Fin(T)$ is a set of finite subsets;
– Con is a consistency predicate such that $Con \subseteq Fin(T)$, and
  1. $Y \in Con$ and $X \subseteq Y \Rightarrow X \in Con$,
  2. $a \in T \Rightarrow \{a\} \in Con$;
– $\vdash$ is an entailment relation such that $\vdash \subseteq Con \setminus \{\emptyset\} \times T$ and
  3. $X \vdash a \Rightarrow X \cup \{a\} \in Con$,
  4. $X \in Con$ and $a \in X \Rightarrow X \vdash a$,
  5. $\forall b \in Y : X \vdash b$ and $Y \vdash c \Rightarrow X \vdash c$.

The *information retrieval system* based on an ontology, finite input data, and rules of the ontology population and the data processing is defined as a triple $R = (A, Con, \vdash)$. Set of tokens A consists of a set of all (underdetermined) ontology *p-instances* formed by the rules in the determination process of initial p-instances which are retrieved from an input text by the special preprocess. Every p-instances $a \in A$ has form (c_a, Dat_a, Rel_a, P_a), where

– class $c_a \in C_O$, and
– every data p-attribute $\alpha_a \in Dat_a$ is of form (α, IV_α), where
  – name $\alpha \in Dat_{c_a}$, where
  – its information values $\bar{v} \in IV_\alpha$ has form $(v_{\bar{v}}, g_{\bar{v}}, s_{\bar{v}})$ with
    – data value $v_{\bar{v}} \in d_\alpha$, a set of all values of α is $Val_{\alpha_a} = \{v_{\bar{v}} \mid \bar{v} \in IV_\alpha\}$,
    – $g_{\bar{v}}$ is grammar information (morphological and syntactic features), and
    – $s_{\bar{v}}$ is structural information (position in input data);
– every relation p-attribute $\rho_a \in Rel_a$ is of form (ρ, O_{ρ_a}), where
  – a name $\rho \in Rel_{c_a}$, and
  – every $\bar{o} \in O_{\rho_a}$ has form (o, p_o), where
    – o is an instance of class c_{ρ_a}, and
    – $p_o \in P_o$ is its position,
  – a set of all relation objects of a is $O(Rel_a) = \{a\} \cup_{\rho_a \in Rel_a} \{o | \bar{o} \in O_{\rho_a}\}$;
– P_a is structural information (a set of positions in input data).

We consider a special set of tokens: a set of lexical objects LO corresponding to values of data attributes retrieved from input data. Every lexical object is a p-instance which has only a single data attribute with a single information value. P-instances correspond to ontology instances in a natural way. Let $a = (c_a, Dat_a, Rel_a, p_a)$ be p-instance, then its corresponding ontology instance is $a' = (c_a, Dat_{a'}, Rel_{a'})$, where every $\alpha \in Dat_{a'}$ has value(s) in Val_{α_a} and every $\rho \in Rel_{a'}$ has value o with $(o, p_o) \in O_{\rho_a}$. Further we omit prefix "p-" if there is no ambiguity. An *information order* relation $\prec$ is defined on ontology instances. Let $a, a' \in A$: $a \prec a'$, if $a = a'$ everywhere except for at least one attribute, with

the number of values of this attribute in a being strictly less than that in a'. For $x, x' \in A$: if $x \prec x'$, then x' is *information extension of* x.

Rules of ontology population and data processing $Rules = \{rule_1, \ldots, rule_n\}$ map finite sets of instances of ontology classes to an instance which is an informational extension of some instance of the domain set or a new instance. This sets must be linguistically and ontologically compatible: specified sets of their attributes and some instances have to satisfy conditions on values, grammatical and structural information [10]:
$rule_i : Dom_i \mapsto A$, $Dom_i \subseteq 2^A$, such that
$\forall X \in Dom_i \ : \ LingCons_i(\cup_{x' \in X} Dat_{x'} \cup Rel_{x'}) = true \wedge \forall x' \in X : c_{x'} \in Class_i$, where predicate $LingCons_i$ and set of classes $Class_i \subseteq C_O$ detect linguistic and ontological compatibility of the instance set, correspondingly. Let for $X \in Dom_i, x \in A$:
$rule_i(X) = x$ iff $((\exists y \in X : y \prec x \wedge c_x = c_y) \vee (\forall y \in X : y \not\prec x \wedge c_x = gen_i(X))) \wedge$
$(Dat_x = \emptyset \vee Dat_x = \cup_\alpha(\alpha, \cup\{(f_i(\bar{V_\alpha}), g_i(\bar{V_\alpha}), s_i(\bar{V_\alpha})) \mid$
$\exists Y_\alpha \subseteq X : dat_\alpha \subseteq \cap_{y \in Y_\alpha} Dat_y \wedge \bar{V_\alpha} = \cup_{\beta \in dat_\alpha}\{\bar{v} | \bar{v} \in IV_\beta\}\})) \wedge$
$(Rel_x = \emptyset \vee \forall o \in O(Rel_x) : o \in X \cup_{y \in X} O(Rel_y))$,
where $gen_i(X)$ generates a new class for a new instance, $f_i(\bar{V})$ produces a value based on values in $(\bar{V})$ for an attribute of instance x, and $g_i(\bar{V})$ and $s_i(\bar{V})$ inherit grammatical and structural information from set of information values $\bar{V}$.

Consistency predicate Con and entailment relation $\vdash$ correspond to the rules of ontology population and data processing. Let $x, x' \in A$ and $X \subseteq A$. The entailment relation connects informationally associated tokens:

– $X \vdash x$, iff $x \in X$, or $x \notin X \wedge$
$((\exists X' \subseteq X, X'' \subseteq A, rule_i \in Rules : rule_i(X' \cup X'') = x) \vee$
$(\exists x' \in A, X'' \subseteq A, rule_i \in Rules : X \vdash x' \wedge rule_i(\{x'\} \cup X'') = x))$,
i.e. instance x is entailed from X, if it is in this set, or information from tokens of this set is used for evaluating attributes of x.

The consistency predicate defines informationally consistent sets of tokens:

– $X \in Con$, iff for some $rule_i \in Rules$ holds $\forall X' \subseteq X(\cup_{x' \in X'} c_x \subseteq Class_i) \Rightarrow$ $(\exists x \in A, X'' \subseteq A : rule_i(X' \cup X'') = x)$, i.e. if there exists some rule which can find in a set of tokens some instances satisfying its class compatibility then these instances should be consistent with some other set of tokens with respect to the rule. Class compatible, but linguistically incompatible sets cannot be processed by rules, hence we do not consider them consistent.

Let us prove the following theorem for the system R:

Theorem 1. *Triple* $R = (A, Con, \vdash)$ *is a Scott information system.*

Proof. Let us show that the consistency predicate Con and the entailment relation $\vdash$ satisfy properties 1–5 of information systems.

1. $Y \in Con$ and $X \subseteq Y \Rightarrow X \in Con$. This fact follows from the definition of the consistency predicate directly because the condition of definition should hold for every subset of a consistent set.

2. $a \in A \Rightarrow \{a\} \in Con$. By the definition for every $rule_i \in Rules$ the class of a is not included in $Class_i$ or single $\{a\}$ can be complemented by some set of tokens in such a way that the $rule_i$ produces a new token.

3. $X \in Con$ and $X \vdash a \Rightarrow X \cup \{a\} \in Con$. $X \cup \{a\}$ is consistent because $a \in X$ or lexical information of a is inherited from X by definitions of the entailment relation and process rules.

4. $X \in Con$ and $a \in X \Rightarrow X \vdash a$. By the def. of the entailment relation.

5. $\forall b \in Y : X \vdash b$ and $Y \vdash c \Rightarrow X \vdash c$. Let $Y_b = Y \setminus \{b\}$. $X \vdash c$ iff $(\exists b \in Y, Y_b \subseteq A, rule \in Rules : X \vdash b \wedge rule(\{b\} \cup Y_b) = c))$ by the def. of the entailment relation.■

The proposition below directly follows from monotonicity of the entailment relation and finiteness of input data.

Proposition 1. *Information retrieval process of ontology population terminates.*

For token $x \in A$: $x^{\uparrow} = \{x\} \cup \{x' \mid x \prec x'\}$ and $x^{\downarrow} = \{x\} \cup \{x' \mid x' \prec x\}$ are *upper and down cones of* x. Let a set of *maximally determined* instances (maximal instances or tokens), which is the result of the analysis of input data, be $A^{\uparrow} = \{x \in A \mid x^{\uparrow} = \{x\}\}$. These instances may populate an ontology. Obviously,

Proposition 2. *Triple* $I = (A^{\uparrow}, Con, \vdash)$ *is a Scott information system.*

An information descendants of token $a \in A^{\uparrow}$ are all maximal tokens (all information) that can be obtained from this token by the entailment relation: $Ds(a) = \{x \in A^{\uparrow} | \{a\} \vdash x\}$. *An information ancestors* of token $a \in A^{\uparrow}$ are all maximal tokens from which a can be obtained: $An(a) = \{x \in A^{\uparrow} | \{x\} \vdash a\}$. In our framework for lexical objects the following equality holds: $An(a) = \{a\}$ because ontology instances are based on retrieved lexical objects. Information descendants are a particular case of Scott information states [14]. Like in the cited paper, we show that tokens from LO and their information descendants form a concept lattice.

Proposition 3. *Consistent sets of lexical objects form FCA concepts. Every consistent set of instances is a base for FCA concepts.*

Proof. Let every set x of information descendants of LO be an object, and every $l \in LO$ be an attribute. Lexical object l is an attribute of x iff $l \in x$. The extension of a set of attributes $L \subseteq LO$ is the set $L' = \{x | L \subseteq x\}$ and the intension of L' is the set $\{l | \forall x \in L', l \in x\}$. L is a concept iff the condition on the intension of the extension of L holds: $L = \{l | \forall x \in L', l \in x\}$ iff L is an information state of information system I iff L is a consistent set. The intension of a set of infostates X is the set $X' = \{l | \forall x \in X, l \in x\}$ and the extension of X' is the set $\{x | X' \subseteq x\}$. X is a concept iff the condition on the extension of the intension of X holds: $X = \{x | X' \subseteq x\}$ iff a set of all instances in set of infostates X forms an infostate too: $X^i = \{a \mid a \in x \in X\}$, hence X^i is a consistent set. Hence every consistent set of instances is a base for FCA concepts.■

3 Ambiguity and Resolution

(1) Lexical ambiguity.
Let $l, l' \in LO$ be in a conflict $l \leftrightsquigarrow l'$ iff $s(l) \cap s(l') \neq \emptyset$. Let set *AmbLO*

be a set of conflict lexical objects and Lex be a set of their descendants. We consider that rules in $Rules$ cannot generate instances which include inconsistent information. I.e. for every $rule_i \in Rules$ holds $\forall X \in Dom_i, a, a' \in X, l, l' \in An(a) \cap An(a') \cap LO : \neg(l \leftrightsquigarrow l')$. Hence for lexical objects l and l' in conflict: $Ds(l) \cap Ds(l') = \emptyset$.

For the lexical disambiguation of two conflicting lexical objects we prefer a lexical object which is more incorporated in an input text than its competitor. For $l, l' \in LO$ if $|Des(l)| > |Des(l')|$ we take l for evaluating attributes of ontology instances and ignore l'.

(2) Syntactic ambiguity.
Detection of syntactic ambiguity frequently requires analysis of homogeneous groups. Syntactic ambiguity is defined for ontology instances, not for lexical objects. Our types of syntactic ambiguity can depend on an ontology specification, hence here we consider syntactic-semantic ambiguity really. We omit "-semantic" for the brevity. Syntactic ambiguity usually can be expressed by corresponding single lexical object to several ontology items in various ways. For disambiguation it is necessary to find in an input text an evidence of correctness of the correspondence. This could be performed using the following inequalities.

For every instance a and its data attribute $\alpha \in Dat_a$ *a set of information values equal to* $v \in d_\alpha$ is $EQ(a, \alpha, v) = \{\bar{v} \in IV_\alpha \mid v_{\bar{v}} = v\}$. For every instance a and its relation attribute $\rho \in Rel_a$ *a set of relation objects with an instance equal to* $e \in c_{\rho_a}$ is $EQ(a, \rho, e) = \{(o, p_o) \in O_\rho \mid o = e\}$. A power of these sets is *an evidence power*. A triple $(a, \alpha, \bar{v})$ denotes information value $\bar{v} \in IV_\alpha$ of data attribute $\alpha \in Dat_a$ of instance a. A couple (a, ρ) denotes a value of relation attribute $\rho \in Rel_a$ of instance a. A set of *information values which effects on* $(a, \alpha, \bar{v})$ is $V(a, \alpha, \bar{v}) = \{(c, \gamma, \bar{w}) \mid \exists rule_i \in Rules, X \subseteq A^\uparrow : rule_i(X) = a \wedge c \in X \wedge \gamma \in dat_\alpha \cap Dat_c \wedge \bar{w} \in IV_\gamma \wedge \bar{v} = (f(\bar{V_\alpha}), g(\bar{V_\alpha}), s(\bar{V_\alpha}))\}$. A set of *instances which effects on* (a, ρ) is $I(a, \rho) = \{e \in A^\uparrow \mid \exists rule \in Rules, X \subseteq A^\uparrow : rule(X) = a \wedge e \in X \wedge O_\rho \cap O(Rel_e) \neq \emptyset\}$. Now we define a method of syntactic disambiguation.

(1) *Some value is incorrectly assigned to some attribute of an instance (Synt11).* An example: "The old men and women sat on the bench." The women may or may not be old. Hence, attribute "age" of instance "women" may not has value "old". Let a set of instances with ambiguity of this type be denoted as $Synt11$. Let in instance a information value $(c, \gamma, \bar{w})$ effect on $(a, \alpha, \bar{v})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v})$. Then in a case of the ambiguity, $(c, \gamma, \bar{w})$ is declared as effecting on $(a, \alpha, \bar{v})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > 1$. Let in instance a instance e effect on (a, ρ): $e \in I(a, \rho)$. Then in a case of the ambiguity instance e is declared as effecting on (a, ρ) iff $|EQ(a, \rho, e)| > 1$.

(2) *Some value is incorrectly assigned to attributes of several instances (Synt12).* An example: "Someone shot the maid of the actress who was on the balcony." Either the actress or the maid was on the balcony. Hence, either attribute "place" of instance "actress" or attribute "place" of instance "maid" may has value "balcony". Let a set of instances with ambiguity of this type be denoted as $Synt12$. Let in instances a and b information value $(c, \gamma, \bar{w})$ effect on $(a, \alpha, \bar{v})$

and $(b, \beta, \bar{u})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v}) \cap V(b, \beta, \bar{u})$. Then in a case of the ambiguity $(c, \gamma, \bar{w})$ is declared as effecting on $(a, \alpha, \bar{v})$ and not on $(b, \beta, \bar{u})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(b, \beta, \bar{u})|$. Let in instances a and b instance e effect on (a, ρ) and (b, o): $e \in I(a, \rho) \cap I(b, o)$. Then in a case of the ambiguity instance e is declared as effecting on (a, ρ) and not on (b, o) iff $|EQ(a, \rho, e)| > |EQ(b, o, e)|$.
(3) *A value is incorrectly assigned to several attributes of an instance (Synt112).* An example: "Cuban jazz band." A group of Cuban musicians performing jazz music or a group of musicians performing Cuban jazz. Hence, attribute "country" or attribute "style" of instance "band" may has value "Cuban". Let a set of instances with ambiguity of this type be denoted as $Synt112$. Let in instance a information value $(c, \gamma, \bar{w})$ effect on $(a, \alpha, \bar{v})$ and $(a, \beta, \bar{u})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v}) \cap V(a, \beta, \bar{u})$. Then in a case of the ambiguity $(c, \gamma, \bar{w})$ is declared as effecting on $(a, \alpha, \bar{v})$ and not on $(a, \beta, \bar{u})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(a, \beta, v_{\bar{u}})|$. Let in instance a instance e effect on (a, ρ) and (a, o): $e \in I(a, \rho) \cap I(a, o)$. Then in a case of the ambiguity instance e is declared as effecting on (a, ρ) and not on (a, o) iff $|EQ(a, \rho, e)| > |EQ(a, o, e)|$.
(4) *Several values are assigned to one-valued attribute of an instance (Synt211).* An example: "Shakespeare is an author of the piece." A gender of Shakespeare may be either male or female. Hence, one-valued attribute "gender" of instance "person" may has value either "male" or "female". Let a set of instances with ambiguity of this type be denoted as $Synt211$. Let in instance a information values $(b, \beta, \bar{u})$ and $(c, \gamma, \bar{w})$ effect on $(a, \alpha, \bar{v})$ and $(a, \alpha, \bar{v'})$, respectively: $(b, \beta, \bar{u}) \in V(a, \alpha, \bar{v})$ and $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v'})$. Then in a case of the ambiguity $(b, \beta, \bar{u})$ is declared as effecting on $(a, \alpha, \bar{v})$, and $(c, \gamma, \bar{w})$ is declared as not effecting on α iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(a, \alpha, v_{\bar{v'}})|$. Let in instance a instance e and e' effect on (a, ρ): $e, e' \in I(a, \rho)$. Then in a case of the ambiguity instance e is declared as effecting on (a, ρ), and e' is declared as not effecting on (a, ρ) iff $|EQ(a, \rho, e)| > |EQ(a, \rho, e')|$.

In a case of equalities of evidence powers the conflict is not resolved. We consider systems in which all these ambiguities are independent, i.e. pairwise intersections of sets Lex, $Synt11$, $Synt12$, $Synt112$ and $Synt211$ are empty. Informal description of action protocols for instance agents presents resolution of independent lexical and syntactic ambiguities. These protocols work correctly if resolution of references and detection of syntactic ambiguities are correct.

4 Multi-agent Ambiguity Resolution

Let a set of lexical objects which effect on some information value of data attribute α of instance a be $L(a, \alpha) = \{l \in LO \mid \exists rule_i \in Rules, X \subseteq A^{\uparrow} : rule_i(X) = a \wedge (\exists x \in X : x \in Ds(l) \wedge (\exists \beta \in dat_\alpha \cap Dat_x : l \in L(x, \beta)))\}$. For every $x \notin A^{\uparrow}$, the corresponding maximally determined instance is $\tilde{x}$ such that $\tilde{x} \in A^{\uparrow} \wedge x \prec \tilde{x}$. Entailment relation $\vdash$ generates *information connections* between maximally determined instances. Let $X \vdash x$ and $y \in X \wedge y \notin x^{\downarrow}$. Then

– *information connections* between $\tilde{y}$ and $\tilde{x}$ are
  – $\tilde{y} \xrightarrow{\tilde{\omega}} \tilde{x}$ iff $(\exists \alpha \in Dat_x, \beta \in Dat_y : \omega \in L(x, \alpha) \cap L(y, \beta)) \vee (\omega \in$

$O(Rel_x) \cap O(Rel_y)$

– of *updating type* $\tilde{y} \xrightarrow{\tilde{\omega}^u} \tilde{x}$ iff $\exists x' \in X \ : \ x' \prec x$,

– of *generating type* $\tilde{y} \xrightarrow{\tilde{\omega}^g} \tilde{x}$ iff $\nexists x' \in X \ : \ x' \prec x$.

An information system of information retrieval R generates a multi-agent system with typed connections. Agents of the system resolve the ambiguities by computing and comparing the context cardinalities and evidence powers. Information system $(A, Con, \vdash)$ generates *Multi-agent System of Ambiguity Resolution* (MASAR) as a tuple $S = (A, C, I, T)$, where

– $A = \{a_x \mid x \in A^{\uparrow}\}$ is a finite set of agents corresponded to maximally determined instances;

– $C = \{\tilde{\omega} \mid \exists x, y \in A : \tilde{x} \xrightarrow{\tilde{\omega}} \tilde{y}\}$ is a finite set of connections;

– mapping $I : C \longrightarrow 2^{A \times A}$ is an interpretation function of ordered connections between agents: $I(c) = (a_x, a_y)$ iff $\tilde{x} \xrightarrow{c} \tilde{y}$;

– mapping $T : C \times A \times A \longrightarrow \{gen, upd\}$ is types of connections: $T(c, a_x, a_y) = gen$ iff $I(c) = (a_x, a_y) \rightarrow (\tilde{x} \xrightarrow{c^g} \tilde{y})$, and $T(c, a_x, a_y) = upd$ iff $I(c) = (a_x, a_y) \rightarrow (\tilde{x} \xrightarrow{c^u} \tilde{y})$. Let *(conflict) lexical agents* correspond to (conflict) lexical objects.

Not every instance from $A^{\uparrow}$ is used for ontology population. There is a set of utility instances Utl. They do not resolve ambiguities or populate an ontology. They just transfer information to its descendants. Hence $A^{\uparrow} = LO \cup Ont \cup Utl$, where only instances from Ont may populate an ontology.

For every agent $a \in A$ we define the following sets of agents and connections. We omit symmetric definitions of ancestors $Anc*$ (for $Des*$) and utility predecessors $UtP*$ (for $UtS*$) for the brevity:

– $C_a = \{c \in C | \exists a' \in A : (a, a') \in I_C(c) \bigvee (a', a) \in I_C(c)\}$ is connections of a;

– $Scg_a^c = \{a' \in A \mid (a, a') \in I_C(c) \wedge T(c, a, a') = gen\}$ is a set of generated successors by c connection;

– $Scu_a^c = \{a' \in A \mid (a, a') \in I_C(c) \wedge T(c, a, a') = upd\}$ is a set of updated successors by c connection;

– $Sc_a^c = Scg_a^c \cup Scu_a^c$ is a set of all successors by c connection;

– $Pr_a^c = \{a' \in A \mid (a', a) \in I_C(c)\}$ is a set of predecessors by c connection;

– $UtS_a^c = \{a' \in Utl \mid (a, a') \in I_C(c)\}$ is a set of utility successors by c;

– $Des_a^c = Sc_a^c \cup \bigcup_{a' \in Sc_a^c} Des_{a'}^c$ is descendants by c connection;

MASAR is a multiagent system of information dependencies. In these systems agents can use information from predecessors and can pass the (processed) information to successors. Hence $Des_a^c \cap Anc_a^c = \emptyset$, i.e. every connection has no cycle because of information transfer.

A weight of an agent corresponds to the number and the quality (in a case of generation) of its non-utility ancestors and descendants. For every $a \in A$

– $wt_{Pr}^a(c) = 1 + \sum_{a' \in Pr_a^c} wt_{Pr}^{a'}(c)$ is *the weight of connection ancestors*,

– $wt_{Sc}^a(c) = 1 + \sum_{a' \in Scg_a^c} wt(a') + \sum_{a' \in Scu_a^c} wt_{Sc}^{a'}(c)$ is *the weight of connection descendants*,

– $wt_{Ut(P/S)}^a(c) = 1 + \sum_{a' \in Ut(P/S)_a^c} wt_{Ut(P/S)}^{a'}(c)$ is *the weight of connection utility ancestors/descendants*,

– $wt(a) = 1 + \sum_{c \in C_a} (wt_{Pr}^a(c) + wt_{Sc}^a(c) - (wt_{UtP}^a(c) + wt_{UtS}^a(c)))$ is *the weight*

of information agents.
Weight of system S is $wt(S) = \sum_{a \in Ont} wt(a)$.

Problem of conflict resolution in MASAR is to get a conflict-free MASAR of the maximal weight. A multiagent algorithm below produces such system.

5 Conflict Resolution in MASAR

In this paper we consider independent ambiguities only. In this case an order of their resolution is irrelevant. But it is naturally to resolve lexical ambiguity first, because this disambiguation effects on existence of ontology instances. Syntactic disambiguation refines distribution of information among instances.

Action protocols for conflict resolution used by MASAR agents form a multi-agent system of conflict resolution MACR. The system MACR includes a set of MASAR agents and an agent-master. Note, that a fully distributed version of our algorithm could be developed but it should be very ineffective. The result of agents' interactions by protocols described below is the conflict-free MASAR. All agents execute their protocols in parallel until the master detects termination. The system is dynamic because MASAR agents can be deleted from the system. The agents are connected by synchronous duplex channels. The master agent is connected with all agents, MASAR agents are connected with their successors and predecessors, and conflict lexical agents are connected too. Messages are transmitted via a reliable medium and stored in channels until being read.

For correct lexical disambiguation it is necessary to find groups of lexical agents which effect on weights of each other in a case of removing. Let us denote these *groups of relatives* as *Relatives*. Agents of groups from *Relatives* have common descendants: $\forall Rlt \in Relatives(\forall a \in Rlt(\exists b \in Rlt : Ds(a) \cap Ds(b) \neq \emptyset \wedge \forall c \in AmbLO \setminus Rlt : Ds(a) \cap Ds(c) = \emptyset))$. Due to the mutual effect of relatives on their weights it is necessary to resolve conflicts between groups of relatives. Let $GR_1 \subseteq Rlt_1$ and $GR_2 \subseteq Rlt_2$, where $Rlt_1, Rlt_2 \in Relatives$. *Relative groups* GR_1 *and* GR_2 *are in a conflict* $GR_1 \leftrightsquigarrow GR_2$ iff $(\forall a \in GR_1 \exists b \in GR_2 : a \leftrightsquigarrow b) \wedge (\forall b \in GR_2 \exists a \in GR_1 : b \leftrightsquigarrow a)$. Let sets $G_1 = \cup_{i=1}^{n} GR_1^i = \cup_{i=1}^{m} Rlt_1^i$ and $G_2 = \cup_{i=1}^{n} GR_2^i = \cup_{i=1}^{m} Rlt_2^i$, where $Rlt_1^i, Rlt_2^i \in Relatives$ for every $i \in [1..m]$ be *groups of friends*. These *groups of friends are in a conflict* iff $(\forall i \in [1..n] : GR_1^i \leftrightsquigarrow GR_2^i)$. Note that due to proposition 3 set *AmbLO* can be disjoined to nonintersecting subsets of relatives. *A conflict is resolved* for a benefit of the group with the greater weight, i.e. if $\sum_{a \in G_1} wt(a) > \sum_{b \in G_2} wt(b)$, then agents of group G_2 are removed from the system, and their descendants delete their inherited values of attributes or the descendant is removed itself if the lexical value from a lexical agent in G_2 is generating for this descendant.

Hence, for resolving all conflicts in the system it is necessary to perform the following steps: (1) to compute weights of agents, (2) to detect relative groups, (3) to compute independent conflict groups of friends, (4) to resolve lexical conflicts between the groups, (5) to make the corresponding change in the system, and (6) to resolve all kinds of syntactic ambiguity. An agent-master coordinates MASAR agents. It computes conflict groups and detects agents to be

removed. All other activities are performed by MASAR agents asynchronously. Due to parallel execution all computations take polynomial time.

(1) An interface protocol for system agents

This protocol specifies agent's reactions for incoming messages. These messages include information which actions should be performed by the agent: (1) `Start`: to start; (2) `CompWeight`: to compute its weight; (3) `FindRlt`: to find relatives; (4) `Remove`: to remove connections or itself; (5) `ResSynt*`: to resolve some syntactic ambiguity. Until an input message causes an agent to react the agent stays in a wait mode. Messages for an agent are stored in its input channel.

(2) The main algorithm for conflict resolution

Let us give an informal description of protocol `Master`. First, the agent-master computes set of lexical agents LO, then it finds set of conflict lexical agents $AmbLO$. After that it sends `Start` to all agents and launches parallel computing agents' weights and finding relatives for conflict lexical agents. After all agents finish their job, the master computes conflict groups of relatives, then detects conflict groups of friends. By comparing weights of conflict groups of friends, it forms a list of agents to be removed. After finishing of this resolution of group conflicts, the master launches the corresponding system changes. After termination of the changing, it initiates all kinds of syntactic disambiguation for instance agents in parallel.

Below we give informal descriptions of several protocols of the system agents.

(3) Computing agents' weight

Following the definitions of the weights agent a computes in parallel weights of (utility) descendants and (utility) ancestors by every connection $c \in C_a$, launching the corresponding subprocesses for each $c \in C_a$. These non-utility subprocesses send the weights of their descendants (ancestors) increased by 1 to predecessors (successors) respectively. Utility subprocesses do not increase the weights. If connection c is of type *gen* then the corresponding descendants' subprocesses send the weight of a to the predecessors. When these parallel computations are finished, the agent computes its own weight. The protocol of weights computing belongs to the class of wave echo algorithms [12].

(4) Computing agents' relatives

Let agents from $AbmLO$ be numerated. Computing relatives consists of two stages. Agents act asynchronously. (1) Pairwise search. Elder agent a using id of every younger agent b sends couple of ids $(a.id, b.id)$ to its descendants via its successors. If some descendant of a finds both numbers among its connections then it returns to a the id of b. After receiving agent a adds b to set of its relatives. Termination of this computation can be detected by AB-algorithm from [4]. (2) Merging. Elder agent a sends a request to every younger agent b for a set of its relatives $b.Rlt$. If $a.Rlt \cap b.Rlt \neq \emptyset$, then agent a merges both sets and agent b removes its set of relatives and stops its computation. After termination of the computation there are several agents with nonempty sets of relative groups.

(5) Removing LO-agents from the system

If agent a has to be removed from the system, then (1) all its predecessors remove all connections with it and delete a from sets of successors; (2) its descendants

remove a) all connections with it, b) the corresponding predecessors c) the corresponding attribute value; and d) if the removing connection is of generating type then the descendant has to be removed from the system.

Resolution of syntactic ambiguities Synt11 and Synt12 consists of two steps.

(6) Synt11 resolution.

(1) Ambiguity detection. Every agent, using sets of its successors, checks if some attribute value effects on values of several instances. If yes and these instances form a homogeneous group, and satisfy a predefined grammar condition, then it sends a message with the type of the conflict and the conflict value to every agent in the group excluding the first agent in the group. (2) Agents in the group resolve the ambiguities following the resolving formulas for Synt11. For this they compute an evidence power of the ambiguous value.

(7) Synt12 resolution.

(1) Ambiguity detection. Every agent, using sets of its successors, checks if some attribute value effects on values of several instances. If yes and these instances do not form a homogeneous group, and satisfy a predefined grammar condition, then it sends a message with the type of the conflict, the conflict value, and ids of the competitors to every agent in the group. (2) Agents in the group send their evidence power to the competitors. Then they resolve the ambiguities following the resolving formulas for Synt12.

(8) Synt112 resolution.

If an agent finds attributes ω_1 and ω_2 with value c then it compares evidence powers $EQ(a, \omega_1, c)$ and $EQ(a, \omega_2, c)$. The attribute value is removed from values of an attribute with the less power.

(9) Synt211 resolution.

If an agent finds attribute ω with values c_1 and c_2 then it compares evidence powers $EQ(a, \omega, c_1)$ and $EQ(a, \omega_1, c_2)$. The attribute value with the less power is removed from values of the attribute.

6 Conclusion

In this paper, we show that maximal instances of the ontology classes that take part in the process of population form, together with the rules of data processing and ontology population, a Scott information system. This result justifies resolution of context-dependent lexical ambiguity by calculating context cardinalities. The Scott information system is also a basis for our approach to syntactic context-dependent ambiguity resolution. This system generates a multi-agent system in which agents resolve the ambiguities by computing the cardinality of their contexts and evidence powers. The suggested algorithm of lexical ambiguity resolution chooses the most powerful group of agents and removes their competitors. The choice is based on agents' weights and their effect on the system.

We considered independent lexical and syntactic ambiguities only. In the near future we plan to study disambiguation of combination of various types syntactic and lexical ambiguities. In this work it is useful to introduce a membership

probability of attribute ambiguity values and a degree of their effect on other instances. We would like to try the developed technique for resolving references also.

References

1. **Alfawareh H.M., Jusoh S.** *Resolving Ambiguous Entity through Context Knowledge and Fuzzy Approach* // International Journal on Computer Science and Engineering (IJCSE). ISSN: 0975-3397, Vol. 3, No. 1, 2011. pp. 410–422
2. **Berry, D.M., Kamsties, E., Krieger, M.M.** *From contract drafting to software specification: Linguistic sources of ambiguity (2003)*, http://se.uwaterloo.ca/dberry/handbook/ambiguityHandbook.pdf (31.01.2016)
3. **Ganter B., Wille R.** *Formal Concept Analysis.* Mathematical Foundations. Springer Verlag, 1996.
4. **N. O. Garanina, E. V. Bodin.** *Distributed Termination Detection by Counting Agent* // Proc. of the 23nd International Workshop on Concurrency, Specification and Programming (CS&P 2014), Chemnitz, Germany, 29. September - 01. Oktober 2014. Humboldt-Universitat zu Berlin, 2014, pp. 69–79.
5. **Garanina N., Sidorova E., Bodin E.** *A Multi-agent Approach to Unstructured Data Analysis Based on Domain-specific Onthology* // Proc. of the 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, Sept. 25-27, 2013. CEUR Workshop Proceedings, Vol. 1032, pp. 122–132
6. **Garanina N., Sidorova E.** *An Approach to Ambiguity Resolution for Ontology Population* // Proc. of the 24th International Workshop on CS&P. Rzeszow, Poland, Sep. 28-30, 2015. – University of Rzeszow, 2015, Vol. 1, pp. 134–145.
7. **Gleich B., Creighton O., Kof L.** *Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources* // Proc. of 16th International Working Conference Requirements Engineering: Foundation for Software Quality, Essen, Germany, June 30–July 2, 2010, LNCS Vol. 6182, pp. 218-232.
8. **Kim D.S., Barker K., Porter B.W.** *Improving the Quality of Text Understanding by Delaying Ambiguity Resolution* // Proc. of the 23rd International Conference on Computational Linguistics, Beijing, 2010. pp. 581–589
9. **Navigli R.** *Word sense disambiguation: a survey.* ACM Computing Surveys, 41(2):1–69, 2009.
10. **Sidorova E., Kononenko I., Zagorulko Yu.** *Knowledge-based approach to document analysis* // International Journal "Information Technologies and Knowledge". Vol.2, No. 1, 2008. pp. 17–22.
11. **Spasic I., Zhao B., Jones C., Button K.** *KneeTex: an ontology-driven system for information extraction from MRI reports.*// J. Biomedical Semantics, 6, 2015, p. 34.
12. **Tel G.** *Introduction to Distributed Algorithms.* Cambridge University Press, 2000.
13. **Winskel G.** *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, 1993.
14. **Zhang G.-Q.** *Chu Spaces, Concept Lattices, and Domains* // Electronic Notes in Theoretical Computer Science (ENTCS). Vol. 83, Jan 2013, pp. 287–302

Semantic Rendering of Data Tables: Multivalued Information Systems Revisited

Marcin Wolski[1] and Anna Gomolińska[2]

[1] Maria Curie-Skłodowska University, Department of Logic and Cognitive Science,
Pl. Marii Curie-Skłodowskiej 4, 20-031 Lublin, Poland
marcin.wolski@umcs.lublin.pl
[2] University of Białystok, Faculty of Mathematics and Informatics,
Konstantego Ciołkowskiego 1M, 15-245 Białystok, Poland
anna.gom@math.uwb.edu.pl

Abstract. Data tables provide a convenient means of representation of descriptive information about objects. They serve also as standard input for data analysis tools or theories. In this paper we focus our attention upon the special class of data tables, namely multivalued information systems introduced by Z. Pawlak and E. Orłowska in the early 80s. The main idea presented in the paper is to interpret multivalued information systems as semantically processed single valued data tables. This interpretation allows us to describe classical rough set theory, dominance-based rough set theory, and formal concept analysis within the framework of multivalued information systems.

Keywords: information system, rough set, dominance relation, formal concept

1 Introduction

Data tables provide a simple and effective means of representation of collected pieces of information about a given set of objects. They serve also as standard input for data analysis tools or theories, output of which is often referred to as *knowledge*. In the present paper we shall focus our attention upon the special class of data tables, namely multivalued/approximate/nondeterministic information systems, introduced by Z. Pawlak and E. Orłowska in the early 80s [3, 4]. These systems are generalisations of the standard data tables/information systems to the case in which for an object x and an attribute A we are given (as the entry in the table) a set V_A of attribute values instead of a single value. The formal definitions of both multivalued and approximate information systems are actually the same (see [4]); it is the interpretation/semantics of the entries of these tables which makes the difference: the first interpretation is *the object x has all values from V_A for the attribute A* (multivalued systems), whereas the second reads as *the object x has a single value from the set V_A for the attribute A* (approximate systems). These systems equipped with a generalised semantics (which reads *for the object x and the attribute A the set V_A provides some possible values*) are called by Z. Pawlak and E. Orłowska nondeterministic information systems [3]. Of course, the semantics of the entries in the table determines how information is further processed; in other words, which relations between objects are used to construct information granules

being the building blocks of knowledge. The main concern of [4] is the informational indiscerniblity between objects (an equivalence relation), whereas the main focus of [3] are the information inclusion and the informational connection (a preorder and a tolerance relation, respectively).

The main novelty of the present paper consists in taking multivalued information systems as semantically enriched single valued data tables; this idea can be summarised by the following equation:

$$\text{data table} + \text{semantics} = \text{multivalued information system}.$$

Thus multivalued information systems represent semantically processed data. We start our study with some standard data tables used in the leading theories of data analysis: single valued information systems from rough set theory (RST) [4–6], single valued information systems enriched by dominance relations taken from dominance based rough set theory (DRS) [7, 8], and formal contexts from formal concept analysis (FCA) [1, 9]. Then we provide these structures with some specific interpretation/semantics. To this end we use *scales* from FCA, which are tools to convert a multivalued formal context (which is actually a standard information system) into a (single valued) formal context. We are going to employ these scales in order to obtain multivalued information systems. Then we focus upon informational relations of indiscerniblity and inclusion. These steps allow us to consider RST, DRS, and FCA within a single conceptual framework of multivalued information systems and to emphasise how these theories differ semantically. Finally, we shall discuss different interpretations of RST, DRS, and FCA based operators in the context of John Stuart Mill inductive reasoning [2].

2 Data Tables and Semantics

In the present section we discuss different forms of data tables considered in the leading theories of data analysis: rough set theory (RST) [4–6], dominance based rough sets (DRS) [7, 8], and formal concept analysis (FCA) [1, 9]. Of course, apart from data tables (input), each theory provides special tools to process the tables and produce some meaningful output. However, in the present section we shall discuss only tables, whereas the ways they may be further processed will be presented in the next section.

Definition 1 (Formal Context [1, 9]). *A formal context is a triple* (U, Att, R)*, where* U *is a set of objects,* Att *a set of binary attributes and* $R \subseteq U \times Att$*.*

If an object x stands in the relation R to A, then we mark it in the data table by ✘. Table depicted by Fig. 1 presents a very simple context; Bob is a good mathematician whereas Agnes is not; on the bright side, she is rich.

Definition 2 (Information System [3, 4]). *A quadruple* $\mathcal{I} = (U, Att, Val, f)$ *is called an* information system*, where:*

- U *is a nonempty finite set of objects,*
- A *is a nonempty finite set of attributes,*

	good mathematician	rich
Steven	✘	
Bob	✘	✘
Agnes		✘

Fig. 1. A simple formal context

- $V = \bigcup_{A \in Att} Val_A$, *where* Val_A *is the value–domain of the attribute* A*, and* $Val_A \cap Val_B = \emptyset$*, for all* $A, B \in Att$ (*the last condition is not necessary, yet it is mathematically convenient*),
- $f : U \times Att \to Val$ *is an information function, such that for all* $A \in Att$ *and* $x \in U$ *it holds that* $f(x, A) \in Val_A$.

If f *is a partial function then the information system* $\mathcal{I}$ *is called* incomplete. *If the codomain of* f *is the powerset of* Val*, then the system is called* multivalued.

In what follows we shall confine our attention to the complete information (a simple example is depicted by Fig. 2) systems and their multivalued version being the result of scaling.

	mathematician (1-6)	rich
Steven	5	$ 0.1 million
Bob	4	$ 0.8 million
Agnes	2	$ 1 million

Fig. 2. An information system

Definition 3 (Dominance-Based Data Table [7, 8]). *A dominance-based data table is a system* $\mathcal{I} = (U, Att, Val, f, \leq_D)$*, where* (U, Att, Val, f) *is an information system such that all attribute values are real numbers. Let us define:* $x \leq_A y$ *iff* $f(A, x) \leq f(A, y)$*, for* $x, y \in U$*. Then* $x \leq_D y$ *iff* $x \leq_A y$ *for all* $A \in Att$.

Thus, information systems allow one to be more specific about the meaning of attributes. In the case of dominance-based data tables, we additionally assume that the attribute values are comparable with respect to some complete order. If $f(A, x) \leq f(A, y)$, then we say that y *is at least as good as* x *with respect to* A. If y is at least as good as x with respect to all attributes then we say that y *dominates* x.

Of course, any information system may be converted into a formal context. The easiest way of doing this is called in FCA a *nominal scaling*. Formally, a scale for A is a formal context (Val_A, Val_A, R_A) having Val_A as both the set of objects and the set of attributes. We also assume that the identity id_{Val_A} is included in R_A. After scaling each pair attribute-value (A, v) is regarded as a separate attribute of the new context $C_I = (U, \{(A, v)\}_{A \in Att\ v \in Val_A}, R)$. For the fundamental relation in rough set

theory is the indistinguishability, the values in (Val_A, Val_A, R_A) are compared by the equality relation: R_A is $=$, for every A; and in consequence 1 is interpreted as 1, 2 as 2, and so on. That is why in the scale depicted in Fig. 3 only the diagonal is marked by ✘. This type of scaling is called *nominal*. However, since all values in our exemplary data

	1	2	3	4	5	6
1	✘					
2		✘				
3			✘			
4				✘		
5					✘	
6						✘

	$ 0.1	$ 0.8	$ 1
$ 0.1	✘		
$ 0.8		✘	
$ 1			✘

	mathematician						rich		
	1	2	3	4	5	6	$ 0.1	$ 0.8	$ 1
Steven					✘		✘		
Bob				✘				✘	
Agnes		✘							✘

Fig. 3. Nominal scaling: the common ground of FCA and RST

table are real numbers, we can compare them with respect to $\leq$ instead of $=$. Actually, when we order these values according to $\leq$ we do change the scaling, or better still, the semantics (interpretation) of attribute values. Now, e.g., Bob's score in physics 5 and Agnes's score 2 means that Bob is a better mathematician than Agnes. In other words, whatever Agnes can solve, Bob can as well. Following DRS, we may say that Bob is at least as good as Agnes with respect to the attribute *mathematician*. Formally, the higher value (e.g., 5) with respect to $\leq$ implies the lower value (e.g., 2). Such scales are called *ordinal scales*. Thus this time the mark ✘ on 5 may mean that Bob has solved at least 5 problems. Under this reading Bob has solved at least 4 problems too. Therefore the scaling representing this interpretation (semantics) may be defined as depicted in Fig. 4. Please note that the values $ 0.1 or $ 0.8 are interpreted in the same fashion. Thus ✘ on $ 0.8 means that Bob has at least $ 0.8 on his bank account, and in consequence he has at least $ 0.1 too. Following DRS, we may say that Bob dominates Steven. In consequence, both RST and DRS start with the same date table but use different scales (interpretations). Of course, there are also (many) other possibilities. We conclude this

	1	2	3	4	5	6
1	✘					
2	✘	✘				
3	✘	✘	✘			
4	✘	✘	✘	✘		
5	✘	✘	✘	✘	✘	
6	✘	✘	✘	✘	✘	✘

	$ 0.1	$ 0.8	$ 1
$ 0.1	✘		
$ 0.8	✘	✘	
$ 1	✘	✘	✘

	mathematician						rich		
	1	2	3	4	5	6	$ 0.1	$ 0.8	$ 1
Steven	✘	✘	✘	✘	✘		✘		
Bob	✘	✘	✘	✘			✘	✘	
Agnes	✘	✘					✘	✘	✘

Fig. 4. DRS in terms of formal contexts

part with a nontrivial interpretation offered by B. Ganter during his seminar at Warsaw

University (a few years ago). He suggested to read the exam scores in a natural language and take the "natural" scale. Under this reading, someone who has done a very good

1 may be interpreted as *bad*
2 may be interpreted as *unsatisfactory*
3 may be interpreted as *satisfactory*
4 may be interpreted as *good*
5 may be interpreted as *very good*
6 may be interpreted as *excellent*

	1	2	3	4	5	6
1	✘	✘				
2		✘				
3			✘			
4			✘	✘		
5			✘	✘	✘	
6			✘	✘	✘	✘

Fig. 5. B. Ganter's semantics of exam's scores and scaling

job has done also a good job. The job of course is also satisfactory. However, someone who has done a bad job has also done an unsatisfactory job, but not *vice versa*. In consequence we obtain yet another scale, as depicted by Fig. 5. In this scale we use two orderings, and this type of scaling is therefore called *bi-ordinal*. Thus, starting from the information system depicted by Fig. 2 we can obtain a number of different multivalued information systems, depending on which scale is applied to the original system (see Fig. 6).

Nominal scaling $\Rightarrow$

	mathematician (1-6)	rich
Steven	$\{5\}$	{\$0.1 million }
Bob	$\{4\}$	{\$0.8 million }
Agnes	$\{2\}$	{\$1 million }

Ordinal scaling $\Rightarrow$

	mathematician (1-6)	rich
Steven	$\{1, 2, 3, 4, 5\}$	{\$0.1 million }
Bob	$\{1, 2, 3, 4\}$	{\$0.1 million , \$0.8 million }
Agnes	$\{1, 2\}$	{\$0.1 million , \$0.8 million \$1 million }

B. Ganter's scaling $\Rightarrow$

	mathematician (1-6)	rich
Steven	$\{3, 4, 5\}$	{\$0.1 million }
Bob	$\{3, 4\}$	{\$0.1 million , \$0.8 million }
Agnes	$\{2\}$	{\$0.1 million , \$0.8 million \$1 million }

Fig. 6. Multivalued information systems as the results of scaling, i.e. providing data tables with interpretation/semantics

More formally, different interpretations/semantics of attribute values lead to different formal contexts. When one starts with a multivalued context (an information system)

$\mathcal{I} = (U, Att, Val, f)$ and a set of scales $S = \{(Val_A, Val_A, R_A) : A \in Att\}$ (as discussed above), then every pair (A, v), where $A \in Att$ and $v \in Val_A$, is regarded as the attribute in the induced formal context $C_I = (U, \{(A, v)\}_{A \in Att\ v \in Val_A}, R)$, where R is defined by

$$R = \{(x, (A, v)) : v \in f_s(x, A)\},$$

$$f_s(x, A) = \{v_i \in Val_A : f(x, A) = v \ \&\ (v, v_i) \in R_A\}.$$

Of course every (multivalued) context $\mathcal{I} = (U, Att, Val, f)$ may also be converted into a multivalued information system $\mathcal{I}_S = (U, Att, Val, f_s)$. The whole process of providing $\mathcal{I}$ with semantics given by S is depicted by Fig.6.

3 Information Processing: Approximation Operators

In the previous section we have discussed information systems (collected pieces of data about objects) which must be further processed to give some meaningful output (knowledge). The information processing in rough set theory [4–6] and dominance based rough set theory (DRS) [7, 8] is done by means of binary relations between objects. However, formal concept analysis (FCA) [1, 9] is based upon a relation between objects and attributes; in some special cases this relation may be reduced to the relation between objects, but it is not always possible. In multivalued information systems, in contrast to the classical information systems where there is considered a single relation, there are three important relations between objects [3, 4].

Definition 4. *Let (U, Att, Val, f) be a multivalued information system; then one can define:*

- *Informational Indiscerniblity: $x\ Ind\ y$ iff $f(x, A) = f(y, A)$,*
- *Informational Connectivity (Similarity): $x\ Sim\ y$ iff $f(x, A) \cap f(y, A) \neq \emptyset$,*
- *Informational Inclusion: $x\ Incl\ y$ iff $f(x, A) \subseteq f(y, A)$,*

for all $A \in Att$ and $x, y \in U$.

Usually, the output of data analysis is related to some aspect of reality represented by a *decision attribute*. For example, we could take the *subjective quality of life* as the decision attribute. Then our aim would be to "express" the subjective quality of life in terms of the attributes *mathematician* and *rich*, which are (in this case) called *conditional attributes* (we would like to obtain knowledge how the subjective quality of life "depends" on wealth and education in mathematics). Information systems having a single attribute distinguished as a decision attribute are called *decision tables*. In such a case, all informational relations (indiscerniblity, connectivity, and inclusion) are defined with respect to (only) conditional attributes.

As earlier, we shall start discussion in this section with FCA (and its operators).

Definition 5 (Derivation Operators). *For a formal context $\mathcal{C} = (U, Att, R)$, define:*

$$R'(X) = \{A \in Att : \forall x \in X\ ((x, A) \in R)\},$$

$$R'(\mathcal{A}) = \{x \in U : \forall A \in \mathcal{A}\ ((x, A) \in R)\},$$

for all $X \subseteq U$ and $\mathcal{A} \subseteq Att$.

	mathematician (1-6)	rich	sub. quality of life (1-3)
Steven	5	$ 0.1 million	3
Bob	4	$ 0.8 million	3
Agnes	2	$ 1 million	1

Fig. 7. A decision table

Definition 6 (Formal Concept). *A formal concept is a pair $(X, \mathcal{A})$ such that $X = R'(\mathcal{A})$ and $\mathcal{A} = R'(X)$. X is called an extension and $\mathcal{A}$ is called an intention of this concept.*

Let us start with a complete information system $\mathcal{I} = (U, Att, Val, f)$ and a semantics S, that is, a family of scales R_A, for all $A \in Att$. As one can easily observe, for every object x in $C_I = (U, \{(A, v)\}_{A \in Att\ v \in Val_A}, R)$, a pair $(R'(R'(\{x\})), R'(\{x\}))$ is a concept, and $y \in R'(R'(\{x\}))$ provided that $x\ Incl\ y$ in the corresponding multivalued information system $\mathcal{I}_S = (U, Att, Val, f_s)$:

$$R'(R'(\{x\})) = \{y \in U : x\ Incl\ y\}.$$

However, it usually happens that

$$R'(R'(X)) \neq \{y \in U : \exists x\ (x\ Incl\ y\ \ \&\ \ x \in X)\}.$$

But it always holds that

$$X \subseteq \{y \in U : \exists x\ (x\ Incl\ y\ \ \&\ \ x \in X)\} \subseteq R'(R'(X)).$$

However, after conversion of an information system $\mathcal{I} = (U, Att, Val, f)$ to the formal context $C_I = (U, \{(A, v)\}_{A \in Att\ v \in Val_A}, R)$ we lose the contact with original attributes (we shall discuss this issue in detail later in the paper).

Definition 7 (Lower and Upper Approximations). *A pair (U, E), where E is an equivalence relation, is called an approximation space. Define after Z. Pawlak:*

$$Low_E(X) = \{x \in U : [x]_E \subseteq X\},$$

$$Upp_E(X) = \{x \in U : [x]_E \cap X \neq \emptyset\}.$$

$Low_E(X)$ is called the lower approximation of X, whereas $Upp_E(X)$ is called the upper approximation of X.

Coming back to information systems, every set of attributes $\mathcal{A} \subseteq Att$ of an information system $\mathcal{I} = (U, Att, Val, f)$ induces an approximation space $(U, E_{\mathcal{A}})$, where $E_{\mathcal{A}} = \{(x, y) : f(x, A) = f(y, A)\ \text{ for all }\ A \in \mathcal{A}\}$. In order to simplify the notation, we shall write $Low_{\mathcal{A}}(X)$ and $Upp_{\mathcal{A}}(X)$ for $Low_{E_{\mathcal{A}}}(X)$ and $Upp_{E_{\mathcal{A}}}(X)$, respectively. In the case when $\mathcal{A} = Att$, we shall leave E without any subscript.

Every information system $\mathcal{I} = (U, Att, Val, f)$ (together with a family of scales S) induces also a multivalued information system $\mathcal{I}_S = (U, Att, Val, f_s)$ and another approximation space (U, Ind). Due to scaling, Ind and E may be two different relations.

For any $\mathcal{A} \subseteq Att$ of (U, Att, Val, f_s) the corresponding indiscernibility relation will be denoted by $Ind_{\mathcal{A}}$. This notational convention will also be used for other relations.

As usual, we can generalise E to any reflexive relation P (e.g. Sim or $Incl$) and obtain generalised approximation operators. Let $[x]_P = \{y \in U : (x, y) \in P\}$ and define:

$$Low_P(X) = \{x \in U : [x]_P \subseteq X\},$$
$$Upp_P(X) = \{x \in U : [x]_P \cap X \neq \emptyset\}.$$

As one can note, it holds that

$$Upp_{Incl}(x) = R'(R'(\{x\})) = \{y \in U : x \; Incl \; y\},$$

and

$$Upp_{Incl}(X) = \{y \in U : \exists x \; (x \; Incl \; y \;\; \& \;\; x \in X)\}.$$

However, $R'(R'(X))$ is much more complex in the settings of information systems than it might seem at the first sight. It is worth noting that $R'(X)$, for $X \subseteq U$, is a set consisting of pairs (A, v). So, in order to go back to the level of information systems we need a method of retrieving the original attributes from this set, so as it would act as $\mathcal{A} \subseteq Att$. Let $Atex$ (attribute extraction) be defined by $Atex(H) = \{A : (A, v) \in H\}$ for $H \subseteq Att \times Val$. Obviously, this a projection operation on the first coordinate and it makes sense only for a family of *regular* scales. Consider the following example. Let $Val = \{1, 2, 3, 4, 5, 6\}$ be a set of values of some attribute A. Assume that a scaling converts 1 to $\{1, 2\}$, 2 to $\{2, 3\}$, and all other values to $\{3, 4, 5, 6\}$. So after scaling A has three value sets: $\{1, 2\}$, $\{2, 3\}$, $\{3, 4, 5, 6\}$. Let $f(x, A) = \{1, 2\}$ and $f(y, A) = \{2, 3\}$. Now, let us start with $(U, \{A\}, Val, f)$, then go to the corresponding C_I, and compute $R'(\{x, y\})$, which is $\{(A, 2)\}$ – but this item does not make sense in our semantics S: $\{2\}$ is the meaning of neither element of Val. Thus, using set intersection $\cap$ we may produce a new non-empty value set, which is not present in $\mathcal{I}_S = (U, Att, Val, f_s)$. A scale is *regular* if that is not possible. Nominal and ordinal scales are regular.

Only for regular scales we are able to define the concepts of the formal context on the level of attributes of information systems. Let $\mathcal{I}_S = (U, Att, Val, f_s)$ be a multi-valued information system obtained from an information system $I = (U, Att, Val, f)$ by means of regular scales S; then

$$R'(R'(X)) = \{y \in U : \forall A \in Atex(R'(X)) \; \exists x \in X \; (x \; Incl_A \; y)\}.$$

If the scale is not regular, then the following inclusion holds only:

$$R''(X) = \{y \in U : \forall A \in Atex(R'(X)) \; \exists x \in X \; (x \; Incl_A \; y)\} \subseteq R'(R'(X)).$$

Therefore, in such a case we need a new name R'' for this operator.

Let us now consider a decision table $\mathcal{I}_G = (U, Att, Val, G, f)$, that is, an information system $\mathcal{I} = (U, Att, Val, f)$ equipped with a decision (goal) attribute $G \notin Att$ and f being defined on $Att \cup \{G\}$. The semantics S for $\mathcal{I}_G$ needs now to include a scale $R_G = (Val_G, Val_G, R_G)$ for the decision attribute. The main goal is to approximate a given pair $(G, [v]_{R_G})$, where $v \in Val_G$ is a specific distinguished value. More precisely, we want to approximate the set $X = \{x \in U : (v, f(x, G)) \in R_G\}$.

Let us take two scaling methods for the decision attribute *subjective quality of life*:

nominal scale Nom: $R_G = \{(i,j) : i,j \in \{1,2,3\} \ \& \ i = j\}$;
ordinal scale Ord: $R_G = \{(i,j) : i,j \in \{1,2,3\} \ \& \ j \leq i\}$.

The nominal scale Nom_G interprets 1 as *low quality of life*, 2 as *average quality of life*, and 3 as *high quality of life*. As expected, the ordinal scale Ord_G interprets $1, 2$, and 3 as: *at least low quality of life*, *at least average quality of life*, and *at least high quality of life*, respectively. Now, let us take the value 3 as the distinguished value of the decision attribute. Then under Nom_G we are going to approximate the set $\{Bob, Steven\}$, however, under Ord_G the set to be approximated is $\{Agnes, Bob, Steven\}$.

The dominance-based rough set approach (DRS) [7, 8] is actually a kind of rough set theory rendered according to the above ideas and the ordinal scaling method. An information system is equipped with a dominance relation $\leq_D$, that is, we consider the system $\mathcal{I} = (U, Att, Val, f, \leq_D)$ (see Section 2). This system induces a multivalued information system $\mathcal{I}_S = (U, Att, Val, f_s)$, where S consists of ordinal scales (Val_A, Val_A, R_A) for every $A \in Att$. Before we recall the definitions of the approximation operators, we need a few auxiliary concepts:

- $D^+(x) = \{y \in U : x \leq_D y\}$ (a set of objects dominating x, or *better than* x);
- $D^-(x) = \{y \in U : y \leq_D x\}$ (a set of objects dominated by x, or *worse than* x);
- Decision attribute G, $V_G = T$, $Cl_t = \{x \in U : f(x,G) = t\}$,

$$Cl_t^{\leq} = \bigcup_{s \leq_G t} Cl_s \text{ and } Cl_t^{\geq} = \bigcup_{t \leq_G s} Cl_s,$$

where $t, s \in T$. It is additionally assumed that

$$Cl_s \cap Cl_t = \emptyset \text{ for } s \neq t \text{ and } \bigcup_{s \in T} Cl_s = U.$$

Classification patterns to be discovered are functions representing granules $Cl^{\leq}{}_t$ and $Cl^{\geq}{}_t$ by means of granules $D^+(x)$ and $D^-(x)$. It is worth emphasising that due to the preference order, the sets to be approximated are not the particular Cl_t (for some $t \in V_G$), but the upward and downward unions.

As said in the previous section, DRS may be represented in terms of ordinal scaling. In what follows we would like to make this scaling explicit in DRS and use relations Ind and $Incl$ (from multivalued information systems) rather than the dominance relation $\leq_D$. Let us consider a multivalued information system $\mathcal{I}_S = (U, Att, Val, f_s)$ obtained from $\mathcal{I} = (U, Att, Val, f, \leq_D)$ by means of a scaling set S. Please note that due to the ordinal scaling of all attributes of $\mathcal{I}$, it holds that:

$$D^+(x) = \{y \in U : x \ Incl \ y\} = R'(R'(\{x\})),$$

$$D^-(x) = \{y \in U : x \ Incl^{-1} \ y\} = \{y \in U : y \ Incl \ x\},$$

$$Cl_t^{\geq} = \{y \in U : \exists x \ (x \ Incl_{\{G\}} y \ \ \& \ \ x \in Cl_t)\},$$

$$Cl_t^{\leq} = \{y \in U : \exists x \ (x \ Incl^{-1}_{\{G\}} y \ \ \& \ \ x \in Cl_t)\}.$$

The specific interpretation of ordinal scaling in DRS makes a new type of inconsistency in data tables possible:

(a) an object x belongs to $Cl^{\geq}{}_t$ (that is, it belongs to Cl_t or a class better than Cl_t), but it is dominated by some objects $y \notin Cl^{\geq}{}_t$ (it is dominated by some object from a worse class),
(b) an object x belongs to the class $Cl^{\leq}{}_t$ (that is, it belongs Cl_t or a class worse than Cl_t), but it dominates some object $y \notin Cl^{\leq}{}_t$ (it dominates some object from a better class).

These objects are regarded as borderline cases: they might or might not belong to a given class. In consequence, in DRS we consider the following approximations:

$$\underline{Cl_t^{\leq}} = \{x \in U : D^-(x) \subseteq Cl_t^{\leq}\},$$

$$\overline{Cl_t^{\leq}} = \{x \in U : D^+(x) \cap Cl_t^{\leq} \neq \emptyset\},$$

$$\underline{Cl_t^{\geq}} = \{x \in U : D^+(x) \subseteq Cl_t^{\geq}\},$$

$$\overline{Cl_t^{\geq}} = \{x \in U : D^-(x) \cap Cl_t^{\geq} \neq \emptyset\}.$$

As earlier, our aim is to express these approximation operators by means of $Incl$. Thus, let be given an information system $\mathcal{I} = (U, Att, Val, f, \leq_D)$ and its corresponding multivalued information system (U, Att, Val, f_s), obtained by means of the ordinal scaling method Ord. Then

$$\underline{Cl_t^{\leq}} = \bigcup_{D^-(x) \subseteq Cl_t^{\leq}} D^-(x) = \{y \in U : \forall x \in U \ (x \ Incl \ y \Rightarrow x \in Cl_t^{\leq})\},$$

$$\overline{Cl_t^{\leq}} = \bigcup_{x \in Cl_t^{\leq}} D^-(x) = \{y \in U : \exists x \in U \ (x \ Incl^{-1} \ y \ \ \& \ \ x \in Cl_t^{\leq})\},$$

$$\underline{Cl_t^{\geq}} = \bigcup_{D^+(x) \subseteq Cl_t^{\geq}} D^+(x) = \{y \in U : \forall x \in U \ (x \ Incl^{-1} \ y \Rightarrow x \in Cl_t^{\geq})\},$$

$$\overline{Cl_t^{\geq}} = \bigcup_{x \in Cl_t^{\geq}} D^+(x) = \{y \in U : \exists x \in U \ (x \ \ Incl \ y \ \ \& \ \ x \in Cl_t^{\geq})\}.$$

So, we are able to transfer FCA, RST, and DRS, along with explicitly given semantics S (a family of scales R_A for every attribute $A \in Att$) into the framework of multivalued information systems. The connections between the operators discussed above are as follows:

$$\underline{Cl_t^{\leq}} = Low_{Incl^{-1}}(Cl_t^{\leq}),$$

$$\overline{Cl_t^{\leq}} = Upp_{Incl}(Cl_t^{\leq}),$$

$$\underline{Cl_t^{\geq}} = Low_{Incl}(Cl_t^{\geq}),$$

$$\overline{Cl_t^{\geq}} = Upp_{Incl^{-1}}(Cl_t^{\geq}) \subseteq R'(R'(Cl_t^{\geq})).$$

Two important comments are needed. As long as we regard all above operators as *approximation* operators, RST and DRS based results are better than that coming from

FCA. However, when we change the context, then the FCA operator may be more preferable. Secondly, in DRS, $x\ Incl\ y$ is read as y *is better than* x. However, there are other readings possible, e.g., *we have more pieces of information about* y *than about* x.

Let us consider an example which brings new meanings for relations and operators we have discussed so far. Let **w** be a serious disease which we have an antibiotic working against. Let us assume that we have a test checking whether someone is ill, but the antibiotic should be given to a patient before the disease develops. So, our aim is to select people who will be given the medicine. One very expensive solution is to give the medicine to all people (that is, all elements of the universe U). On the other extreme, we could give the medicine only to people who test positive for **w**. Of course, both solutions are not good and we need to find another method of selection. We are going to employ the John Stuart Mill inductive reasoning [2] here, which was designed to solve problems of this type, but under complete knowledge. We shall focus our attention on the very basic cannon, namely the direct method of agreement (Fig. 8). We are not going to use the pure form of this cannon, but rather its rough set based rendering.

A B C D	**occur together with**	**w, v**
A E F G	**occur together with**	**w, z**
	Therefore A is the cause of	**w**

Fig. 8. Direct method of agreement: *If two or more instances of the phenomenon under investigation have only one circumstance in common, the circumstance in which alone all the instances agree, is the cause (or effect) of the given phenomenon* [2].

Let be given an information system $\mathcal{I} = (U, Att, Val, f)$ representing our (medical) knowledge about people. At first, we employ a semantics S consisting only of ordinal scales such that for each attribute A of $\mathcal{I}_S = (U, Att, Val, f_S)$, the relation $x\ Incl_A\ y$ means that y *is in a worse medical condition than* x with respect to A and **w**. In our settings, Mill's inductive reasoning is modelled in the following way. The concept **w** is actually a set $\{x \in U : f(x, \mathbf{w}) = \top)$, where $Val_{\mathbf{w}} = \{\top, \bot\}$ (truth and false, respectively). The scale $R_{\mathbf{w}}$ is given by $\{(\top, \top), (\top, \bot), (\bot, \bot)\}$, so if $f(x, \mathbf{w}) = \top$, then $f_S(x, \mathbf{w}) = \{\top, \bot\}$, and if $f(x, \mathbf{w}) = \bot$, then $f_S(x, \mathbf{w}) = \{\bot\}$. Thus, as required, if $x\ Incl_{\mathbf{w}}\ y$, then y is in the same or worse medical condition than x, so if x is ill, then y must be ill as well. Let $Cl_\top$ be a set of positive examples of **w** (direct method of agreement). The term "in common" (Fig. 8) is beyond the expressive power of pure RST and DRS. However, in our case we can retrieve common attributes by means of $Atex(R'(Cl_\top))$. Now we can compute possible solutions to our problem:

$$Cl_\top \subseteq Upp_{Ind}(Cl_\top) \subseteq Upp^{-1}_{Incl}(Cl_\top) = \overline{Cl^{\geq}_\top} \subseteq R'(R'(Cl^{\geq}_\top).$$

As said $Cl_\top$ is an extreme solution, another one may be given by $Upp_{Ind}(Cl_\top)$ (that is, we give the antibiotic to all people with exactly the same medical description in terms of conditional attributes as some patients having positive test for **w**). It seems reasonable, however the next solution $Upp^{-1}_{Incl}(Cl_\top) = \overline{Cl^{\geq}_\top}$ is much better: we give the medicine to all people with the same or worse medical condition than some patients who have

positively tested for **w**. Better still, we may apply the direct method of agreement and give medicine to all people in $R'(R'(Cl_{\top}^{\geq})$ having medically worse results only for attributes which seem to be relevant to **w**. It is worth emphasising that regarded as an approximation of $Cl_{\top}$, the set $R'(R'(Cl_{\top}^{\geq})$ is the worst candidate, but in this very settings it is the best solution for the problem at issue. Consider a non-regular scale now and assume that all people who positively tested on **w** have problems with blood pressure A. So scaling of A shows how unstable is the pressure. Some patients may have value $\{normal, high\}$, some $\{low, high\}$, and some $\{low, normal, high\}$, but none of them has $\{normal\}$. This time $R'(R'(Cl_{\top}^{\geq})$ is a very bad solution, because it may include people with a normal blood pressure. However, we can still use the direct method of agreement in a modified version, and take $R''(Cl_{\top}^{\geq})$ as the solution.

4 Conclusions

In the paper we have investigated the implicit semantics used in some leading theories of data analysis: rough set theory (RST) [4–6], dominance based rough set theory (DRS) [7, 8], and formal contexts from formal concept analysis (FCA) [1, 9]. We have presented all theories within the unifying framework of multivalued information systems [3, 4], enriched with the scaling methods from FCA. We have also discussed the relations between the operators coming from these theories, and presented their different interpretations in the context of John Stuart Mill inductive reasoning [2].

References

1. Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations, Springer Verlag (1999)
2. Mill, J.S.: A System of Logic. Vol. 1. London (1843)
3. Orłowska, E., Pawlak, Z.: Representation of nondeterministic information. Theoretical Computer Science 29, pp. 27–39 (1984)
4. Pawlak, Z.: Systemy informacyjne. Podstawy teoretyczne. Warszawa, WNT (1983)
5. Pawlak, Z.: Rough Sets: Theoretical Aspects of Reasoning about Data. Kluwer Academic Publisher (1991)
6. Pawlak, Z.: Wiedza z perspektywy zbiorów przybliżonych. Institute of Computer Science Report 23, (1992)
7. Słowinski, R., Greco, S., Matarazzo, B.: Dominance-based rough set approach to reasoning about ordinal data. Lecture Notes in Artificial Intelligence 4585, 5–11, (2007)
8. Słowiński, R., Greco, S., Matarazzo, B.: Rough sets in decision making. In: R.A.Meyer y (Ed.), Encyclopedia of Complexity and Systems Science, Springer, NY, 7753–7786, (2009)
9. Wille, R.: Concept lattices and conceptual knowledge systems. Computers & Mathematics with Applications 23, 493–515, (1992)

Superposition Principle in Composable Hybrid Automata

Jafar Akhundov, Peter Tröger, and Matthias Werner

Operating Systems Group, TU Chemnitz, Germany
jafar.akhundov | peter.troeger | matthias.werner@cs.tu-chemnitz.de

Abstract. In the existing abundance of different hybrid automata formalisms concurrent composition is seldom considered or requires additional semantics which is not always defined. This work considers three common reasons of problems with hybrid automata composition: contradicting resets in the discrete transitions, global time reference with contradicting initial conditions and redundant non-determinism for firing time. An overview is provided of the existing formalisms and the attempts to solve these particular problems. A reduced hybrid automata formalism, called linear time-invariant hybrid automata, is introduced. It avoids all those problems and yet provides a powerful modeling tool with practical applications. Also, a short discussion is provided for the problem of Zeno behavior and what conditions are demanded for a model to fulfill so that Zeno behavior would not arise during composition.

1 Introduction

Hybrid systems modeling has various applications in model-driven design and verification of embedded and reactive systems. It has been a topic of intensive research in the past 20 years [MMP91]. Their hallmark is the combination of discrete and continuous behavior. Most hybrid systems include computational components which operate in discrete steps and physical components with continuous behavior over time. Typical examples are aerospace systems, robotic systems, or process control systems. Since most of these systems are too complex to design and build as a whole, they are decomposed into subsystems and components with reduced complexity and simpler behavior. This process can, of course, be recursively repeated until complexity is manageable. In order for this process to be supported on the modeling level, it is necessary that the applied formalism allows for (de)composition with respect to the verifiable properties such as reachability or liveness. The process of decomposition has been historically used in the control systems engineering applications [Nis11]. An important property which is often used to simplify design and analysis is the superposition[1] principle which is mathematically defined as:

$$h(ax_1 + bx_2) = ah(x_1) + bh(x_2)$$

[1] also called linearity

Since their introduction, hybrid automata formalisms have been emerging with restricted properties to simplify analysis and sometimes composition [AD94] [Hen96] [LSV03] [Ábr12]. Examples of subsets of hybrid automata are timed automata [AD94], linear hybrid automata [Hen96], rectangular hybrid automata [HKPV98], hybrid I/O automata [LSV03], etc. Several definitions of the general hybrid automata exist as well, each with slight deviations in the underlying semantics.

A handful of frameworks leave some of the semantics unspecified which makes it difficult for the designer to apply them - separately or compositionally [Hen96] [Ras05] [Ábr12] [LLL09]. An example is a general structural definition of HA where each location has an invariant and several outgoing transitions with respective guards [Hen96] [Ras05] [LLL09] [Ábr12]. The problem arises when the invariant is violated thus forcing the automaton to switch its location to another one but no guarding condition of the outgoing transitions is enabled. It remains unspecified what happens to the model in such a situation. Another example is the passage of time in several parallel composed automata with synchronising labeled transitions [Ábr12]. Since the event (action) semantics is not always specified fully and consistently, i.e. are events buffered or ignored, or what is the global time reference for two composed automata, it is unclear whether one synchronising edge should wait for another one with the same label in the second automaton. There are formalisms which allow for such "waiting" which enables to model physical systems where objects are floating in space waiting for some other event to occur.

Thorough comparison of the existing HA formalisms has lead to the conclusion that three common reasons of problems for hybrid automata composition exist:

1. contradicting resets in the discrete transitions,
2. global time reference with contradicting initial conditions and
3. redundant non-determinism for firing time.

For the practical application of composable control systems a formalism is needed which has none of the aformentioned problems and fulfills the property of superposition of continuous functions.

The contribution of this work lies in the introduction of a new formalism for modeling hybrid systems with a fully specified timing, firing, event and composition semantics and fulfilling the property of superposition motivated by the applicability from the control systems engineering. Our approach is driven by the motivating example of a dedicated domain specific language for the verification of a space mission at the early design phases where superposition is a critical issue [ASGW16], [ATW15], [STF+13].

The article starts with an overview of the existing hybrid automata formalisms which experience and/or partially solve the composition problems. The general definition of the utilized hybrid automata variation is given in Section 3. The text continues with a detailed discussion of composition semantics and the arising problems. Section 4 shows how the LTI-HA solve these problems. The

paper is concluded by a discussion of further work and possible applications of the formalism.

2 State of the Art

In the existing abundance of different hybrid automata formalisms concurrent composition is seldom considered in full depth or requires additional semantics which is not always defined [Ras05] [LLL09] [Ábr12]. In the general setting, HA experience all of the three mentioned problems [Hen96] [Ras05] [Ábr12] [LLL09] [Alu15]. In [Ábr12], an overview is provided of the existing hybrid automata formalisms with rising complexity, starting from labeled transition systems, timed automata and ending with the general hybrid and rectangular automata. That work provides a conceptualized structural view on the hybrid systems. All three types of problems occur in the generalized HA and at least partially in the other formalisms. Furthemore, many formalisms suffer from incompleteness of semantics definition [Hen96] [Ras05] [LLL09] [Ábr12].

Hybrid I/O automata (HIOA) were introduced by Lynch et al. first in 1996 [LSVW96] but have been modified several times since [LSV03]. The definition of hybrid I/O automata is unique in the sence that it eliminates a handful of problems by defining the hybrid automata by the notion of hybrid traces. Hybrid I/O automata have been demonstrated to be both composable and receptive[2]. However, HIOA are too restrictive for some of the control applications where explicit notion of superposition is important. For example, HIOA are required to have disjunct output trajectories [LSV03, p.131,p.141] which excludes the possibility of superposition.

Superposition of the flow functions of hybrid automata has been exploited in the linear hybrid automata, however, the introduced formalisms still have at least one of the semantic problems listed in the problem statement [Hen96] [Pap98].

3 Linear Time-Invariant Hybrid Automata (LTI-HA)

In the following section, a new formalism is introduced to solve the problems 1-3.

3.1 Definition

Before the linear time-invariant hybrid automata are defined, several supporting definitions are provided.

Definition 1 (Valuation of a variable). *A valuation $V(x)$ of a variable x is the assignment to x of a value from its domain $\mathcal{D}$: $V(x) : x \mapsto \mathcal{D}(x)$.*

This definition can be extended to a set of variables:

[2] Not experiencing Zeno behavior, even under the composition.

Definition 2 (Valuation of a set of variables). *A valuation $V(X)$ of a variable set X is the union of all valuations for all $x \in X$ of a value from the corresponding domains $\mathcal{D}(x)$: $V(X) : X \mapsto \mathcal{D}(x)$.*

The set of all possible valuations is defined as $\mathcal{V}(\mathcal{X}) = D(x_1) \times D(x_2) \times ... \times D(x_n)$.

Definition 3 (State of a hybrid system). *A state of hybrid system is a pair $(L, V)(t)$ consisting of two time-dependent components: the discrete state (L) and the continuous state (V).*

Definition 4 (LTI-HA). *A linear time-invariant hybrid automaton $\mathcal{H}$ is a tuple $(\mathcal{L}, \mathcal{T}, \mathcal{X}, \mathcal{S}_I, \mathcal{S}_O, \mathcal{E}, \mathcal{A}, G, \mathcal{F}, \mathcal{I})$, where:*

- *$\mathcal{L} = (L_1, \ldots, L_n)$ is a set of discrete **locations** also called modes;*
- *$\mathcal{T} \subseteq \mathcal{L} \times \mathcal{L}$ is a (not necessarily complete) multiset **transition relation**;*
- *$\mathcal{X}$ is a set of continuous **state variables**. To each $x \in \mathcal{X}$, a value from $\mathcal{D}(x) \subseteq \mathbb{R}^m \cup \{dc\}$ can be assigned where $m \geq 1$ but is finite and dc is a special term for unspecified ("don't care") value;*
- *$\mathcal{S}_I$ and $\mathcal{S}_O$ are two disjunct sets of input and output events, respectively, which define the automaton's event signature;*
- *$\mathcal{E} : \mathcal{T} \mapsto \mathfrak{P}(\mathfrak{P}(\mathcal{S}_I))$ and $\mathcal{A} : \mathcal{T} \mapsto \mathfrak{P}(\mathcal{S}_O)$ are assignments of the interface events $\mathcal{S}_I, \mathcal{S}_O$ to the transitions of the automaton;*
- *$G : \mathcal{T} \times \mathcal{V}(\mathcal{X}) \times \mathfrak{P}(\mathcal{S}_O) \times \mathfrak{P}(\mathfrak{P}(\mathcal{S}_I)) \mapsto \{\text{true, false}\}$ is a **guard function**. For all transitions τ, $G(\tau, \mathcal{V}(\mathcal{X}), \mathcal{A}(\tau), \mathcal{E}(\tau)) = g_\tau(\mathcal{V}(\mathcal{X}), \mathcal{A}(\tau), \mathcal{E}(\tau))$ is called the **guard (function)** of τ;*
- *For any location L and for all variables from $\mathcal{X}$ there exists a linear differential equation $\mathcal{F}(f_L(x,t)) = g(x,t), x \in \mathcal{X}, t \in \mathbb{R}^{\geq 0}$ with $g(x,t) : \mathcal{X} \times \mathbb{R}^{\geq 0} \mapsto \mathbb{R}^m$ describing the change of the corresponding variable, where $f_L(x,t) : \mathcal{X} \times \mathbb{R}^{\geq 0} \mapsto V(\mathcal{X})$ is called a **flow function**, $\mathcal{F}(f_L)$ is a linear operator $P_0(t) f_L^{(k)} + P_1(t) f_L^{(k-1)} + ... + P_k(t) f_L$ with $f_L^{(l)} = \frac{d^l f_L}{dt^l}$ and $P_i(t) : \mathbb{R} \mapsto \mathbb{R}$ being any functions. The set of all flow functions in a given location describe how the valuations of continuous state variables change over time in that location;*
- *$\mathcal{I}$ is the **initial state** of the system $(L_\mathcal{I}, V_\mathcal{I})$, where $L_\mathcal{I} \in \mathcal{L}$ is the initial active mode and $V_\mathcal{I} = V_\mathcal{I}(\mathcal{X})$ is the initial valuation of all the variables in $\mathcal{X}$.*

In contrast, the general definition of hybrid automata usually also includes additional constructs such as location invariants, variable resets along the transitions, flow functions are uncostrained and events that are labels with a simple synchronisation semantics.

Definition 5 (Time Semantics). *Evaluation of flow functions is based only on the duration of time interval spent in the corresponding location. In each active location, time elapses at the same rate. Transitions are timeless.*

Global time reference can be implemented by taking any fixed reference time value which is progressing along with the automata execution. At any time,

exactly one location is *active*, beginning with the $L_{\mathcal{I}}$. Automaton's state changes either with time with respect to the flow functions of the corresponding locations or the discrete transitions, starting in the initial state $\mathcal{I}$.

Definition 6 (Transition Semantics). *As long as an automaton has an active location L, the valuation of continuous variables $V(\mathcal{X})$ changes according the location's flow function f_L. If no explicit flow function is given for some variables from $\mathcal{X}$ their rate of change is assumed to be 0 at the given location. If at some time point a guard g_τ of an outgoing transition (L_c, L_d) evaluates to true, L_d becomes the new active location without delay and all events $e \in \mathcal{A}(\tau)$ occur. If more than one guard of an outgoing transition evaluates to true, one of the transitions is chosen non-deterministically.*

Definition 7 (Event Propagation Semantics). *The output events have a one-to-all semantics, that is, every output event is broadcasted. The input events have a one-to-one semantics and are therefore only generated by a single other automaton. Each input event has to be defined and specified.*

Definition 8 (Event Structure Semantics). *The input events for a transition τ form a set $\mathcal{E}(\tau) \in \mathfrak{P}^2(\mathcal{S}_I)$ where $\mathfrak{P}^2(\mathcal{S}_I)$ is a power set of a power set over the set of input events, that is complex events can be formed by coupling the (elementary) input events in the following way: for the transition τ to become enabled, at least one of the (complex) events $S \in \mathfrak{P}^2(\mathcal{S}_I)$ in the set $\mathcal{E}(\tau)$ has to occur. Occurrence of such an event implies that **all** participating events $s \in S$ have occurred (simultaneously).*

Definition 9 (Event Timing Semantics). *Events don't have duration and occurrences are not buffered.*

There are two possibilities to describe *interval events*: by two events, one for the start e_s and one for the completion e_f, respectively, or by setting global variables values. Problem with modeling by just events arises when they are not caught thus leading to either offsets in the interval perceptions or overly complex conditions for well-definedness and composability. Overlapping intervals are easily modeled by global variables with constant values.

3.2 Semantics of the LTI-HA

Definition 10 (Timed Transition System (TTS)). *A timed transition system (TTS) is a tuple $(\Sigma, \Sigma_0, \mathcal{S}, \rightarrow)$ where Σ is a (possibly infinite) state space with $\Sigma_0 \subseteq \Sigma$ being the initial state and $\mathcal{S}$ is a (finite) set of labels. Transition relation is defined as $\rightarrow \subseteq \Sigma \times \mathcal{S} \cup \mathbb{R}^{\geq 0} \times \Sigma$.*

Definition 11 (Trace Semantics of a Hybrid Automaton). *Trace semantics of a hybrid automaton $\mathcal{H} = (\mathcal{L}, \mathcal{T}, \mathcal{X}, \mathcal{S}_I, \mathcal{S}_O, \mathcal{E}, \mathcal{A}, G, \mathcal{F}, \mathcal{I})$ is defined as a transition system where:*

- *the (possibly infinite) state space is the set of pairs $(l, V_l(\mathcal{X}))$, where $V_l(\mathcal{X})$ is in the range of possible valuations in l, defined by f_l*

– *initial state is* $\mathcal{I}$
– *and the transitions "*$\rightarrow$*" are either:*
  - ***discrete**:* $\forall T \in \mathcal{T} \quad \exists (l_i, V_i(\mathcal{X})) \rightarrow_\sigma (l_j, V_j(\mathcal{X})), \sigma \in \mathfrak{P}(\mathcal{S}), l_i, l_j \in \mathcal{L}$
  - *or* ***continuous**:* $\exists \delta \in \mathbb{R}^{\geq 0}, \delta = t_f \quad \exists (l_i, V_i(\mathcal{X})) \rightarrow_\delta (l_i, V_j(\mathcal{X})) \wedge f_{l_i}$ *is differentiable on* $[0, \delta]$ *and the following conditions hold:*
    1. $f_{l_i}(0) = V_i(\mathcal{X})$,
    2. $f_{l_i}(\delta) = V_j(\mathcal{X})$,*and*
    3. $f_{l_i}(0)$ *is closed under subintervals.*

Thus, a trace of a hybrid automaton is a finite sequence alternating between continuous evolutions with finite durations and discrete transitions:

$$\pi = s_0 \epsilon_0 s_1 \epsilon_1, ..., \epsilon_{n-1} s_n,$$

where s_i are the states in TTS, ϵ_i are the transitions (discrete or continuous) between them and $s_0 = \mathcal{I}$. Duration of a trace $d(\pi)$ is defined as the sum of all durations along that trace. Since a HA can be non-deterministic, many different traces are possible. Generating a control sequence of external events $\mathcal{C} \subset (S, t)$, where $S \subset \mathcal{S}_I$ and t is a time point with respect to the global time reference, is not part of the model but a task for an external solver.

3.3 Composition of LTI Hybrid Automata

Definition 12 (Composability). *Two LTI hybrid automata* $\mathcal{H}^1, \mathcal{H}^2$ *are called composable,* $\mathcal{H}^1 \mho \mathcal{H}^2$*, iff* $\forall x \in \mathcal{X}^1 \cap \mathcal{X}^2 : V^1_{\mathcal{I}}(x) = V^2_{\mathcal{I}}(x)$*, where* $dc = \kappa$ *is always true for all values of* κ*.*

Definition 13 (Composition). *Given two composable hybrid automata* $\mathcal{H}^1$ *and* $\mathcal{H}^2$*, the composition* $\mathcal{H}^1 \circ \mathcal{H}^2$ *provides a new hybrid automaton* $\mathcal{H}^c = (\mathcal{L}^c, \mathcal{T}^c, \mathcal{X}^c, \mathcal{S}^c_I, \mathcal{S}^c_O, \mathcal{E}^c, \mathcal{A}^c, G^c, F^c, \mathcal{I}^c)$ *where*

1. $\mathcal{L}^c = \mathcal{L}^1 \times \mathcal{L}^2 = \{(L^1_1, L^2_1), \ldots, (L^1_1, L^2_{n_2}), (L^1_2, L^2_1), \ldots, (L^1_{n_1}, L^2_{n_2})\}$
   $= \{L^c_{11}, L^c_{12}, \ldots, L^c_{1n_1}, \ldots, L^c_{n_1 n_2}\}$;
2. $\mathcal{T}^c = \{t = (L^c_{ij}, L^c_{kl}) | t \in \mathcal{L}^c \times \mathcal{L}^c, (L^1_i, L^1_k) \in \mathcal{T}^1 \vee (L^2_j, L^2_l) \in \mathcal{T}^2\}$
3. $\mathcal{X}^c = \mathcal{X}^1 \cup \mathcal{X}^2$,
4. $\mathcal{S}^c_I = (\mathcal{S}^1_I \cup \mathcal{S}^2_I) \backslash \mathcal{S}^1_O \backslash \mathcal{S}^2_O$
5. $\mathcal{S}^c_O = \mathcal{S}^1_O \cup \mathcal{S}^2_O$
6. $\forall \tau \in \mathcal{T}$: $\mathcal{A}^c(\tau) = \{e \mid \tau = (L^c_{ij}, L^c_{kl}) \wedge$
   $\big((e \in \mathcal{A}^1((L^1_i, L^1_k)) \wedge j = l) \vee (e \in \mathcal{A}^2((L^2_j, L^2_l)) \wedge i = k)\big)\}$
7. $\forall \tau \in \mathcal{T}$: $\mathcal{E}^c(\tau) = \{S \mid \forall E \in \mathcal{E}^1(\tau) \cup \mathcal{E}^2(\tau) : S = E \backslash \mathcal{S}^1_O \backslash \mathcal{S}^2_O\}$
8. $G^c = \{g^c_{(L^c_{ij}, L^c_{kl})} = g^1_{(L^1_i, L^1_k)} | (L^c_{ij}, L^c_{kl}) \in \mathcal{T} \wedge j = l\} \cap$
   $\{g^c_{(L^c_{ij}, L^c_{kl})} = g^2_{(L^2_j, L^2_l)} | (L^c_{ij}, L^c_{kl}) \in \mathcal{T} \wedge i = k\}$
9. $\forall L_{ij} \in \mathcal{L}^c$: $f_{L_{ij}} = f_{L_i} + f_{L_j}$
10. $\mathcal{I}^c = (L^c_{i_1 i_2}, V^1_{\mathcal{I}} \cup V^2_{\mathcal{I}})$

Properties from semantics definitions 5-9 remain preserved.

Properties 1 and 2 define the new location set which is now a cartesian product of two initial location sets, and the transition in a new location set exists if there was at least one transition in the corresponding locations of initial automata $\mathcal{H}^1$ and $\mathcal{H}^2$. The variable set is defined as a union set. All variables with the same names are to be considered global and can be adjusted in a composed manner (property 9). Properties 4-7 describe how the input and output events of the automata are composed. Since the input events have the one-to-one semantics, if there one of the two composed automata $\mathcal{H}^1$ and $\mathcal{H}^2$ having an output event which is the input event for the second automata this event can be cancelled out in the input set of the resulting composed automaton. This, however, does not apply for the output events because of their one-to-all semantics. Properties 6-8 define how the input and output events and the guards are assigned to their corresponding transitions in the initial automata and. Mildly speaking, G builds a cut set of enabling valuations in two composed transitions and a union of the input and output events, respectively. Property 9 follows trivially from the linearity property of the superpositioned differential equations. Since $\mathcal{H}^1$ and $\mathcal{H}^2$ are composable it is safe to apply property 10.

4 Semantics of Composition of Hybrid Automata

4.1 General Hybrid Automata

In the general setting, hybrid automata are defined as follows [Hen96, p.2] [Ras05, p.4] [LLL09] [Ábr12]:

Definition 14. *A hybrid automaton consists of:*

- *A set of continuous variables X;*
- *A finite directed multigraph (V, E) representing the discrete modes and the transitions between them;*
- *Initial conditions describe how the continuous variables are reset after a timeless discrete transition has been taken;*
- *Invariants are the predicates assigned to the discrete locations and must hold in the respective location when it is active;*
- *Flow conditions describe how the variables in X change continuously with time;*
- *Guard conditions which are the predicates over the values of variables of X and are the enabling conditions for a transition to be taken;*
- *Events which are assigned to the transitions of the automaton. Transitions with with the same labels in different automatas must synchronise.*

There are some extensions and small differences between the definitions which both solve some problems and introduce others. For instance, in [Ábr12] it is proposed that discrete transitions in concurrent automata are interleaved, and synchronisation is enhanced with special τ-transitions for the automaton which is waiting on the synchronising edge. It is immediately clear that this extension solves the problem of race conditions of the resets with non-determinism for the

case when, e.g. two transitions with different labels fire simultaneously with the following resets: $x = y + 1$ and $y = x + 1$. However, introducing τ-transitions allows for modeling of a satellite in the orbit which, while waiting for some maneuver comand from an operations control center, suddenly freezes dead in its orbit since τ-transitions are also timeless. Furthermore, it does not solve a problem of possible Zeno behavior when time is prevented from passing. A trivial example is given in Fig. 1. If the automata start in states A and B with $x = 0, y = 0$, then after 10 time units, the system will converge and generate an infinite amount of discrete events. The execution trace would be $(A \to B) \Rightarrow (C \to D) \Rightarrow (B \to A) \Rightarrow (D \to C) \Rightarrow ...$

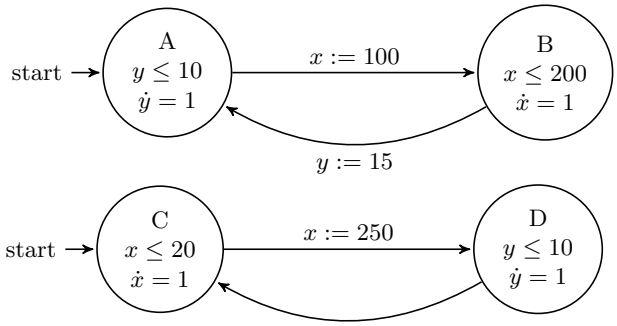


Fig. 1: Example of time convergence because of the discrete resets

In the general setting, usually no assumption is made[3] about the type of differential equations governing the continuous change of the state variables. If those are not time invariant, it leaves the question open as to how the flow functions are overlapped during parallel composition. Another important drawback is the lack of specification of the semantics in the case when a state invariant is violated prior to enabling of any outgoing transition [HKPV98] [Ras05] [LLL09] [Ábr12].

4.2 Other Formalisms

Several other formalisms based on the general hybrid automata have been presented [Hen96] [Ras05] [LLL09] [Ábr12].

An overview of the three possible problems mentioned earlier occurring in the HA formalisms is provided in table (1). Timed automata, being the simplest form of the HA, avoid most of the problems of composition, as well as lack expressivity to describe complex hybrid phenomena [LLL09] [Cas05]. Linear hybrid automata have support for superposition but still have the invariants and resets, and, hence, the implied problems that could arise. Rectangular automata, just as the

[3] An exception would be [LLL09]

general automata, experience all of the three problems and also increase the complexity by introducing randomness and uncertainties [Hen96] [HKPV98].

Hybrid I/O automata solve all of the mentioned problems of composition but have explicitly eliminated the possibility for superposition of trajectories for the continuous variables [LSV03, p.131,p.141].

HA Formalism	Problems with composition			Superposition Support
	R	IC	FTND	
Timed Automata [AD94]	-	-	✓	-
Linear Hybrid Automata	✓	-	✓	✓
I/O Hybrid Automata	-	-	-	-
General Hybrid Automata	✓	✓	✓	-
Rectangular Hybrid Automata	✓	✓	✓	-

Table 1. Compositional problems with the HA formalisms - R: resets; IC: initial conditions; FTND: fire time non-determinism.

4.3 Composition in LTI Hybrid Automata

Since composition is the cornerstone of the new formalism that is introduced in this paper, the absence of each of the three undesirable prooerties $\mathcal{P}_i$ can be only guaranteed if and only if LTI hybrid automata as a formalism fulfill the following two conditions:

1. the property $\mathcal{P}_i$ cannot exist in a single automaton;
2. the $\mathcal{P}_i$-freeness is preserved and $\mathcal{P}_i$ not induced by composition,

where $\mathcal{P}_1 \equiv$ R, $\mathcal{P}_2 \equiv$ IC and $\mathcal{P}_3 \equiv$ FTND.

Theorem 1. *No contradicting resets are possible in the LTI hybrid automata.*

Proof

1. Discrete transition resets are not a part of the LTI definition 4. Furthermore, since there is no composition taking place, no contradictions are possible. The proof follows trivially.
2. The proof of preservation also follows trivially from the definition 13 of composition, since no rule introduces discrete resets. The only discrete jumps of the variable values are possible due to Dirac impulses which do not violate the superposition and time-invariance property.□

Theorem 2. *No global time reference exists with the contradicting initial conditions in the LTI hybrid automata.*

Proof

1. As per definition 5, time flow is identical for all discrete states, hence time invariance. Definition 4 implies that the flow functions are also linear. Since every variable can only be assigned with value once in the initial state, no contradictions are possible.
2. Proving that this statement is preserved and not induced by composition is equivalent to proving that if two automata are composed with each other the induced automaton is also composable with some other third automaton, since non-contradicting initial conditions are the necessary and sufficient condition for composability.
   We assume that automata $\mathcal{H}^1, \mathcal{H}^2$ and $\mathcal{H}^3$ are pairwise composable. Without loss of generality, rules 3 and 10 of the composition definition 13 are applied to automatas $\mathcal{H}^1, \mathcal{H}^2$:

$$\text{Theorem 2} \frac{\mathcal{H}^1 \ \mho \ \mathcal{H}^2, \mathcal{H}^2 \ \mho \ \mathcal{H}^3, \mathcal{H}^1 \ \mho \ \mathcal{H}^3}{(\mathcal{H}^1 \circ \mathcal{H}^2) \ \mho \ \mathcal{H}^3}$$

   After applying rule 13.11, $V_{\mathcal{I}}^{12} = V_{\mathcal{I}}^1 \cup V_{\mathcal{I}}^2$. Thus, set $V_{\mathcal{I}}^{12}$ can be divided to three subsets, elements only from $V_{\mathcal{I}}^1$, elements only from $V_{\mathcal{I}}^2$ and elements from $V_{\mathcal{I}}^1 \cap V_{\mathcal{I}}^2$. Let us assume that $V_{\mathcal{I}}^1 \cap V_{\mathcal{I}}^3 \neq \emptyset$ and $V_{\mathcal{I}}^2 \cap V_{\mathcal{I}}^3 \neq \emptyset$. Then, from the initial assumption and the definition 12,

$$\forall x \in \mathcal{X}^1 \cap \mathcal{X}^3 : V^1(x) = V^3(x) \ \wedge$$
$$\forall x \in \mathcal{X}^2 \cap \mathcal{X}^3 : V^2(x) = V^3(x)$$

   Hence,

$$\forall x \in \mathcal{X}^1 \cap \mathcal{X}^2 \cap \mathcal{X}^3 : V^1(x) = V^2(x) = V^3(x)$$

   Linearity and time-invariance are preserved with respect to superposition [Nis11]. Since all of the flow functions in the LTI-HA are linear and time-invariant, the same applies for the composed automata after applying rules 1 and 9 of definition 13. □

Theorem 3. *Execution of LTI-HA never stalls due to the contradicting invariants and guard conditions.*

Proof

1. Invariants are absent in the definition of the LTI-HA 4. Hence, it is not possible for the behavior of the model to be unspecified due to a violated invariant with no guards enabled.
2. The rule for guards in the composition definition, 13.8, combines several guards of the initial automata. If the guards are contradicting each other, transition is omitted altogether. No invariants are created and the firing enforcement (definition 6) is preserved. □

5 Discussion

Although some of the common problems mentioned in the introduction of this work have been eliminated in the LTI-HA, others may remain which cannot be completely excluded for hybrid systems or are implied by the semantics of the modeled system itself. It is therefore useful to determine a set of properties, which, combined, will introduce a notion of well-definedness of the model. A well-defined model would guarantee correct behavior with respect to the property of interest, that is, model would behave without experiencing unexpected or unwanted behavior. One of such properties is divergence of time. If a model is Zeno-free, time never converges, i.e. it is impossible to find a subsequence of the model execution trace which includes an infinite amount of events in a finite time. This notion has been extended by Lynch et al. in [LSV03] with the case where a trajectory of a continuous variable is never asymptotic.

Introduction of such a notion into the LTI-HA formalism would allow for automatic checking if a model can or cannot end in a Zeno execution thus making verification impossible. One possible condition for guaranteeing Zeno-freeness would be the absence of closed transition loops of length ≥ 1 consisting of only transient modes, i.e. modes for which at least one outgoing transition is enabled when the mode is entered. If there exists such a cycle, then the model will be executed, following the transition semantics 6, endlessly without the progress of time.

6 Conclusion

We presented a formalism that allows the semantic description of linear control systems based on hybrid automata. Several problems of composition of other formalisms have been demonstrated with a comparative analysis to our method.

As next steps, we intend to derive the necessary and sufficient conditions for non-Zenoness of the LTI-HA models and demonstrate that this property is preserved by composition. Furthermore, the problem of analysis of liveness and reachability and the implementation of a corresponding tool support will follow.

References

[Ábr12] Erika Ábrahám. Modeling and analysis of hybrid systems: Lecture notes, April 2012.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Alu15] R. Alur. *Principles of Cyber-physical Systems.* 2015.

[ASGW16] Jafar Akhundov, Volker Schaus, Andreas Gerndt, and Matthias Werner. Using timed automata to check space mission feasibility in the early design phases. In *IEEE Aerospace 2016 Proceedings*, Big Sky, Montana, USA, March 2016.

[ATW15] Jafar Akhundov, Peter Tröger, and Matthias Werner. Considering concurrency in early spacecraft design studies. In *CS&P 2015 Proceedings*, pages 22–30, Rzeszow, Poland, 9 2015.

[Cas05] B. Brard F. Cassez. Comparison of the expressiveness of timed automata and time Petri nets. In *In Proc. FORMATS05, vol. 3829 of LNCS*, pages 211–225. Springer, 2005.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–, Washington, DC, USA, 1996. IEEE Computer Society.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94 – 124, 1998.

[LLL09] Jan Lunze and Franoise Lamnabhi-Lagarrigue, editors. *Handbook of hybrid systems control : theory, tools, applications*. Cambridge University Press, Cambridge, UK, New York, 2009.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, August 2003.

[LSVW96] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. *Hybrid I/O automata*, pages 496–510. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, pages 447–484, 1991.

[Nis11] N.S. Nise. *Control Systems Engineering*. Wiley, 2011.

[Pap98] G. Pappas. Hybrid Systems: Computation and Abstraction. 1998.

[Ras05] Jean-François Raskin. *An Introduction to Hybrid Automata*, pages 491–517. Birkhäuser Boston, Boston, MA, 2005.

[STF+13] Volker Schaus, Michael Tiede, Philipp M. Fischer, Daniel Lüdtke, and Andreas Gerndt. A Continuous Verification Process in Concurrent Engineering. In *AIAA Space Conference*, September 2013.

Appendix: A Modelling and Composition Example

To further motivate the use of composable hybrid automata as introduced in this work, a small practical example for satellite functionality is discussed in this Appendix.

In the early conceptual study phase of development, a satellite downlink module which sends gathered information back to Earth can be modelled as having only two distinct states: *Sending*, when a ground station is visible and there is data to send, or *Not Sending*, when either no ground station is available or no data is there to be sent (or both). In the *Sending* state the rate of change of available data and sent data is the same with opposite signs, whereas in the *Not Sending* state both parameters remain constant (Fig. 2).

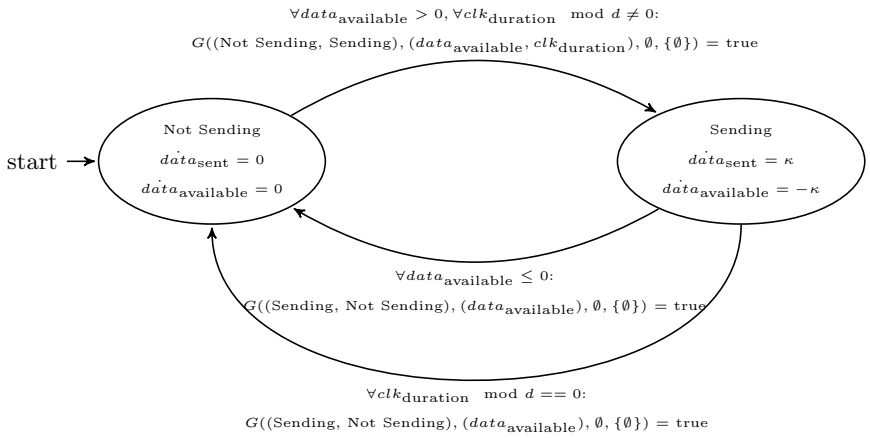


Fig. 2: Downlink module model definition using a composable LTI hybrid automaton.

The automaton representing ground station availability is presented in Fig. 3. Here, as well, the system has only two states, since a ground station is either available or not. For now, we ignore irregularities of this otherwise periodic process, such as orbit perturbations and communication faults - they can be integrated into the model by taking the worst, shortest possible availability period. Ground station visibility is modelled by two running clocks, one for the period and one for the duration. Once the period time is up, ground station becomes visible for the possible duration time which is measured by the second clock, clk_{duration}. When the clock reaches its maximum value, the automaton switches its discrete state back to the "Not Visible" mode. Since there are no resets in the LTI-HA formalism, it is not possible to reset the clocks. Hence, modular arithmetic is applied. When either of the transitions is taken in the ground station automaton, an output event is generated, one for the start of the visibility period, and one for the end, so that other concurrent automatas could

synchronise their transitions with this periodic interval. However, it is also possible to model communication by using the global values of the two clocks of the ground station automaton. This approach is used in the Fig. 2 - it is easy to see that whenever the value of the clock clk_{duration} is not zero modulo the interval duration, the ground station automaton is in its visible state, so the transition from "Not Sending" to "Sending" modes remains enabled. However, when the value is zero modulo interval duration and the downlink module is active, or there is no data to be sent, it should switch back to the "Not Sending mode".

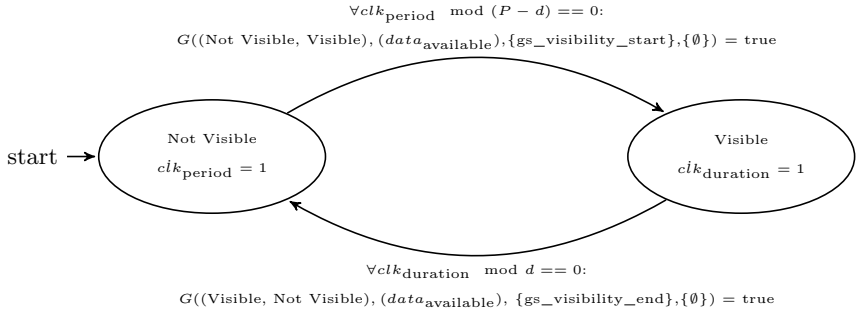


Fig. 3: Periodic ground station visibility model definition using a composable LTI hybrid automaton.

It is assumed that initial states for the initial automata are ("Not Sending", $\{data_{\text{sent}} = 0,\ data_{\text{available}} = C\}$) and ("Not Visible", $\{clk_{\text{period}} = 0, clk_{\text{duration}} = 0\}$). Obviously, both are composable, since the cut set of their initial valuations does not have contradictions. The composed automaton is built by applying composition rules 1-10 and its control graph is depicted in Fig. 4:

1. $\mathcal{L}^c$ = (('Not Sending', 'Not Visible'), ('Not Sending', 'Visible'), ('Sending', 'Not Visible'), ('Sending', 'Visible')) = (nsnv, nsv, snv, sv)
2. $\mathcal{T}^c$ = { ((nsnv, nsv), (nsnv, snv), (nsnv, sv), (snv, nsv), (snv, nsv), (snv, sv), (snv, nsnv), (snv, nsnv), (nsv, sv), (nsv, snv), (nsv, nsnv), (sv, nsv),(sv, nsv), (sv, nsnv), (sv, nsnv), (sv, snv))}
3. $\mathcal{X}^c = \{data_{\text{available}}, data_{\text{sent}}, clk_{\text{period}}, clk_{\text{duration}}\} \cup \{clk_{\text{period}}, clk_{\text{duration}}\} = \{data_{\text{available}}, data_{\text{sent}}, clk_{\text{period}}, clk_{\text{duration}}\}$,
4. $\mathcal{S}_I^c = \emptyset \cup \emptyset = \emptyset$
5. $\mathcal{S}_O^c = \emptyset \cup$ {gs_visibility_start, gs_visibility_end} = {gs_visibility_start, gs_visibility_end}
6. It is clear that for all the cases when only one of the states in a pair changes, transitions remain unchanged from the corresponding initial automaton. The output events of the corresponding edges are:
   - (nsnv, nsv): { gs_visibility_start };
   - (nsnv, snv): ∅;
   - (nsnv, sv): { gs_visibility_start };
   - (snv, nsv): { gs_visibility_start };

- (snv, nsv): { gs_visibility_start };
- (snv, sv): { gs_visibility_start };
- (snv, nsnv): $\emptyset$;
- (snv, nsnv): $\emptyset$;
- (nsv, sv): $\emptyset$;
- (nsv, snv): { gs_visibility_end };
- (nsv, nsnv): { gs_visibility_end };
- (sv, nsv): $\emptyset$;
- (sv, nsv): $\emptyset$;
- (sv, nsnv): { gs_visibility_end };
- (sv, nsnv): { gs_visibility_end };
- (sv, snv): { gs_visibility_end }.

7. Since the set of input events is empty, $\{\emptyset\}$ is assigned as input events to the all of the transitions.
8. The guards of the resulting transitions are:
   - (nsnv, nsv): $g((\forall clk_{\text{period}} \mod (P-d) == 0), \{ \text{gs\_visibility\_start} \}, \{\emptyset\}) = true$;
   - (nsnv, snv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0), \emptyset, \{\emptyset\}) = true$; this transition will never be taken, since this condition cannot be fulfilled before the ground station becomes visible;
   - (nsnv, sv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0, \forall clk_{\text{period}} \mod (P-d) == 0), \{ \text{gs\_visibility\_start} \}, \{\emptyset\}) = true$ this transition is never taken since decision about taking a transition falls *before* time starts ticking in the ground station visibility mode which is required to enable the second condition;
   - (snv, nsv): $g((\forall data_{\text{available}} \leq 0), \{ \text{gs\_visibility\_start} \}, \{\emptyset\}) = true$; this transition will never be taken, since the ground station should have been visible for the sending to be possible before this transition;
   - (snv, nsv): $g((\forall clk_{\text{duration}} \mod d == 0), \{ \text{gs\_visibility\_start} \}, \{\emptyset\}) = true$; same as the last point;
   - (snv, sv): $g((\forall clk_{\text{period}} \mod (P-d) == 0), \{ \text{gs\_visibility\_start} \}, \{\emptyset\}) = true$; same as last point;
   - (snv, nsnv): $g((\forall data_{\text{available}} \leq 0), \emptyset, \{\emptyset\}) = true$; this transition will never be taken since the automaton cannot send when the ground station is unavailable;
   - (snv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \emptyset, \{\emptyset\}) = true$; same as the last point. The only exception is if the snv state is transitive and was jumped in from the state sv;
   - (nsv, sv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0), \emptyset, \{\emptyset\}) = true$
   - (nsv, snv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0, \forall clk_{\text{duration}} \mod d == 0), \{ \text{gs\_visibility\_end} \}, \{\emptyset\}) = true$; contradicting conditions, transition will never be taken;
   - (nsv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \{ \text{gs\_visibility\_end} \}, \{\emptyset\}) = true$
   - (sv, nsv): $g((\forall data_{\text{available}} \leq 0), \{\emptyset\}, \{\emptyset\}) = true$

- (sv, nsv): $g((\forall clk_{\text{duration}} \mod d == 0), \emptyset, \{\emptyset\}) = true$ this transition will never be taken since the automaton cannot stop sending available data while the ground station is available;
- (sv, nsnv): $g((\forall data_{\text{available}} \leq 0, \forall clk_{\text{duration}} \mod d == 0), \{$ gs_visibility_end $\}, \{\emptyset\}) = true$
- (sv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \{$ gs_visibility_end $\}, \{\emptyset\}) = true$
- (sv, snv): $g((\forall clk_{\text{duration}} \mod d == 0), \{$ gs_visibility_end $\}, \{\emptyset\}) = true$; this transition will be immediately followed by the (snv, nsnv), since the guard is also fulfilled

9. The flow functions of the resulting automaton are:
   - nsnv: $\dot{data}_{\text{sent}} = 0,\ \dot{data}_{\text{available}} = 0,\ \dot{clk}_{\text{period}} = 1, \dot{clk}_{\text{duration}} = 0$
   - nsv: $\dot{data}_{\text{sent}} = 0,\ \dot{data}_{\text{available}} = 0,\ \dot{clk}_{\text{period}} = 0, \dot{clk}_{\text{duration}} = 1$
   - snv: $\dot{data}_{\text{sent}} = \kappa,\ \dot{data}_{\text{available}} = -\kappa,\ \dot{clk}_{\text{period}} = 1, \dot{clk}_{\text{duration}} = 0$
   - sv: $\dot{data}_{\text{sent}} = \kappa,\ \dot{data}_{\text{available}} = -\kappa,\ \dot{clk}_{\text{period}} = 0, \dot{clk}_{\text{duration}} = 1$
10. Initial state of the composed automaton is given as:
    $\mathcal{I}^c = (nsnv, \{data_{\text{sent}} = 0,\ data_{\text{available}} = C\} \cup \{clk_{\text{period}} = 0, clk_{\text{duration}} = 0\}) = (nsnv, \{data_{\text{sent}} = 0,\ data_{\text{available}} = C, clk_{\text{period}} = 0, clk_{\text{duration}} = 0\})$

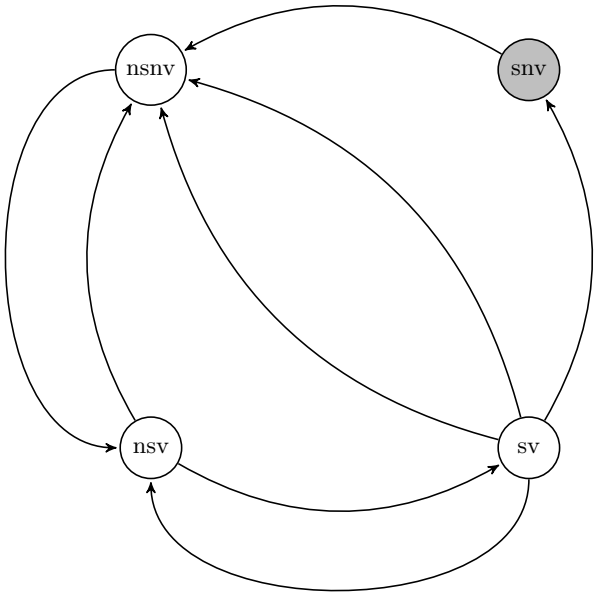


Fig. 4: The resulting control graph for the composed automaton. Labels are omitted for simplicity. Since snv-state is transient, it is coloured.

Extending Taylor Approximation to Hybrid Automata with Integrals

Ruggero Lanotte[1] and Simone Tini[1]

University of Insubria (IT)

Abstract. In [10, 11] we proposed a technique to approximate Hybrid Automata (HA) with Polynomial HA. The idea was to replace functions appearing in formulae with their Taylor Polynomials. Here we extend this technique to HA with formulae admitting integral functions. We prove that we get over-approximations of the original HA. We study the conditions ensuring that: 1. the "distance" between the formulae of the original HA and its approximation get close to 0 when increasing the degree of the Taylor polynomial (syntactical approximation), 2. the "distance" between the configurations reached in n steps by the two HA get close to 0 when increasing the degree of the Taylor polynomial (semantic approximation).

1 Introduction

Hybrid automata [1, 2] (HA, for short) are a widely studied model for *hybrid systems* [13], i.e. systems with discrete and continuous state changes. HA extend classic finite state machines with continuously evolving *variables*, and exhibit two kinds of state changes: discrete jump transitions, occurring instantaneously, and continuous flow transitions, occurring while time elapses. The two kinds of transitions are guarded by *jump conditions* and *activity functions*, resp., which are formulae expressing constraints on the source and target value of the variables. Extensions to HA are considered to deal with particular scopes. As an example in [12, 8] HA are extended with data structures to face with safety and security problems. But most of hybrid system applications is modelling and verifying systems where digital computational processes interact with analog physical ones. In this setting, integrals have several applications. In physics and engineering, where hybrid systems are widely used, we mention: work and impulse, electromagnetism, first moment and center of mass, application in fluid mechanics.

As an example, the HA in Fig. 1 models a controller of a tank. The controller continuously senses the level of water and fills or empties it, aiming to keep the level between m and M litres ($M > m$). The water level, represented by variable x, varies with time depending on input/output flows. When the controller fills the tank (state in), the flow rate depends on time y, and is $1 - \cos(y^2)$ litres/second. Thus, after a time t the water level is increased by $\int_0^t 1 - \cos(y^2)\, dy$, as modeled by activity function ϕ_{in}. When the controller empties the tank (state out), the flow rate at time y is y^2. Thus, after a time t the water level is decreased by $\int_0^y y^2\, dt = \frac{1}{3}t^3$, as modeled by activity function ϕ_{out}.

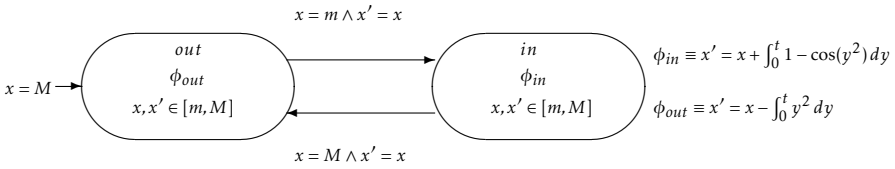


Fig. 1. The tank controller

In this example it is relevant to solve integrals by finding their antiderivatives. Unfortunately, it is well known that the integration problem is "difficult", and in many cases impossible. For instance the antiderivative is non elementary for the filling flow function $1 - \cos(y^2)$ we consider. Indeed the antiderivatives cannot be expressed by an algebraic expression of rationals, exponentials, logarithms, absolute values and trigonometric functions. A classical practical example of non elementary antiderivative function is given by the Gauss integral error $\int_a^b e^{x^2} dx$. Therefore this problem cannot be considered a problem with restricted impact. Moreover in [14] it is showed that the integration problem is undecidable for functions with a non elementary antiderivative.

Our work is inspired by the necessity of using integrals in modeling real problems with HA, meanwhile dealing with the problem of managing and solving integrals, which is in general hard and even impossible for non elementary antiderivatives.

HA are usually used to prove *safety properties* (i.e. properties requiring that a given set of *bad configurations* cannot be reached). The decidability of *reachability* problem (i.e. whether or not a given configuration can be reached) becomes determinant. Unfortunately, for most classes of HA, reachability is undecidable [7] and the introduction of integrals complicates this analysis. However, for some classes of HA, computing the successors (or predecessors) of configurations sets is reasonably efficient, and, therefore, reachability *in a limited number of steps* is decidable. For instance for *Polynomial HA* computing the successors of configuration sets is decidable [15]. A methodology proposed in [6] fills this gap: the idea is to over-approximate an HA H with another HA H' s.t. computing the successors of configuration sets for H' is decidable and the computations of H' are a superset of the set of all the possible computations of H. Hence, if we prove that a *bad* configuration cannot be reached by H' then we can infer that it cannot be reached by the original H. In [6] it is required that the approximation H' is in the class of the *Linear HA*, for which the successors of configuration sets are computable.

The notion of approximation is then strengthened in [6] by *ϵ-approximation*, which is motivated by the need to limit the *error* introduced by the approximation. In [6] an asymptotically complete approximation operator, called *rationally rectangular phase-portrait approximation*, is given which approximates any jump condition or activity function by a predicate satisfied by all points lying in a space consisting of a products of intervals with rational endpoints.

In [10, 11] over-approximations are based on replacing functions over variables with their Taylor polynomial. Since Taylor polynomials allow us to ap-

proximate functions and integrals, in the present paper we extend our technique in [10, 11] to over-approximate HA with integrals. In detail, given any HA H and $k \in \mathbb{N}$, $\boldsymbol{A}(H,k)$ is the set of the Polynomial HA (for which successors of configuration sets is decidable) that are obtained by replacing in jump conditions and activity functions of H each integral $\int_l^u f(\vec{x})dx$ with a polynomial based on Taylor polynomial theory. The resulting polynomial HA over-approximates the original one according to [6]. We study the conditions ensuring that our approximation is asymptotically complete, in the sense that, for each $\epsilon > 0$ there exists some k_0 s.t., for all $k > k_0$, $\boldsymbol{A}(H,k)$ contains only ϵ-approximations for H. This analysis of the error is *syntactic*, meaning that it does not consider the behaviour of H and its approximation. We consider also *semantic* analysis of the error and study conditions ensuring that, when k tends to the infinity, the behaviour of any $H_k \in \boldsymbol{A}(H,k)$ gets close to the behaviour of H.

2 Hybrid Automata

In this section we recall the formalism of Hybrid Automata (see, e.g., [13]).

A *vector* of dimension n over a set U is a tuple $\vec{u} = (u_1,\ldots,u_n)$ in U^n. By $\vec{u}_i$ we denote the i^{th} element u_i. We denote by $\vec{u} \oplus (u)$ the vector $(u_1,\ldots,u_n,u) \in U^{n+1}$. Then, for $\vec{u} = (u_1,\ldots,u_n)$ and $\vec{v} = (v_1,\ldots,v_m)$, we denote by $\vec{u} \oplus \vec{v}$ the vector $(u_1\ldots,u_n,v_1,\ldots,v_m)$ in U^{n+m}. A *space* over U^n is a set of vectors in U^n.

We assume a finite set of *real variables* X ranged over by $x,y,z,w,\ldots$. Each $x \in \mathsf{X}$ can assume values in $\mathsf{Dom}(x) \subseteq \mathbb{R}$. An *evaluation* over X is a mapping $v\colon \mathsf{X} \to \mathbb{R}$ s.t. $v(x) \in \mathsf{Dom}(x)$ for $x \in \mathsf{X}$. For an evaluation v, a variable y and a real $c \in \mathsf{Dom}(x)$, the evaluation $v[y := c]$ is defined by $v[y := c](x) = v(x)$, for $x \neq y$, and $v[y := c](y) = c$. For vectors $\vec{x} = (x_1,\ldots,x_n)$ over X^n and $\vec{u} = (u_1,\ldots,u_n)$ with $u_i \in \mathsf{Dom}(x_i)$, we write $v[\vec{x} := \vec{u}]$ for $v[x_1 := u_1]\ldots[x_n := u_n]$. Then, $v(x_1,\ldots,x_m)$ denotes the vector $(v(x_1),\ldots,v(x_m)) \in \mathbb{R}^m$. We denote by $\vec{\mathsf{X}}$ the vector $(x_1,\ldots,x_{|\mathsf{X}|})$ over $\mathsf{X}^{|\mathsf{X}|}$. Finally, we write $[\vec{\mathsf{X}} := \vec{u}]$ to denote the evaluation v s.t. $v(\vec{\mathsf{X}}) = \vec{u}$.

We assume a set of *function symbols* F, together with an *arity* mapping $\mathsf{r}\colon \mathsf{F} \to \mathbb{N}$ that assigns to each $f \in \mathsf{F}$ its rank $\mathsf{r}(f)$. If $\mathsf{r}(f) = 0$ then f is called a *constant*. We assume a unique *interpretation* I associating to each function symbol $f \in \mathsf{F}$ a continuous function $I(f)\colon \mathsf{Dom}(f) \to \mathbb{R}$ s.t. $\mathsf{Dom}(f) \subseteq \mathbb{R}^{\mathsf{r}(f)}$. Being I unique, sometimes with abuse of notation we use f for $I(f)$. In order to build polynomials with rational coefficients, we require that F contains the constant symbol q with $I(q) = q$ for all $q \in \mathbb{Q}$, and symbols $+$, $\cdot$, $-$ denoting, resp., binary summation, binary multiplication and unary negation over reals.

Definition 1. *The set* $\Phi(F,X)$ *of the* formulae *over* F *and* X *is the least set s.t.:*

- $\Phi(F,X)$ *contains all* basic formulae *of the form*

$$\int_{l_1}^{u_1}\left(\ldots\left(\int_{l_n}^{u_n} f\left(g_1(w_1),\ldots,g_{r(f)}(w_{r(f)})\right)\,\mathrm{d}w_{i_n}\right)\ldots\right)\mathrm{d}w_{i_1} \sim ax, \textit{ where:}$$

- $n \geq 0$ *and, whenever* $n > 0$, *then* $l_1, u_1, \ldots, l_n, u_n \in X \cup \mathbb{Q}$;
- $w_1, \ldots, w_{r(f)} \in X \setminus \{l_1, u_1, \ldots, l_n, u_n\}$ *and* $\{i_1, \ldots, i_n\} \subseteq \{1, \ldots, r(f)\}$;
- $g_1, \ldots, g_{r(f)} \in F$ *are polynomial functions s.t.* $Dom(w_i) \subseteq Dom(g_i)$;
- $f \in F$ *with* $Dom(f) \subseteq g_1(Dom(w_1)) \times \ldots \times g_{r(f)}(Dom(w_{r(f)}))$;
- $\sim$ *is a comparison operator in* $\{<, \leq, =, \geq, >\}$;
- $x \in X$ *and* $a \in \{0, 1\}$;
- $[\min(l, r), \max(l, r)] \subseteq Dom(w_{i_j})$ *for* $l \in Dom(l_j), r \in Dom(u_j), j = 1, \ldots, n$.

– $\neg\phi$ *is in* $\Phi(X, F)$ *whenever* ϕ *is in* $\Phi(X, F)$;
– $\phi_1 \vee \phi_2$ *and* $\phi_1 \wedge \phi_2$ *are in* $\Phi(X, F)$ *whenever both* ϕ_1 *and* ϕ_2 *are in* $\Phi(X, F)$;
– $\forall y.\, \phi$ *and* $\exists y.\, \phi$ *are in* $\Phi(X, F)$ *whenever* $y \in X$ *and* ϕ *is in* $\Phi(X, F)$.

The subset of polynomial formuale *is obtained by restricting to (i) those* $f \in F$ *that are polynomial functions s.t.* $Dom(f)$ *is a product of intervals with bounds in* $\mathbb{Q} \cup \{\pm\infty\}$, *(ii) those variables* $x \in X$ *s.t.* $Dom(x)$ *is an interval with bounds in* $\mathbb{Q} \cup \{\pm\infty\}$.

In [11] we restricted to basic formulae of Def. 1 with $n = 0$, i.e. general continuous function without integrals. The definition of basic formulae could appear restrictive at first glance. We argue that Def. 1 gives us expressiveness and flexibility by some examples:

1. By existential quantification, arbitrary expressions be compared. For instance, $e^{x+\sin y} \leq x/(y^2+1)$ is expressed by $\exists z.\, (e^{x+\sin y} \leq z \wedge x/(y^2+1) = z)$.
2. By existential quantification, we give to the user as much freedom as possible in choosing the functions to be approximated. For instance, for $f, g \in \mathsf{F}$, we can rewrite a formula $h(\vec{x}) \sim ax$ with $h = f \circ g$ by $\exists y.\, g(\vec{x}) = y \wedge f(y) \sim ax$. In the first case the function $f \circ g$ is approximated, in the second case f and g are approximated separately, e.g. in order to approximate the exponential and the sin separately, the formula $\exists z.\, (e^{x+\sin y} \leq z \wedge x/(y^2+1) = z)$ in item 1 can be rewritten as $\exists z_1 . \exists z_2.\, (e^{z_1} \leq z_2 \ \wedge\ x + \sin y = z_1 \ \wedge\ x/(y^2+1) = z_2)$.
3. Also arbitrary expressions dealing with integrals can be compared. For instance, the expression $h(x) + \int_x^y f(z)\mathrm{d}z = \int_x^y g(z)\mathrm{d}z$ can be expressed by $\exists w_1 \exists w_2.\ \int_x^y f(z)\mathrm{d}z = w_1 \ \wedge\ \int_x^y g(z)\mathrm{d}z = w_2 \ \wedge\ h(x) + w_1 = w_2$.
4. Expressions with integrals can be arguments of functions. For instance, $\cos\left(\int_x^y f(z)\mathrm{d}z\right) > 0$ can be expressed by $\exists w.\ \cos(w) > 0 \ \wedge\ \int_x^y f(z)\mathrm{d}z = w$, and $\int_0^5 f\left(x, \int_0^3 g(y)\mathrm{d}y\right)\mathrm{d}x \leq 7$ by $\exists z.\ \int_0^3 g(y)\mathrm{d}y = z \ \wedge\ \int_0^5 f(x, z)\mathrm{d}x \leq 7$.
5. We can deal also with general bounds for integrals. For instance the formula $\int_4^{e^x} f(y)\mathrm{d}y \leq x$ can be expressed by $\exists z.\ \int_4^z f(y)\mathrm{d}y \leq x \ \wedge\ e^x = z$.

We write $v \models \phi$ to denote that *the evaluation* v *satisfies the formula* ϕ. Relation $\models$ is defined inductively as follows:

– $v \models \int_{l_1}^{u_1} \left(\ldots \left(\int_{l_n}^{u_n} f\left(g_1(w_1), \ldots, g_{\mathrm{r}(f)}(w_{\mathrm{r}(f)})\right)\mathrm{d}w_{i_n}\right)\ldots\right)\mathrm{d}w_{i_1} \sim ax$ iff

$$\int_{v(l_1)}^{v(u_1)} \left(\ldots \left(\int_{v(l_n)}^{v(u_n)} I(f)\left(I(g_1)(e_1), \ldots, I(g_{\mathrm{r}(f)})(e_{\mathrm{r}(f)})\right)\mathrm{d}w_{i_n}\right)\ldots\right)\mathrm{d}w_{i_1} \sim I(a)v(x)$$

where either $e_j = w_j$, if $j \in \{i_1, \ldots, i_n\}$, or $e_j = v(w_j)$, otherwise.

– $v \models \neg\phi$ iff $v \not\models \phi$ (namely $v \models \phi$ does not hold).
– $v \models \phi_1 \wedge \phi_2$ (resp. $v \models \phi_1 \vee \phi_2$) iff $v \models \phi_1$ and $v \models \phi_2$ (resp. $v \models \phi_1$ or $v \models \phi_2$).
– $v \models \forall y.\, \phi$ (resp. $v \models \exists y.\, \phi$) iff $v[y := c] \models \phi$ for all (resp. for some) $c \in \mathsf{Dom}(y)$.

For a formula $\phi \in \Phi(\mathsf{F}, \mathsf{X})$, let $[\![\phi]\!]$ denote the set $\{v : \mathsf{X} \to \mathbb{R} \mid v \models \phi\}$ of the evaluations satisfying ϕ. Two formulae ϕ_1, ϕ_2 are *equivalent* iff $[\![\phi_1]\!] = [\![\phi_2]\!]$.

Definition 2. *The subset of the* normal forms *in $\Phi(F, X)$ contains the formulae of the form $Q_1 y_1.\ \ldots\ Q_m y_m.\ \phi$, where: (i) $Q_i \in \{\forall, \exists\}$ for $i = 1, \ldots, m$; (ii) ϕ contains neither quantifiers nor negations; (iii) ϕ contains only relations in $\{<, \leq\}$; (iv) all basic formulae in ϕ are of the following form, for $n \leq r(f)$ and $z_1, \ldots, z_n \in X$:*

$$\int_0^{z_1} \left(\ldots \left(\int_0^{z_n} f\left(g_1(w_1), \ldots, g_{r(f)}(w_{r(f)})\right) \mathrm{d}w_n \right) \ldots \right) \mathrm{d}w_1 \sim ax.$$

Proposition 1. *Given any formula $\phi \in \Phi(X, F)$, there exists a normal form equivalent to ϕ that can be constructed from ϕ.*

E.g. $\int_4^z e^y \mathrm{d}y \leq w$ is equivalent to the normal form $\exists x. \int_0^x e^{y+4} \mathrm{d}y \leq w \wedge z - 4 = x$.

Definition 3. *An* Hybrid Automaton *(*HA *for short) H over X and F is a tuple of the form $H = \langle \phi_{init}, Q, q_0, T, Act \rangle$, where:*

– *$\phi_{init} \in \Phi(F, X)$ is the* initial condition.
– *Q is a finite* set of states, *and $q_0 \in Q$ is the* initial state.
– *$T \subseteq Q \times \Phi(F, \{x_1, \ldots, x_{|X|}, x'_1, \ldots, x'_{|X|}\}) \times Q$ is a finite* set of transitions. *Variables $x'_1, \ldots, x'_{|X|}$ represent the values taken by $x_1, \ldots, x_{|X|}$ after the firing of a transition.*
– *$Act : Q \to \Phi(F, \{x_1, \ldots, x_{|X|}, t, x'_1, \ldots, x'_{|X|}\})$ is the* activity function *assigning to each state q a formula $Act(q)$. Variable t represents time elapsing.* [1]

Then, H is a Polynomial Hybrid Automaton *(*PHA *for short) iff ϕ_{init}, $Act(q)$ for all states q and ϕ for each transition (q, ϕ, q') are all polynomial formulae.*

Example 1. The tank controller represented in Fig. 1 has two states: in state in the controller fills the tank, in state out the controller empties the tank. The jump condition $x = m \wedge x' = x$ (resp. $x = M \wedge x' = x$) ensures that the jump from out to in (resp. from in to out) happens when the level of the water is m (resp. M), and the firing of the transition does not cause any change in the water level.

In state in, the water flow rate at time y is $1 - \cos(y^2)$. Hence, staying in in for t units of time causes a water level growing of $\int_0^t 1 - \cos(y^2)\, dy$. This is modelled by the activity function ϕ_{in}, which can be written in normal form in several ways. Let ϕ' be the formula $x \leq M \wedge x \geq m \wedge x' \leq M \wedge x' \geq m$, or $x, x' \in [m, M]$ for short. Given the functions f, g s.t. $f(y) = 1 - \cos(y^2)$ and $g(y) = -\cos(y^2)$, we can write ϕ_{in} in the two following ways, which are semantically equivalent:

$$\phi^1_{in} \equiv \phi' \wedge x' - x = z \wedge \int_0^t f(y)\, dy = z \qquad \phi^2_{in} \equiv \phi' \wedge x' - x - t = z \wedge \int_0^t g(y)\, dy = z$$

[1] Note that invariants can be expressed by means of universal quantifiers (see [11]).

However, when non-polynomial functions are approximated by their Taylor polynomials, in the former case we approximate f and in the latter g.

In state out, the water flow rate at time y is y^2. Hence, staying in state out for t units of time causes a water level decrement of $\int_0^t y^2\ dy$. This is modelled by the activity function $\phi_{out} \equiv \phi' \wedge x - x' = z \wedge \int_0^t y^2\, dy = z$. Obviously $\int_0^t y^2\, dy = \frac{t^3}{3}$. Therefore in this case no approximation is necessary.

A *configuration* of an HA H is a pair $(q, \vec{u})$, with $q \in Q$ and $\vec{u} = (u_1, \ldots, u_{|X|})$ a vector in $\mathbb{R}^{|X|}$ representing that each variable x_i has value u_i. H can evolve from $(q, \vec{u})$ to $(q', \vec{u}')$, written $(q, \vec{u}) \to (q', \vec{u}')$, by an activity or transition step. An *activity step* describes the evolution from $(q, \vec{u})$ due to remaining in q and passing of time. In δ time units, $Act(q)$ takes H to a new evaluation of the variables:

$$\text{if } \delta \geq 0 \text{ and } [\vec{X} := \vec{u}, t := \delta, \vec{X}' := \vec{u}'] \models Act(q), \text{ then } (q, \vec{u}) \to (q, \vec{u}').$$

A *transition step* describes the evolution from $(q, \vec{u})$ due to a transition from q:

$$\text{if } (q, \phi, q') \in T \text{ and } [\vec{X} := \vec{u}, \vec{X}' := \vec{u}'] \models \phi, \text{ then } (q, \vec{u}) \to (q', \vec{u}').$$

A *run* is a sequence of (activity and transition) steps $(q_0, \overrightarrow{u_0}) \to \ldots \to (q_i, \overrightarrow{u_i}) \ldots$ with q_0 the initial state and $[\vec{X} := \overrightarrow{u_0}] \in [\![\phi_{init}]\!]$. A configuration $(q, \vec{u})$ is *reachable in n steps* iff there is a run $(q_0, \overrightarrow{u_0}) \to \ldots \to (q_n, \overrightarrow{u_n}) \ldots$ s.t. $q_n = q$ and $\overrightarrow{u_n} = \vec{u}$. A configuration is *reachable* iff it is reachable in n steps for some $n \geq 0$.

A *region* R of a HA H is a set of configurations. The region reachable by H from a region R is denoted $\mathsf{Post}(R, H)$. Formally: $\mathsf{Post}(R, H) = \{(q', \vec{u}') \mid \exists (q, \vec{u}) \in R$ such that $(q, \vec{u}) \to (q', \vec{u}')\}$. Let $\mathsf{Post}^n(H)$ denote either the region $\{(q_0, \overrightarrow{u_0}) \mid [\vec{X} := \overrightarrow{u_0}] \in [\![\phi_{init}]\!]\}$, if $n = 0$, or the region $\mathsf{Post}(\mathsf{Post}^{n-1}(H), H)$, if $n > 0$. Moreover, let $\mathsf{Post}(H)$ denote the region $\bigcup_{n \in \mathbb{N}} \mathsf{Post}^n(H)$. The following result is folklore.

Theorem 1. *For each $n \in \mathbb{N}$, a configuration $(q, \vec{u})$ is reachable in n steps iff $(q, \vec{u}) \in \mathsf{Post}^n(H)$. Hence $(q, \vec{u})$ is reachable iff $(q, \vec{u}) \in \mathsf{Post}(H)$.*

The following result follows from Tarski's results [15] and from the fact that the antiderivative of a polynomial is a polynomial.

Theorem 2. *If H is polynomial and $n \in \mathbb{N}$ then $(q, \vec{u}) \in \mathsf{Post}^n(H)$ is decidable.*

3 Taylor Approximation

The i^{th} derivative of $f \in \mathsf{F}$ wrt. coordinate j^{th} is denoted $\mathsf{D}^i_j f$. Let C^k denote the set of the functions that are derivable k times, namely $f \in \mathsf{C}^k$ iff $\mathsf{D}_1^{j_1} \ldots \mathsf{D}_{\mathsf{r}(f)}^{j_{\mathsf{r}(f)}} f$ exists whenever $j_1 + \ldots + j_{\mathsf{r}(f)} = k$.

Definition 4. *Assume a function $f \in C^k$ and a vector $\vec{v} \in \mathsf{Dom}(f)$. The* polynomial of Taylor of degree k for f wrt. $\vec{v}$ *is defined by*

$$P^k(f, \vec{w}, \vec{v}) = \sum_{j_1 + \ldots + j_{r(f)} \leq k} \frac{(D_1^{j_1} \ldots D_{r(f)}^{j_{r(f)}} f)(\vec{v}) \cdot (w_1 - v_1)^{j_1} \ldots (w_{r(f)} - v_{r(f)})^{j_{r(f)}}}{j_1! \cdot \ldots \cdot j_{r(f)}!}$$

where $\vec{w}$ is the vector of variables $(w_1, \ldots, w_{r(f)})$. For $\vec{u} \in Dom(f)$, the value $r^k(f, \vec{u}, \vec{v})$ defined by $r^k(f, \vec{u}, \vec{v}) = f(\vec{u}) - P^k(f, \vec{u}, \vec{v})$ is called the remainder.

The intuition is that $\mathsf{P}^k(f, \vec{w}, \vec{v})$ is a polynomial that approximates $f(\vec{w})$, and the error of the approximation in $\vec{u} \in \mathsf{Dom}(f)$ is given by $\mathsf{r}^k(f, \vec{u}, \vec{v})$. This error is quantified by the following result, known as Lagrange Remainder Theorem.

Theorem 3 (Lagrange). *For a function $f \in C^{k+1}$, a convex set $S \subseteq Dom(f)$ and two vectors $\vec{u}, \vec{v}$ in S, there exists a vector $\vec{z}$ on the segment linking $\vec{u}$ and $\vec{v}$ s.t.:*

$$r^k(f, \vec{u}, \vec{v}) = \sum_{j_1 + \ldots + j_{r(f)} = k+1} \frac{(D_1^{j_1} \ldots D_{r(f)}^{j_{r(f)}} f)(\vec{z}) \cdot (u_1 - v_1)^{j_1} \ldots (u_{r(f)} - v_{r(f)})^{j_{r(f)}}}{j_1! \cdot \ldots \cdot j_{r(f)}!}.$$

Our aim is to give an upper bound to $|\mathsf{r}^k(f, \vec{u}, \vec{v})|$, under suitable hypothesis.

Definition 5. A function $f \in \mathsf{C}^{k+1}$ is analytic in $S \subseteq \mathsf{Dom}(f)$ *if there are two constants C, L s.t., for all $j_1, \ldots, j_{r(f)}$ with $j_1 + \cdots + j_{r(f)} \leq k+1$ and $\vec{z} \in S$, we have*

$$|(D_1^{j_1} \ldots D_n^{j_{r(f)}} f)(\vec{z})| \leq L \cdot C^{j_1 + \ldots + j_{r(f)}}.$$

Then, f is analytic *if f is analytic in $Dom(f)$ and $Dom(f)$ is convex.*

Example 2. Trigonometric functions are analytic. For instance, for the function $\sin(x)$ it is sufficient to take the constants $L = C = 1$. Exponential and logarithmic functions are analytic in finite intervals. For instance, for function e^{2x} and interval $[0, 10]$, it is sufficient to take the constants $C = 2$ and $L = e^{20}$.

Let us assume an analytic function $f \in \mathsf{C}^{k+1}$. Then, for $\hat{C}$ and $\hat{L}$ the minimal values satisfying the condition of Def. 5, for any k we denote with $\mathsf{C}(f, k)$ the value $\hat{L} \cdot \hat{C}^{k+1}$. Moreover, let $\mathsf{R}^k(f, \vec{w}, \vec{v})$ denote the polynomial over $\vec{w}$ defined by

$$\mathsf{R}^k(f, \vec{w}, \vec{v}) = \frac{\mathsf{C}(f, k) \cdot (\mathsf{r}(f))^{k+1} \cdot \prod_{j=1}^{\mathsf{r}(f)} ((w_j - v_j)^{2 \cdot \left\lceil \frac{k+1}{2} \right\rceil} + 1)}{\left\lfloor \frac{k+1}{\mathsf{r}(f)} \right\rfloor !}$$

By definition, $\mathsf{R}^k(f, \vec{u}, \vec{v})$ is an upper bound to $|\mathsf{r}^k(f, \vec{u}, \vec{v})|$ for all $\vec{u} \in \mathsf{Dom}(f)$. Moreover, $\mathsf{R}^k(f, \vec{u}, \vec{v})$ gets close to 0 when k tends to the infinity. Formally:

Proposition 2 ([11]). *Let $f \in F$ be analytic. Then, for all $\vec{u}, \vec{v} \in Dom(f)$ we have: (1) $|r^k(f, \vec{u}, \vec{v})| \leq R^k(f, \vec{u}, \vec{v})$, and (2) $\lim_{k \to \infty} R^k(f, \vec{u}, \vec{v}) = 0$.*

From $|\mathsf{r}^k(f, \vec{u}, \vec{v})| \leq \mathsf{R}^k(f, \vec{u}, \vec{v})$, $f(\vec{u}) = \mathsf{r}^k(f, \vec{u}, \vec{v}) + \mathsf{P}^k(f, \vec{u}, \vec{v})$ and monotonicity of the integral we get the following result.

Proposition 3. *Let $f \in F$ be analytic and $\vec{v} \in Dom(f)$. Then for all vectors $\vec{e} = (g_1(w_1), \ldots, g_n(w_n)) \oplus (g_{n+1}(c_{n+1}), \ldots, g_{r(f)}(c_{r(f)}))$ with $c_{n+1}, \ldots, c_{r(f)} \in \mathbb{R}$, and $r_1, \ldots, r_n \in \mathbb{R}$ s.t. $g([0, r_1]) \times \cdots \times g([0, r_n]) \times \{g_{n+1}(c_{n+1})\} \times \cdots \times \{g_{r(f)}(c_{r(f)})\} \subseteq Dom(f)$, we have*

$$\int_0^{r_1} \left(\ldots \left(\int_0^{r_n} f(\vec{e}) \, \mathrm{d}w_n \right) \ldots \right) \mathrm{d}w_1 \geq \int_0^{r_1} \left(\ldots \left(\int_0^{r_n} \left(P^k(f, \vec{e}, \vec{v}) - R^k(f, \vec{e}, \vec{v}) \right) \mathrm{d}w_n \right) \ldots \right) \mathrm{d}w_1.$$

If we replace $f(g_1(w_1),\dots,g_{r(f)}(w_{r(f)}))$ with $\mathsf{P}^k(f,(g_1(w_1),\dots,g_{r(f)}(w_{r(f)}),\vec{v}) - \mathsf{R}^k(f,(g_1(w_1),\dots,g_{r(f)}(w_{r(f)}),\vec{v})$ in a basic formula, by Prop. 3 we get a less demanding formula, provided the operator $\sim$ is in $\{<,\leq\}$, like in normal forms.

4 Approximation of Hybrid Automata

Approximations of HA are obtained by weakening formulae [6].

Definition 6 ([6]). *An HA H' is an* approximation *of an HA H if H' is obtained from H by replacing each formula ϕ in H with a formula ϕ' s.t. $[\![\phi]\!] \subseteq [\![\phi']\!]$.*

We aim to give a notion of approximation for HA respecting Def. 6. We start with a notion of approximation for normal forms inspired by Prop. 3.

Definition 7. *For a normal form $\phi \in \Phi(X,F)$ and $k \in \mathbb{N}$, if each $f \in F \setminus \{+,\cdot,-\}$ that appears in ϕ is derivable $k+1$ times and is analytic, then the* approximation of ϕ of degree k *is the set of formulae denoted $\mathbf{A}(\phi,k)$ defined inductively wrt. ϕ as follows:*

1. *If $\phi \equiv \int_0^{z_1}\left(\dots\left(\int_0^{z_n} f(g_1(w_1),\dots,g_{r(f)}(w_{r(f)}))\,\mathrm{d}w_n\right)\dots\right)\mathrm{d}w_1 \sim ax$, then either $\mathbf{A}(\phi,k)$ is the singleton $\{\phi\}$, if f is a polynomial, or $\mathbf{A}(\phi,k)$ contains all the formulae*

$$\phi_{k,\vec{v}} \equiv \int_0^{z_1}\left(\dots\left(\int_0^{z_n}\left(P^k(f,\overrightarrow{g(w)},\vec{v}) - R^k(f,\overrightarrow{g(w)},\vec{v})\right)\mathrm{d}w_n\right)\dots\right)\mathrm{d}w_1 \sim ax$$

   with $\overrightarrow{g(w)} = (g_1(w_1),\dots,g_{r(f)}(w_{r(f)}))$ and $\vec{v} \in Dom(f)$;
2. *If $\phi \equiv \phi^1 \wedge \phi^2$ then $\mathbf{A}(\phi,k) = \{\phi_k^1 \wedge \phi_k^2 \mid \phi_k^1 \in \mathbf{A}(\phi^1,k)$ and $\phi_k^2 \in \mathbf{A}(\phi^2,k)\}$;*
3. *If $\phi \equiv \phi^1 \vee \phi^2$ then $\mathbf{A}(\phi,k) = \{\phi_k^1 \vee \phi_k^2 \mid \phi_k^1 \in \mathbf{A}(\phi^1,k)$ and $\phi_k^2 \in \mathbf{A}(\phi^2,k)\}$;*
4. *If $\phi \equiv \exists y.\,\phi'$ then $\mathbf{A}(\phi,k) = \{\exists y.\,\phi_k' \mid \phi_k' \in \mathbf{A}(\phi',k)\}$;*
5. *If $\phi \equiv \forall y.\,\phi'$ then $\mathbf{A}(\phi,k) = \{\forall y.\,\phi_k' \mid \phi_k' \in \mathbf{A}(\phi',k)\}$.*

Let us prove that all formulae in $\boldsymbol{A}(\phi,k)$ are less demanding than ϕ.

Theorem 4. *For a normal form ϕ and $k \in \mathbb{N}$ s.t. $\mathbf{A}(\phi,k)$ is defined, then $[\![\phi]\!] \subseteq [\![\phi']\!]$ for all $\phi' \in \mathbf{A}(\phi,k)$.*

Proof (sketch). By structural induction over ϕ. The proof of the base case follows from Prop. 3, the inductive steps are standard.

From the approximation of normal forms we get an approximation of HA.

Definition 8. *Assume an HA H s.t. $\mathbf{A}(\phi,k)$ is defined for each formula ϕ in H. The* approximation of degree k for H *is the set of the PHA denoted $\mathbf{A}(H,k)$ that are obtained from H by replacing each formula ϕ in H with some formula in $\mathbf{A}(\phi,k)$.*

An immediate corollary of Thm. 4 states that Def. 8 respects Def. 6.

Corollary 1. *Given any HA H and $k \in \mathbb{N}$, all PHA in $\mathbf{A}(H,k)$ are approximations of H according to Def. 6.*

Example 3. Let us consider the tank controller H of Ex. 1 where $\phi_{in} \equiv \phi_{in}^2$. The set $\boldsymbol{A}(H,4)$ contains the automaton obtained from H by approximating function g in ϕ_{in} by choosing the real 0 as vector $\vec{v}$. (ϕ_{out} does not change since all functions are polynomial). Since $D_w^k(-\cos(w)) = -D_w^k \cos(w) = -\cos(w + k \cdot \frac{\pi}{2})$, it holds that $P^4(\cos, y^2, 0) \equiv -\cos(0) - \cos(\frac{\pi}{2}) \cdot (y^2)^1 - \cos(2 \cdot \frac{\pi}{2}) \cdot \frac{(y^2)^2}{2!} - \cos(3 \cdot \frac{\pi}{2}) \cdot \frac{(y^2)^3}{3!} - \cos(4 \cdot \frac{\pi}{2}) \cdot \frac{(y^2)^4}{4!} = -1 + \frac{y^4}{2} - \frac{y^8}{24}$. Moreover, $R^4(\cos, y^2, 0) = \mathsf{C}(\cos, 4) \cdot \frac{(y^2)^6+1}{120}$. Now, $\mathsf{C}(\cos, 4) = \max\{|-\cos(w + 4 \cdot \frac{\pi}{2})| : w \in \mathsf{Dom}(g)\} = 1$, therefore we have that

$$\left(\phi_{in}^2\right)_{4,0} \equiv \phi' \wedge x' - x - t = z \wedge \int_0^t -1 + \frac{y^4}{2} - \frac{y^8}{24} + \frac{y^{12}+1}{120}\, dy = z.$$

The behaviour of any PHA H_k in $\boldsymbol{A}(H,k)$ approximates the behaviour of H, meaning that any configuration reachable by H is reachable also by H_k, in the same number of steps.

Theorem 5. *Given any HA H and $k, n \in \mathbb{N}$, if $\mathbf{A}(H,k)$ is defined, then, for all PHA $H_k \in \mathbf{A}(H,k)$ it holds that $\mathsf{Post}^n(H) \subseteq \mathsf{Post}^n(H_k)$.*

Proof (sketch). By Thm. 4 and the monotonicity of Post.

Thm. 5 gives us a sound method for showing that H cannot reach some *bad* configuration $(q, \vec{u})$ in n steps. In fact, by Thm. 2 it is computable if $(q, \vec{u})$ can be reached in n steps by a PHA H_k in $\boldsymbol{A}(H,k)$. By Thm. 5 if $(q, \vec{u})$ cannot be reached in n steps by H_k then it cannot be reached in n steps by H as well.

5 Analysis of the Error

In order to limit the error introduced by the approximation, Def. 6 is strengthened in [6] by the notion of ϵ-approximation, which requires that any vector satisfying a formula ϕ' of the approximation H' must be "close" to at least one vector satisfying the corresponding formula ϕ in the original HA H. We reformulate this notion in terms of a notion of neighborhood of a space in $\mathbb{R}^n$.

Given two vectors $\vec{u} = (u_1, \ldots, u_n)$ and $\vec{v} = (v_1, \ldots, v_n)$ in $\mathbb{R}^n$, let $d(\vec{u}, \vec{v})$ denote their *distance* $d(\vec{u}, \vec{v}) = \sqrt{(u_1 - v_1)^2 + \ldots + (u_n - v_n)^2}$.

Given a vector $\vec{v}$ and a real $\epsilon > 0$, let $\mathsf{N}(\vec{v}, \epsilon)$ denote the space of vectors $\{\vec{u} \mid d(\vec{v}, \vec{u}) \leq \epsilon\}$. Then, for a space $S \subseteq \mathbb{R}^n$ and a real $\epsilon \geq 0$, the *neighborhood of ray ϵ of space S* is the set of spaces $\mathsf{N}(S, \epsilon) = \{S' \supseteq S \mid \forall \vec{v}' \in S' \ \exists \vec{v} \in S \text{ s.t. } d(\vec{v}, \vec{v}') \leq \epsilon\}$.

Definition 9. *A formula $\phi' \in \Phi(\vec{F}, \vec{X})$ is an* ϵ-approximation *of a formula $\phi \in \Phi(\vec{F}, \vec{X})$ iff $\{v(\vec{X}) \mid v \in [\![\phi']\!]\} \in \mathsf{N}(\{v(\vec{X}) \mid v \in [\![\phi]\!]\}, \epsilon)$.*

Definition 10 ([6]). *An HA H' is an* ϵ-approximation *of an HA H if H' is obtained from H by replacing each formula ϕ in H with a formula ϕ' s.t. ϕ' is an ϵ-approximation of ϕ.*

Our aim is to study the conditions over formulae in H ensuring that, for any $\epsilon > 0$, there exists some $k_0 \in \mathbb{N}$ s.t. for all $k > k_0$ we have that the set $\boldsymbol{A}(H,k)$ contains only ϵ-approximations for H. In [11] we argued that formulae of the form $f(\vec{x}) \sim c$ with $\sim \in \{<,>\}$ should be avoided, since they describe open sets. In [11] we argued also that we can manage only formulae constraining variables within bounded intervals, thus avoiding variables that can tend to the infinity.

Definition 11. *A normal form $\phi \in \Phi(F,X)$ is* bounded *iff for any variable x in ϕ we have that $\mathsf{Dom}(x) = [l_x, r_x]$, for suitable rationals $l_x, r_x \in \mathbb{Q}$, and for each function f in ϕ we have that $\mathsf{Dom}(f) = [l_1, r_1] \times \ldots \times [l_{r(f)}, r_{r(f)}]$, for suitable rationals $l_1, r_1, \ldots, l_{r(f)}, r_{r(f)} \in \mathbb{Q}$.*

5.1 Syntactical Analysis of the Error

First of all let we give the intuition why for bounded normal formulae with comparison operator $\leq$ we have that and for all $\epsilon > 0$ there exists some k_0 s.t. for all $k > k_0$, $\boldsymbol{A}(\phi,k)$ contains only ϵ-approximations of ϕ.

Consider a normal form $\phi \equiv \int_0^d f(x,y)\mathrm{d}x \leq 0$. All formulae in $\boldsymbol{A}(\phi,k)$ are of the form $\int_0^d \left(\mathsf{P}(f,(x,y),(c_x,c_y)) - \mathsf{R}^k(f,(x,y),(c_x,c_y))\right)\mathrm{d}x \leq 0$ for a vector $(c_x,c_y) \in \mathsf{Dom}(f)$. Since ϕ is bounded, we can split $\mathsf{Dom}(\int_0^d f(x,y)\mathrm{d}x)$ (which is a function over variable y) in m closed intervals $S_1, \ldots, S_m$ of size strictly $< \epsilon$. Let $i_1, \ldots, i_l \in \{1, \ldots, m\}$ be the indexes s.t. no evaluation in $[\![\phi]\!]$ maps y to $S_{i_1} \cup \ldots \cup S_{i_l}$, namely there is no $u \in S_{i_1} \cup \ldots \cup S_{i_l}$ satisfying $\int_0^d f(x,u)\mathrm{d}x \leq 0$. It is enough to show that no evaluation v_k in any $[\![\phi_k]\!]$ with $\phi_k \in \boldsymbol{A}(\phi,k)$ maps y to $S_{i_1} \cup \ldots \cup S_{i_l}$. In fact, if $v_k(y) \in S_j$ with $j \notin \{i_1, \ldots, i_n\}$, by the definition of $j_1, \ldots, j_l$ there is some $v \in [\![\phi]\!]$ with $v(y) \in S_j$ and, since the size of S_j is bounded by ϵ, we infer $v_k(y) - v(y) < \epsilon$.

Hence the target is to show that there exists some k_0 s.t. for all $k > k_0$ we have that for all $u \in S_{i_1} \cup \ldots \cup S_{i_l}$ the following inequality holds:

$$\int_0^d \left(\mathsf{P}(f,(x,u),(c_x,c_y)) - \mathsf{R}^k(f,(x,u),(c_x,c_y))\right)\mathrm{d}x > 0. \tag{1}$$

Since $S_{i_1} \cup \ldots \cup S_{i_l}$ is a closed set, $\int_0^d f(x,u)\mathrm{d}x$ is a continuous function (which follows by the continuity of f), and the comparison symbol $\leq$ guarantees that $\int_0^d f(x,u)\mathrm{d}x$ is strictly positive in $S_{i_1} \cup \ldots \cup S_{i_l}$, we can define $\vartheta = \min\{\int_0^d f(x,u)\mathrm{d}x \mid u \in S_{i_1} \cup \ldots \cup S_{i_l}\}$. Since $[0,d] \times S_{i_1} \cup \ldots \cup S_{i_l}$ is a closed set, we can define $e_k = \max\{\mathsf{R}^k(f,(u',u),(c_x,c_y))) \mid u' \in [0,d] \wedge u \in S_{i_1} \cup \ldots \cup S_{i_l}\}$. By Prop. 2.2 we can find a k_0 s.t. for all $k > k_0$, $e_k < \vartheta/(2 \cdot d)$. Assume $k > k_0$. We show Eq. 1 by

$$
\begin{aligned}
&\int_0^d (\mathsf{P}(f,(x,u),(c_x,c_y)) - \mathsf{R}^k(f,(x,u),(c_x,c_y)))\mathrm{d}x \\
\geq &\int_0^d (\mathsf{P}(f,(x,u),(c_x,c_y)) - e_k\mathrm{d}x \\
= &\int_0^d (\mathsf{P}(f,(x,u),(c_x,c_y)) + \mathsf{r}^k(f,(x,u),(c_x,c_y)) - \mathsf{r}^k(f,(x,u),(c_x,c_y)) - e_k\mathrm{d}x \\
= &\int_0^d f(x,u) - \mathsf{r}^k(f,(x,u),(c_x,c_y)) - e_k\mathrm{d}x \geq \int_0^d f(x,u) - e_k - e_k\mathrm{d}x \\
> &\int_0^d f(x,u) - \frac{\vartheta}{d}\mathrm{d}x = \int_0^d f(x,u)\mathrm{d}x - \vartheta \geq \int_0^d f(x,u)\mathrm{d}x - \int_0^d f(x,u)\mathrm{d}x = 0
\end{aligned}
$$

with the first inequality by the definition of e_k and the monotonicy of the integral, the second by $|\mathsf{r}^k(f,(x,u),(c_x,c_y))| \leq \mathsf{R}^k(f,(x,u),(c_x,c_y))$ and the definition of e_k, the third by $e_k < \frac{\vartheta}{2\cdot d}$, and the last inequality by the definition of ϑ.

Theorem 6. *Given any bounded normal form $\phi \in \Phi(F,X)$ s.t. each subformula*

$$
\int_0^{z_1} \left(\ldots \left(\int_0^{z_n} f(g_1(w_1),\ldots,g_{r(f)}(w_{r(f)}))\,\mathrm{d}w_n \right) \ldots \right) \mathrm{d}w_1 \sim ax
$$

in ϕ is such that $\sim$ is $\leq$, then, for each $\epsilon > 0$, there exists some k_0 s.t. for each $k > k_0$, the set $\mathbf{A}(\phi,k)$ contains only ϵ-approximations for ϕ.

The result above can be immediately extended to automata.

Corollary 2. *Given any HA H s.t. each formula in H satisfies the hypothesis of Thm. 6, then, for each $\epsilon > 0$, there exists some k_0 s.t. for each $k > k_0$ the set $\mathbf{A}(H,k)$ contains only ϵ-approximations for H.*

5.2 Semantical Analysis of the Error

Our aim is to measure how close the behaviors of the PHA in $\boldsymbol{A}(H,k)$ and the behavior of H are.

Definition 12. *Let $\epsilon \geq 0$. The* neighborhood of ray ϵ of a region R *is the set of regions $\mathsf{N}(R,\epsilon) = \{R' \supseteq R \mid \forall (q',\vec{u}') \in R'.\ \exists\ (q,\vec{u}) \in R.\ \ q = q' \text{ and } d(\vec{u},\vec{u}') \leq \epsilon\}$.*

Under the hypothesis of Thm. 6, for all $n \in \mathbb{N}$, if k tends to the infinity, then the behavior of length at most n of each PHA $H_k \in \boldsymbol{A}(H,k)$ gets close to the behavior of H, in the sense that $\mathsf{Post}^n(H_k)$ is in a neighborhood of $\mathsf{Post}^n(H)$ of ray arbitrarily small. This comes from the fact that $\mathsf{Post}^n(H_k)$ can be expressed by means of a formula by using existential quantifications.

Theorem 7. *Consider an HA H s.t. each formula in H satisfies the hypothesis of Thm. 6. For each $\epsilon > 0$ and $n \in \mathbb{N}$, there exists some k_0 s.t., for all $k > k_0$, we have $\mathsf{Post}^n(H_k) \in \mathsf{N}(\mathsf{Post}^n(H),\epsilon)$ for all $H_k \in \mathbf{A}(H,k)$.*

6 Conclusion and Future Works

In this paper we have defined syntactical over–approximations for Hybrid Automata enriched with integrals. The approximation is based on Taylor polynomials. We have also studied their syntactical and semantical convergence w.r.t. the original specifications.

As future work we will also study under–approximations based on the same technique. The idea is to define the *under–approximation* of degree k of a formula ϕ by using the polynomial which approximates the reminder to increasing the Taylor polynomial. Moreover we can extend our work with function variables by following the theory developed in [4, 5]. Finally, our results can be used to study cyber physical attacks ([9]) by using tools like as Ariadne ([3]) based on Taylor theory.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The Algorithmic Analysis of Hybrid Systems. Theor. Comput. Sci. 138(1) (1995) 3–34.
2. R. Alur, T. A. Henzinger, P. H. Ho. Automatic Symbolic Verification of Embedded Systems. IEEE Trans. Software Eng. 22(6) (1996) 181–201.
3. A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, A. L. Sangiovanni-Vincentelli. Ariadne, a Framework for Reachability Analysis of Hybrid Automata. Proc. Int. Symp. on Mathematical Theory of Networks and Systems, 2006.
4. V. Castiglioni, R. Lanotte, S. Tini. A Function Elimination Method for Checking Satisfiability of Arithmetical Logics. Proc. of the 23th International Workshop CS&P 2014, CEUR Workshop Proceedings 1269, pp. 46–57 (2014).
5. V. Castiglioni, R. Lanotte, S. Tini. A Function Elimination Method for Checking Satisfiability of Arithmetical Logics. Fundamenta Informaticae 143: 51–71 (2016).
6. T. A. Henzinger, P. H. Ho, H. Wong-Toi. Algorithmic Analysis of Nonlinear Hybrid Systems. IEEE Trans. Automat. Contr. 43(4) (1998) 540-554.
7. T. A. Henzinger, P. W. Kopke, A. Puri, P. Varaiya. What's Decidable About Hybrid Automata? J. Comput. Syst. Sci. 57(1) (1998) 94–124.
8. R. Lanotte. Expressive Power of Hybrid Systems with Real Variables, Integer Variables and Arrays. J. Autom. Lang. Comb. 12 (3): 373–405 (2007).
9. R. Lanotte, M. Merro, R. Muradore, L. Viganó. A Formal Approach to Cyber-Physical Attacks. Submitted for publication.
10. R. Lanotte, S. Tini. Taylor Approximation for Hybrid Systems. Proc. Hybrid Systems: Computation and Control, LNCS 3114, Springer, Berlin, 1999, pp. 402–416.
11. R. Lanotte, S. Tini. Taylor Approximation for Hybrid Systems. Information and Computation 205(11): 1575-1607 (2007).
12. R. Lanotte, A. Maggiolo-Schettini, S. Tini. Information flow in hybrid systems. ACM Trans. Embedded Comput. Syst. 3 (4): 760–799 (2004).
13. A. Pnueli and J. Sifakis (Eds.), Special Issue on Hybrid Systems, Theor. Comput. Sci. 138(1) (1995).
14. D. Richardson. Some Undecidable Problems Involving Elementary Functions of a Real Variable. J. Symbolic Logic 33 (1968), no. 4, 514-520.
15. A. Tarski. A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley, California, 1951.

Analysing of M-AHIDS with future states on DARPA and KDD99 benchmarks

Mikuláš Pataky and Damas P. Gruska

Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovak Republic
{pataky,gruska}@fmph.uniba.sk

Abstract. Second generation of Multi-agent heterogeneous intrusion detection system (M-AHIDS) is a prototype proposed to detect untrusted and unusual network behaviour. The M-AHIDS is based on online traffic statistics in sFlow format acquired by network device with the sFlow agent and is able to perform a real-time surveillance of the 10 Gb networks. However, after an immense reimplementation it is capable to process also offline data set from DARPA Intrusion Detection Evaluation Data Set and KDD99 Cup data set. Offline data sets are used for the correct comparison with another IDSs. The main contribution of the system is the integration of several anomaly detection techniques, new future state prognostic and new machinery of multi-agent temporal logic with hybrid argumentation. Every detection technique is represented by featuring a specific detection autonomous agent. At this stage, every agent determines the flow trustfulness from aggregated connection. The anomalies are used as an input for machinery of multi-agent temporal logic which is represented by the logical agent. M-AHIDS is already partially implemented, tested and modified accordingly for more than three years.

1 Introduction

The number of users using internet and local networks is increasing every day. Consequently, there are many threats of trying to have an access to private password, to data or to injure users by other ways. Fortunately, current generation of network devices allows a real-time scraping of structured snapshots of a traffic on the networks. This information is provided by various technologies. Two the mostly used technologies are the NetFlow format introduced by CISCO and the sFlow format. These technologies allow us to observe the individual flows on the network. A flow is an unidirectional component of TCP connection (or UDP/ICMP equivalent), defined as a set of packets with identical source and destination IP addresses, ports and protocol, packed size, MAC addresses, switch ports, flags and more.

A piece of information provided by NetFlow or sFlow can be used to detect a network attack. The most frequent attacks on networks can be divided to three

main classes [1]: **Breaks privacy rules**, compromising the information confidentiality; **Alters information**, compromising the data integrity; **Denial of service attacks** (DOS or DDOS attacks), which makes a network infrastructure unavailable or unreliable, compromising the availability of the resource.

The protection of networks is, therefore, more than useful, if it is vital for long time. This issue requires monitoring of real distributed hosts, of various events and of exchanges between these hosts. Multi agent system (MAS) is very effective approach for this kind of problems as it can integrate many different techniques to one solution.

The aim of this paper is to propose the second generation of multi-agent system for network intrusion detection M-AHIDS. The first generation was presented in [2]. This generation is based on several years of experiences with developing, improving, implementing, deploying and testing of M-AHIDS. The main contribution of the second generation of M-AHIDS is the integration of several anomaly detection techniques, new future state prognostic and new machinery of multi-agent temporal logic with hybrid negotiation based on argumentation. Every detection technique is represented by featuring a specific detection autonomous agent and every agent determines the flow trustworthiness from aggregated connection. Inspiration for our agents came from project CAMNEP [3, 4]. All CAMNEP agents are more or less separate IDS and the project CAMNEP tries to connect their results to the more trustworthy results. But we have decided to use another approach in our IDS. Our agents are as simple as possible.

We are also still improving our unique [1] Web agent. The web agent is based on our past project [5–7] about de-anonymization of an Internet user. This project has been deployed on all web pages of Comenius University for more than three years. We can detect ordinary users' behaviour from its data. We used all the collected data for deep analysis and we created Web agent which is able to detect a trustworthy host based solely on his activity on the web pages.

We have used another new approach for making decisions about intrusion from agent's knowledge base detection. For this purpose we have used specifically developed multi-agent temporal logic (M-ATL). The anomalies are used as an input for machinery of M-ATL and the new version of hybrid argumentation which are represented by a logical agent. The logical agent is one of the system advantages because it has huge capabilities for making the right decision about the intrusions from detected anomalies. All detected intrusions are the past states in M-ATL and we are using newly implemented prediction methods base on regression models of time series for the future states. The regression models are used for computation of the future states from the collection of the past and the actual connections.

The most important contributions of our research presented in this paper are THE FOLLOWING: Improving the integration of the several anomaly detection techniques in a form of an agent; Extension of machinery of the multi-agent temporal logic and hybrid negotiation about the future state; Major update of argumentation framework; Presenting new testing approach based on offline

[1] with our best knowledge

DARPA Intrusion Detection Evaluation Data Set and KDD99 Cup data set. M-AHIDS is partially implemented and tested on local network of Department of Applied Informatics. Results obtained on KDD99 are comparable to another IDS.

The organization of the paper is as follows: in **Section 2** – overview of the IDS and selected existing solutions and approaches; in **Section 3** – proposal of detection system architecture; in **Section 4** – detailed description of all agents in M-AHIDS; in **Section 5** – overview of case study, tests and results.

2 Intrusion detection systems

Intrusion Detection System or IDS is a software, hardware or combination of both used to detect an intruder's activity. The base characteristics of IDS [8] are neutralizing illegal intrusion attempts in the real time. Consequently, it must be executed constantly in a host or in a network.

There are many types of IDS and each of them has some advantages and disadvantages. Their strengths and weaknesses depend mostly on the way they recognize the threats. Two main approaches for detection intrusion are [1]:

Behaviour-based intrusion detection approach discovers intrusive activity by comparing user's or system's behaviour profile with normal behaviour profile;

Knowledge-based (signature-based) intrusion detection approach detects intrusions upon a comparison between the parameters of users' session and the known pattern attacks stored in a database.

In recent years, several new approaches in IDS systems have been published. Certain approaches have been identified as relevant for our project. The first, multi-agent distributed IDS(DIDS) model based on the **BP neural network** adopts the modes of distributed detection and distributed response [9]. The second, **emulation-based** network intrusion detection systems have been devised to detect the presence of shellcode in the network traffic by trying to execute (portions of) the network packet payloads in an instrumented environment and checking the execution traces for signs of shellcode activity [10]. The fourth, **multi-stage approach** to constructing hierarchical classifiers that combines process mining, feature extraction based on temporal patterns and constructing classifiers based on a decision tree [11]. The fifth, **content anomaly detection** (CAD) models the payloads of traffic instead of the higher level attributes. Zero-day attacks then appear as outliers to the properly trained CAD sensors [12]. The sixth approach is to detect TCP connection based attacks using certain **data mining algorithms**[13]. J-48 decision tree algorithm and Nave Bayes classifiers were learnt on 19 selected features from KDD 99 dataset. The selected feature had been chosen by Markov blanket and Pearson correlation. The approach could detect about 74% of novel attacks with 19 features.

3 M-AHIDS

The following section briefly proposes the foundations for the second generation network intrusion detection multi-agent system M-AHIDS. Design of the system arose from theoretical research as well as from practical experiences which have been already obtained by testing for more than three years.

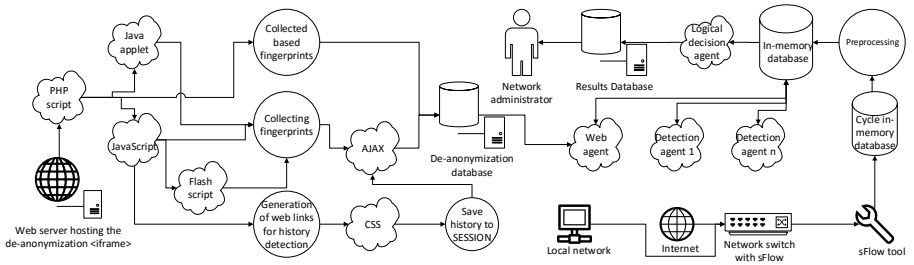


Fig. 1. Architecture of IDS

3.1 System layers

M-AHIDS network intrusion detection system consists of four layers.

The first layer contains the 10Gb network switch with the sFlow agent. This switch can be replaced by another network device with the sFlow agent. The sFlow agent sends sFlow datagram to M-AHIDSwhich functions also the sFlow collector.

The second layer contains sFlowTool and the pre-processing agent. sFlowTool receives the sFlow UDP datagrams. M-AHIDS reads the encoded result from sFlowTool and the important data are saved to the in-memory database. Here we use this information from sFlow: 'srcIP', 'dstIP', 'srcMAC', 'dstMAC', 'srcPort', 'dstPort', 'IPProtocol', 'sampledPacketSize', 'UDPBytes', 'TCPFlags', 'inPort', 'outPort' and 'time'.

The third layer contains upgraded detection agents. Every agent is implemented as an independent thread. The number of the actually active agents depends on the number of the computer processor cores.

The forth layer contains the new version of the logical agent, database with results and the front-end for network administrator which can be used to correct the results.

3.2 sFlow

sFlow is a multi-vendor sampling technology embedded within network switches and routers. It provides the ability to continuously monitor application level traffic flows at wire speed on all interfaces simultaneously. sFlow monitoring of high-speed, routed and switched networks has the following properties [14]: **Accurate, Detailed, Scalable, Low Cost** and **Timely**.

M-AHIDS saves approximately 10 minutes window of received sFlow datagrams in SQLlite in-memory database. In-memory database enables to analyse large amounts of received data very quickly. All detection agents work with this database and the database is also an input to the logical agent.

3.3 Implementation details

Diagram of the second generation M-AHIDS is shown in figure 1. M-AHIDS is based on Microsoft .Net 4.5 framework and multi-vendor sampling technology sFlow. It originally runs on Microsoft Server 2012. However, it can also be run on Linux based operating systems using mono platform. M-AHIDS is implemented as multi-thread application which uses sFlow for receiving sFlow UDP datagrams.

4 Agents

As written in [2], our agents were inspired by the project CAMNEP [3, 4]. However, there are several main differences: We have built the agents differently, we have added new type of agent - the Web agent, we have used the hybrid negotiation with argumentation and immune cell inspiration, prediction of future states and we have created a logical agent to complete the final decisions.

4.1 The pre-processing agent

The first step after IDS receives the sFlow datagram is pre-processing, as can be seen on figure 1. For the coverage of this function, a pre-processing agent is implemented. M-AHIDS is designed for a very high network traffic on 10Gb network switch. For this reason, agent needs to make quick decisions which connections are important (connection has probability of being an intrusion). Similarly to the other mentioned IDS we implemented this with several rules. The rules define which source, destination, port and protocol or their combinations are problem-free and they are not interesting for the detection agents. The administrator of the network can define and edit these rules.

4.2 Detection agents DA

Six types of innovated intrusion detection agents have been tested. Two of these agents have the arguments suitable for specification. Using this, we get **15** intruder detection agents. Every detection agent evaluates every connection from the pre-processing agent. The output of this evaluation is an integer. Higher number indicates behaviour that is more unusual.

The count agent is the first scalable type of agent which is counting number of connections with the same property ('dscIP', 'srcIP', 'dscPort', 'srcPort'). Higher number of connections with particular property means that connections are more suspicious. The exact mathematical formula is:

$$R_{CO}(V) = \{r_v | r_v = |C_v| : \forall v \in V\} \tag{1}$$

where R_{CO} is the set of results of the count agent, r_v is the result for all connections with particular property [2] $v \in V$. C_v is the set of the connections with property v and V is the set of all properties. M-AHIDS has a separate agent for every connection's property which is running in its own thread.

The average agent is the second scalable type of agent and it computes average number of connections with the same property ('dscIP', 'srcIP', 'dscPort', 'srcPort'). Higher difference between the number of connections and the average number of connections with particular property means more suspicious connections. The exact mathematical formula is:

$$R_{AVG}(V) = \{r_v | r_v = ||C_v| - \texttt{Avg}(R_{CO}(V))| : \forall v \in V\} \tag{2}$$

where R_{AVG} is the set of the results of the average agent.

The volume agent counts the number of connections which have the same value in linked properties. Specifically, agent links srcIP to dstIP, dstIP to srcIP,

[2] e.g. dscPort 45

srcIP to dstPort and dstIP to srcPort. All of these links are provided by separate agents, which are running in parallel. The exact mathematical formula is:

$$R_{VOL}(V) = \{r_v | r_v = |C_v| : \forall v \in V\} \tag{3}$$

where V is the set of ordered pairs $\{(v_{1,1}, v_{1,2}), (v_{2,1}, v_{2,2}), \ldots, (v_{n,1}, v_{n,2})\}$ and C_v is the set of all connections which have property $v_1 \in v \in V$ and they are linked with connections with property $v_2 \in v \in V$.

The cluster agent is the most computationally complex agent. This agent computes normalization distance between each of the connections. Agent uses dscIP, srcIP, dscPort, srcPort, dstMac and srcMac for distance computations.

$$R_{CLU} = \left\{ r | r = \frac{\sum_{c' \in C} |c, c'|}{n} : \forall c \in C \right\} \tag{4}$$

where C is the set of connections, $|c, c'|$ is the distance between two connections from C, $c, c' \in C$ and n is the capacity of set C.

The Web agent is one of our contributions in this area of research. The web agent uses the database from de-anonymization system shown on left side of the fig. 1. It compares the IP from the sFlow database with IP address of all the visitors of all web pages. If the IP address is in both databases, agent calculates if there is a higher probability of a system or a real user behind a connection and then agent determines the intrusion score for the connection using the analysis of the visited pages. If the web pages are systematically visited page by page, then there is a high probability that the visitor is a system. If the same page is visited more than once in short time, then there is a high probability that the visitor is a real user. The database of the university serving as web page visitors database was created using Internet users anonymity research [5–7].

Entropy agent captures the degree of diffusion or gathering of distribution of the connection properties. This detection method is based on equation:

$$H(X) = -\sum_{i=1}^{N} (\tfrac{n_i}{S}) log_2 (\tfrac{n_i}{S})$$

where $S = \sum_{i=1}^{N} n_i$ and X is the set of connection properties $X = \{n_1, ..., n_N\}$.

4.3 The logical agent (LA)

The logical agent makes the final decision about every connection and if this agent evaluates that this connection is an intrusion, then the agent inserts this connection to the permanent database and can be used to alert server administrator. The LA is based on Multi-Agent Temporal Logic M-ATL which was presented in [2] with argumentation upgrades described below. The M-ATL and also the argumentation was developed specifically for the needs of M-AHIDS.

The new (upgraded) version of LA also contains computation of the future states. The past states in M-ATL come from previous results which are saved in the permanent database. The future states are computed based on regression models of time series [15]. This approach was chosen as it is one of the fastest prediction technique. Low computational complexity is very important for real time IDS. The inputs are the counts of the same connection during each 10 seconds from 10 minutes time window. That means that we have 60 counts for each connection which makes the time series. From the time series six future

states are computed for the following one minute. The trend part of the time series is chosen based on MAPE [16] rating from linear trend $T = a_0 + a_1 t$, parabolic trend $T = a_0 + a_1 t + a_2 t^2$ or exponential trend $T = a_0 a_1^t$; where T is trend function, a_i is parameter of the function and t is time.

The LA has three important tasks. The first one is to build a knowledge base from the results of the DA. At this stage, LA normalizes the results to interval $\langle 0, 1\rangle$. The normalization uses the network administrator's corrections and the immune inspiration for updating the DA's trust weights. The trust weights are also real numbers from interval $\langle 0, 1\rangle$. Higher number means more trust for the agent. LA converts the results of DA to boolean value. This conversion is based on agents' trust and the mathematical formula is:

$$C_A = \{c_{Ai} | \exists r_{Ai} \in R_A : r_{Ai} > (1 - W_A)\} \tag{5}$$

where $C_A = \{c_{A1}, c_{A2}, \ldots, c_{An}\}$ is the set of intrusions detected by DA A, $R_A = \{r_{A1}, r_{A2}, \ldots, r_{An}\}$ is the set of normalized results from agent DA A and W_A is the trust weight of the agent A.

After normalization, LA uses the new argumentation framework to negotiate the final decision – which connections are intrusions. We describe our argumentation framework below. The last task for LA is to save the results to the permanent database.

The argumentation framework (FA) is one of the approaches for negotiation amongst agents. The implemented FA can evaluate all used logical clauses but is not complete as the intrusion detection is computationally hard and M-AHIDS must work in parallel with network operation.

However, as the logic machinery of our M-ATL runs after all our DA agents in M-AHIDS have finished evaluation of all connections, we do not have to think about incomplete knowledge in our argumentation framework. This fact simplifies the proposal of argumentation framework.

The new version of argumentation framework is based on work of Dung [17]

An argumentation framework AF is ordered pair $AF = \langle AR, attacks\rangle$ where AR is set of arguments and $attacks$ is binary relation based on AR: $attacks \subseteq AR \times AR$

A conflict-free set of arguments S is if there are no arguments $A, B \in S$ such that $(A, B) \in attacks$

An acceptable argument $A \in AR$ with respect to a set S iff for each argument $B \in AR$: if B $attacks$ A then B is $attacked$ by S.

An admissible set of arguments is a conflict-free set of arguments S iff each argument in S is acceptable with respect to S.

A preferred extension of an argumentation framework AF is a maximal (with respect to set inclusion) admissible set of AF, which defines the (credulous) semantics of an argumentation framework.

Important provable conclusion [17] is, that every argumentation framework possesses at least one preferred extension.

A stable extension is a conflict-free set of arguments$S$ iff S $attacks$ each argument which does not belong to S.

S is a stable extension iff $S = \{A | A$ is not attacked by $S\}$

Another important conclusion that every **stable extension** is also preferred extension, but not vice versa, is proved in [17]. This determination of the argumentation framework is sufficient for our proposes.

The base of **our new version of argumentation** is also the binary relation of preferences (*attaks*) $\longmapsto$. $\varphi \longmapsto \varphi'$ means that φ is stronger than φ'. The logical formulas φ and φ' belong to $\longmapsto$ iff both contain the same atomic formula p with an opposite value. That means that the two DAs have contradict results about trust of the same connection. For building relation of preferences we use rules:

$$X_I\varphi : w_I \longmapsto X_J\varphi : w_J \text{ iff } \sum_{i\in I} w_i > \sum_{j\in J} w_j, \tag{6}$$

$$p_I : w_I \longmapsto p_J : w_J \text{ iff } \sum_{i\in I} w_i > \sum_{j\in J} w_j \tag{7}$$

$$X_i p_i \longmapsto p_j \tag{8}$$

$$H_i\varphi \longmapsto P_j\varphi \tag{9}$$

$$G_i\varphi \longmapsto F_j\varphi \tag{10}$$

$$X_A\varphi \text{ iff } \varphi \tag{11}$$

where $X \in \{F, G, P, H\}$, p_i is the evaluation of connection by agent a_i, I, J are same sets of labelling of agents, $i \in I$, $j \in J$ and w_i is the weight of agents' a_i trustfulness. The connectors F (some future state), G (all future states), P (some past state), H (all past states) and logical formula φ are defined in our previous paper [2]. Rules 6 - 11 should be interpreted as: 6 and 7 - the agents with higher collective trust beat the agents with lower trust; 8 - complex knowledge beats simple knowledge; 9 - all past states beat one past state; 10 - all future states beat one future state; 11 - formula is true iff all agents have the same evaluation.

The LA computes preferred extensions of AF and that is a solution for the problem with evaluation of the connection represented by one atomic variable p. If this extension is also stable extension and it contains arguments which claim that the connection is part of the intrusion, LA will write this connection to the permanent database of results.

5 Results

We have implemented M-AHIDS bottom up using several iterations, because the most important requirement on IDS is the real time detection. After each iteration performance test and optimization were performed.

Table 1. False negative (FN) rate of DA and LA

		Agents							
Attack	**#**	**Count**	**Average**	**Volume**	**Cluster**	**Web**	**Entropy**	***Logical***	**FP**
DOS	100	96	98	99	99	95	97	99	1,00%
DDOS	100	94	95	60	97	99	99	96	4,00%
Port Scans	100	96	97	95	96	98	95	98	2,00%
BitTorrents	100	70	73	98	95	23	96	97	3,00%
Malwares	100	62	59	99	97	56	94	97	3,00%
ALL	500	418	422	451	484	371	481	487	2,60%
FN		**16,40%**	**15,60%**	**9,80%**	**3,20%**	**25,80%**	**3,80%**	**2,60%**	

Table 2. False positive (FP) rate of DA and LA

		Agents							
Attack	**#**	**Count**	**Average**	**Volume**	**Cluster**	**Web**	**Entropy**	*Logical*	**FP**
DOS	100	177	183	80	125	137	145	127	27,00%
DDOS	100	165	170	58	128	165	150	129	29,00%
Port Scans	100	139	144	122	123	135	128	132	32,00%
BitTorrents	100	69	75	146	124	33	134	143	43,00%
Malwares	100	70	59	161	138	68	126	157	57,00%
ALL	500	620	631	567	638	538	683	688	**37,60%**
FP		24,00%	26,20%	13,40%	27,60%	7,60%	36,60%	**37,60%**	

M-AHIDS is now running on server based on Intel i7-4770S, 2x8GB 1600MHz DDR3 CL10 DIMM RAM, 1TB HDD and OS Windows 2012 server. The sFlow agent is running on switch Zyxel GS1910-24.

During the tests, the system was supervised and it learnt the usual network behaviour. After three days of learning we tested system for attacks like DoS, DDoS, Port Scanning, BitTorrent (usually unwanted in commercial networks) and Malware attacks.

The Table 2 shows a false positive rate of the agents and the Table 1 shows a false negative rate of the agents.M-AHIDS was tested during usual week network operation. Every attack was sent 100 times and with these attacks we sent the same number of connections with similar properties as the sent attacks.

This test scenario was repeated two times. Once with the simpler LA with smaller AF and afterwards with the new AL with more complex AF. The new LA had better FN about $0,4$ percentage points which is $13,33$ **percentage progress** and it had the worst FP about $1,2$ percentage points which is only $3,3$ percentage retrogression.

5.1 Benchmark KDD99 Cup data set

Furthermore the second generation M-AHIDS was adapted for KDD99 Cup data set. KDD99 Cup data set was adopted for this study because it is widely used intrusion detection data set. The paper [18] compared 125 intrusion detection systems using KDD99 Cup data set between 2010 and 2015. This indicates that although KDD99 dataset is more than 15 years old, it is still widely used in the academic research. By this way the comparison between M-AHIDS and other similar studies is achieved. KDD99 Cup data set is created by extracting some features (IP number, port number, initial date) from DARPA 98 and it has about 4 900 000 data vectors. This data set is prepared by Stolfo et al. [19] and is built based on the data captured in DARPA98 IDS evaluation program [20]. KDD99 Cup data set includes 80% of attack and 20% of normal data. The package with the cup data set also contains training data set (labelled) and testing data set (without label). Each connection vector has 41 features and is labelled either normal or attack.

However M-AHIDS was not compatible with KDD99 Cup data set, because it was designed for recognizing intrusion directly from sFLOW. So it had to be adapted for comparison. The adaptation processes was done in two major steps:

1. The first step was about the changing of M-AHIDS to process offline data. The raw data from DARPA Intrusion Detection Evaluation Data Set [21] was used. This was similar to data set which had been provided by sFlow.
2. The second step was the modification of M-AHIDS to process extracted features provided by KDD99 Cup data set. Difference between KDD99 data set and data set provided by sFLOW is significant. For that reason all agents had to be updated.

The detailed description about those adaptation processes are out of range of this paper. Only results achieved after each step are presented.

The biggest disadvantage of using the DARPA Intrusion Detection Evaluation Data Set and KDD99 Cup data set is that the important Web Agent feature of M-AHIDS cannot be used because there are no data for it. Despite this fact we compared our approach with both data sets.

In addition, the measures were used to evaluate the performance of MAS-IDS: accuracy, detection rate, false alarm rate:

$$DedectionRate = \frac{NumberOfDetectedAttacks}{NumberOfAttacks} \cdot 100\% \tag{12}$$

$$FalsePositive = \frac{MisclassifiedConnections}{NumberOfNormalConnection} \cdot 100\% \tag{13}$$

$$Accuracy = \frac{CorrectClassifiedConnections}{NumberOfConnections} \cdot 100\% \tag{14}$$

The detailed results can be seen in Tables 3 and 4

Table 3. Darpa success rates

	Agents					
Measures	**Count**	**Average**	**Volume**	**Cluster**	**Entropy**	***Logical***
Detection Rate	78,4	81,4	79,4	85,3	83,9	**93,1**
False Positive	10,1	9,8	10	9,2	9,3	**8,8**
Accurancy	78,2	80,9	79,1	84,8	83,1	**92,3**

Table 4. KDD99 success rates

	Agents					
Measures	**Count**	**Average**	**Volume**	**Cluster**	**Entropy**	***Logical***
Detection Rate	77,9	81,1	78,8	84,9	82,4	**91,9**
False Positive	10,6	10,1	10,1	9,5	9,7	**9,1**
Accurancy	77,7	80,5	78,2	84,1	82,6	**91,5**

6 Conclusion

In this paper we presented the proposal for the second generation of the system for detection intrusions in a network. The most important system features of the developed and partially implemented M-AHIDS are integration of several innovated anomaly detection techniques in a form of agent, machinery of a multi-agent temporal logic, hybrid negotiation with new version of argumentation and immune cell inspiration, newly implemented computation of future

states and last but not least the new innovative Web agent which is able to detect trustworthy host from his activity on web pages. This agent is based on our previous research which is deployed on all web pages of Comenius University for three years.

When the system passed $2,6\%$ false negative in the normal connections, the system achieved $37,6\%$ false positive in the malicious connections. That is a satisfactory result as project CAMNEP [4] achieved with 1% false negative in the normal connections only 40% false positive in the malicious connections.

Satisfactory results were achieved on DARPA Intrusion Detection Evaluation Data Set (Detection Rate = 93,1; False Positive = 8,8; Accuracy = 92,3) and KDD99 Cup data set (Detection Rate = 91,9; False Positive = 9,1; Accuracy = 91,5) which are comparable witch other IDS systems. However we have to consider that one of our major feature (Web agent) can not be used due to the lack of data. There is a reasonable belief that the results in the online testing with Web agent will yield better results.

M-AHIDS is still in the development phase, but parts of the system are deployed for more than three years on the department network. Here, we have implemented the most of the presented features of M-AHIDS.

As the next step we would like to implement the rest of the features to M-AHIDS, to optimize the already implemented features and to provide more and longer lasting tests. Here we also consider more sophisticated approach for data clustering as in [22].

References

1. Boudaoud, K., Labiod, H., Guessoum, Z., Boutaba, R.: Network security management with intelligent agents. In: NOMS 2000, IEEE/IFIP Network Operations and Management Symposium, 08-14 April 2000, Honolulu, Hawaii, Honolulu, UNITED STATES (04 2000)
2. Pataky, M., Gruska, D.P.: Multi-agent heterogeneous intrusion detection system. In: Proceedings of the 23th International Workshop on Concurrency, Specification and Programming, Chemnitz, Germany, September 29 - October 1, 2014. (2014) 184–195
3. Rehak, M., Pechoucek, M., Bartos, K., Grill, M., Celeda, P., Krmicek, V.: Camnep: An intrusion detection system for high-speed networks. Progress in Informatics **5**(5) (March 2008) 65–74
4. Rehák, M., Pechoucek, M., Grill, M., Stiborek, J., Bartoš, K., Celeda, P.: Adaptive multiagent system for network traffic monitoring. IEEE Intelligent Systems **24**(3) (2009) 16–25
5. Pataky, M.: The anonymity of the internet user. In: Proceedings of the Scientific Conference of Technology and Innovation Processes 2013, Hradec Králové, CZ, MAGNANIMITAS (2013) 35–41
6. Pataky, M.: Anonymita používateĺa v internete. In: ITAT 2013: Information Technologies - Applications and Theory Proceedings, CreateSpace Independent Publishing Platform (2013) 18–23
7. Pataky, M.: De-anonymization of an internet user based on his web browser. In: CER Comparative European Research 2014 Proceedings, London, Sciemcee Publishing (2014) 125–128

8. Benyettou, N., Benyettou, A., Rodin, V., Berrouiguet, S.Y.: The multi-agents immune system for network intrusions detection (MAISID). Oriental Journal Of Computer Science & Technology **6**(4) (December 2013) 383–390
9. Zhai, S., Hu, C., Weiming, Z.: Multiagent distributed intrusion detection system model based on bp neural network. International Journal of Information and Network Security (IJINS) **3**(3) (2014)
10. Abbasi, A., Wetzels, J., Bokslag, W., Zambon, E., Etalle, S.: On emulation-based network intrusion detection systems. In Stavrou, A., Bos, H., Portokalidis, G., eds.: Research in Attacks, Intrusions and Defenses. Volume 8688 of Lecture Notes in Computer Science. Springer International Publishing (2014) 384–404
11. Bazan, J.G., Szpyrka, M., Szczur, A., Dydo, L., Wojtowicz, H.: Classifiers for behavioral patterns identification induced from huge temporal data. In: Proceedings of the 23th International Workshop on Concurrency, Specification and Programming, Chemnitz, Germany, September 29 - October 1, 2014. (2014) 22–33
12. Whalen, S., Boggs, N., Stolfo, S.J.: Model aggregation for distributed content anomaly detection. In: Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop. AISec '14, New York, NY, USA, ACM (2014) 61–71
13. Ugtakhbayar, N., Usukhbayar, B., Nyamjav, J.: An approach to detect tcp/ip based attack. International Journal of Computer Science and Network Security **16**(4) (April 2016) 37–40
14. sFlow.org: Traffic monitoring using sflow (2003)
15. OSTERTAGOV, E.: Modelovanie asovch radov. The 13th International Scientific Conference: Trends and Innovative Approaches in Business Processes 2010 (2010)
16. Tofallis, C.: A better measure of relative prediction accuracy for model selection and model estimation. JORS **66**(8) (2015) 1352–1362
17. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2) (September 1995) 321–357
18. Özgür, A., Erdem, H.: A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. PeerJ PrePrints **4** (2016) e1954
19. Stolfo, S.J., Fan, W., Lee, W., Prodromidis, A., , Chan, P.K.: Cost-based modeling for fraud and intrusion detection: Results from the jam project. discex **2** (2000) 1130
20. Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., Zissman, M.A.: Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. discex **2** (2000) 1012
21. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 darpa off-line intrusion detection evaluation. Comput. Netw. **34**(4) (October 2000) 579–595
22. Lasek, P., Lasek, K.: Relative constraints as features. In Popova-Zeugmann, L., ed.: Proceedings of the 23th International Workshop on Concurrency, Specification and Programming, Chemnitz, Germany, September 29 - October 1, 2014. Volume 1269 of CEUR Workshop Proceedings., CEUR-WS.org (2014) 121–125

A graph-based reduction in PlanICS abstract planning, based on partial orders of services (Extended Abstract)

Maciej Szreter

Institute of Computer Science, Polish Academy of Sciences

Abstract. The paper deals with the abstract planning problem – a stage of the Web Service Composition in the PlanICS framework. The planning task is viewed from the graph perspective, searching a graph built of vertices representing service and object types, and edges connecting service types to object types processed by each corresponding service. The preprocessing search identifies all the service and object types, which can potentially participate in a plan bounded by the given length. Then, the planning problem is split into subproblems, at the basis of the relation of independence between the sets of object types listed in user query as the desired result of the composition. The ontology is divided into disjoint sub-ontologies, each of which contains only the types relevant for the respective set. If there is a sub-plan for every sub-ontology, the resulting plan is composed out of these sub-plans.
The presented approach uses similarily defined graphs, and makes use of planners as external tools, as in [Szr15] describing graph-based pruning of ontologies, however the aim is different. In [Szr15], there are removed all the service and object types which, for the given user query, cannot occur in any plan bound by a fixed length. The current work optimizes planning in what is left by this reduction.

1 Introduction

The key concept of Service-Oriented Architecture (SOA) [Erl05] consists in using independent (software) components available via well-defined interfaces. Frequently, there is no web service directly satisfying the user objective, but a composition of services can deliver the requested result. An automatic composition of Web services should relieve the user of a manual preparation of detailed execution plans, matching services to each other, and of choosing optimal providers for all the components. The problem of finding such a composition is hard and well known as the Web Service Composition Problem (WSCP) [Erl05]. There is a number of various approaches to solve WSCP [LOKX10].

Automated composition of web services Web services are widely used to implement SOA paradigm, but much of their benefits is revealed when they can be composed automatically. The existing solutions to WSCP are divided into

several groups. Following [LOKX10] our approach belongs to AI planning methods, including also approaches based on: automata theory, situation calculus, Petri nets, automated theorem proving, and model checking.

In the paper [SPAS03] Sycara et.al. present DAML-S, the predecessor of OWL-S, which is one of the standards to describe Semantic Web services. The authors show how to compose services using the DAML-S virtual machine and matchmaking mechanism with a support of a planning-capable agent. Another approach to WSC using OWL-S is described in [KG05] where Klusch et.al. introduce the OWLS-Xplan framework.

Graph-based approaches to planning and web services composition Several papers use graph-based approach to the WSCP. An example is [HM06], where information about inputs and outputs of services is represented by interface automata. The key difference between these papers and our approach is that we model matching of the service input and output at the graph level, by reducing it to the problem of testing reachability in graphs. Graph vertices model not only services, but also objects read and produced by services. Due to introducing the abstract planning we can reduce matching of service and object types to the problem of existing paths between selected graph vertices. The other approaches use graphs as a model for representing the state space, but test matching objects to services by calling specialized algorithms. This can hamper the performance. Another original feature is using graph databases for representing graphs and doing operations on them. To the best of our knowledge, we could also find no experimental analysis showing that graph-based approaches can effectively deal with ontologies with significant numbers of service and object types.

Graph databases [RWE13] are a relatively recent addition in the domain of the NoSQL databases, defined by rejecting the traditional database model of relational tables, and using alternative solutions, in this case graphs. Neo4j [Lal15] is one of the most popular implementations.

2 Solution

For the efficiency reason, the planning process in PlanICSis divided into two phases: abstract and concrete planning. The former deals with matching types, and the latter with matching the exact values of attributes. The current paper deals only with the former one. Here, we briefly recall the formulation of the abstract planning problem (APP) in PlanICS. We present only the intuitions, but all the formal definitions can be found in [D$^+$11].

PlanICS has much in common with well known approaches to composition of web services in the Semantic Web environment. One of similarities between PlanICS and DAML-S/OWL-S is the description of service capabilities. That is, both the approaches use the implicit capability representation, i.e., a service is described by the state transformation, its input, and output. This paradigm is often called IOPE - Input, Output, Precondition, Effect. Services process objects: read those placed in their input list, and write or produce those in output lists.

In addition, objects from inout lists can be both read and written. We refer to all these lists as *input and output lists*. Objects have as attributes either primitive types such as integers, booleans, and strings, or other objects. Pre- and post-conditions formulae, assigned to service types, determine the state of (some of) these attributes before and after executing the service. All the service and object types are organized in an inheritance hierarchy and represented in an *ontology*. A *user query* represents the aim of the composition, with similar input and output lists, and pre-/postcondition formulae as for services, determining the initial and the expected state of the composition process. The precisely defined semantics defines which sequences of services are the solution of the user query.

Semantics of abstract planning In abstract planning, the aim is to find the abstract plan(s). A *world* is a set of objects with valuations assigned to their attributes. An *transformation* is an application of a service to an *input* world, producing an *output* world such that the valuations of objects from both worlds occuring in the lists for the service are consistent with the pre- and postcondition formulae. A *solution* is a sequence of service types, with a *context function* assigning instances of objects to the input and output list of every service. A solution explains which transformations are needed in order to deliver all the object types requested in the user query, with the appropriate attributes set or not. Note that the objects needed by services can be provided in the input lists of the user query, or by services producing objects while taking no input. Two solutions are *equivalent* if they contain the same number of occurences for every service type. An *abstract plan* is an equivalence class with respect to this relation, so that the irrelevant orderings of services are abstracted away.

Currently there are three PlanICS solvers, based on SMT [NP13], Genetic Algorithms (GA) [SNP13] and the hybrid (H) [NPS15] being the combination of SMT and GA. For testing the performance, an ontology generator has been implemented, with several parameters characterizing the randomly generated ontologies. These parameters include the minimal and maximal numbers of services, objects, object attributes, objects in service lists, and objects in the user query. The structure of the plans is determined by the number of partial orders out of which which every object from the final world can be constructed.

The performance of the planners exhibits some general characteristics. SMT planner can handle smaller ontologies than GA, but finds all the plans and can determine that no plan exists at all. The hybrid planner combines the features of both SMT and GA planner. A common feature of all the planners is that their performance degrades when the length of the solutions increases. Also, all the plans found for the given depth need to be of the equal length.

Graph-based reduction of ontology In [Szr15] we have shown a graph-based pruning approach removing from an ontology all the service and object types which are not relevant for any plan of the given length, as determined by the user query. The search is implemented using a graph database. This improved significantly the performance of all the tested planners, and it corresponds to a

typical scenario where only a small part of the ontology is relevant for the user query. Now we briefly describe this reduction.

First, the *ontology graph* is constructed for the given ontology, so that every service type and every object type are uniquely represented by a graph vertex. For every vertex representing a service type, there are incoming edges from vertices for the object types present in the input lists of the service, and there are outgoing edges to vertices for the object types in the output lists.

For the given user query, the additional two vertices are added to the ontology graph: the start and final vertex. The start vertex has outgoing edges to every object type from the input lists of the user query. The final vertex has incoming edges from every object type from the output lists of the user query. Then, we search for all the paths leading from start to final vertex, yielding *query k-subgraph* G_{qs}. We proved that only the service and object types present in G_{qs}, with their supertypes and the subtypes of the object types from the query, can occur in any plan of the length bound by k, and all the other types can be pruned.

Some pruned ontologies are still hard for the planners. We diagnosed that the problem is caused by the length of the plans rather than by their number. Usually, even restricting the ontology with more plans only to the types occuring in a single plan does not improve the situation.

Paper contribution: finding sub-ontologies in the reduced ontology Now we will focus on more efficient planning for the ontologies pruned by the graph reduction. In particular, we identify the disjoint sub-ontologies which can be checked independently, so that the resulting plans can be composed by taking a sub-plan found for every sub-ontology. The sub-ontologies are identified by using the graph approach based on analysis of G_{qs}. In particular, for every object type occurring in the output lists of the user query, we define a *Object Type Derivation Graph* (OTDG) which is G_{qs} restricted to the paths going via the vertex representing this type.

Then, we introduce the independence relation between the subsets of object types occurring in the output lists of the user query. Two such subsets are independent iff for every pair of object types from both these sets, their OTDGs are disjoint (have no common vertices). For every subset, the *sub-ontology* contains all the service and object types present in the OTDGs for the object types from the set, and the predecessors of these types. The user query is restricted to sub-queries accordingly.

The correctness of the reduction is shown in the following way. We claim that the set of plans generated by our construction at the basis of sub-plans found for every sub-ontology is equal to the set of plans for the k-reduced ontology. To show this, first we prove that every plan composed of sub-plans for sub-ontologies is a plan in the k-reduced ontology. Then, we show that for every plan from the k-reduced ontology, there exists a plan composed from the elements of union of a sub-plan for every sub-ontology such that these two plans are equivalent.

Note that if there is no sub-plan for a sub-ontology, then no plan exists for the complete ontology.

Experiments We briefly describe experiments in which we compared the performance of two planners (SMT and GA) for three approaches: the full ontology (FO), the k-reduced ontology (RO) and the sub-ontologies (SO). The examine two groups of benchmarks produced by a scalable generator accepting several parameters. There are five objects of distinct object types to be produced ($n_{|EW|} = 5$). 102 object types and 64 service types are present in every ontology. In the group A the length for every subplan is $len = k/n_{|EW|}$, compared to k being the length of every plan for FO and RO. There are 4 plans, because two object types can be produced in two ways, and other object types in a single way. In the group B, we scale the number of ways in which some two object types can be produced. The length of every plan is thus constant but the number of plans changes.

We implemented the described algorithm generating the k-query subgraphs for every query, and determining the sub-ontologies corresponding to the subgraphs determined by our independence relation. Its running time is very short (well below 1 second) for all the benchmarks presented in the paper.

A	$len = 2 + id_1, n_{\|EW\|} = 5, k = 5 * len, \|p\| = 4$											
id_1	SMT_{FO}		SMT_{RO}		SMT_{SO}		GA_{FO}		GA_{RO}		GA_{SO}	
	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t
1	4	7.7	4	3.2	4	0.4	4	3.9	4	2.7	4	1.3
2	4	88.5	4	34.8	4	0.7	4	6.4	3	4.4	4	1.3
3	4	1025	4	560	4	1.1	2	12.1	3	6.9	4	1.4
4	4	TO	4	TO	4	2.3	0	20.6	1	11.9	4	1.5
5	4	TO	4	TO	4	2.9	0	32.7	0	19.9	4	1.6
6	3	TO	3	TO	4	3.2	0	50.7	0	30.2	4	1.8
7	0	TO	1	TO	4	3.9	0	73.9	0	49.3	4	2.1

B	$len = 2, k = 10, n_{\|EW\|} = 5, \|p\| = 2^{id_2}$											
id_2	SMT_{FO}		SMT_{RO}		SMT_{SO}		GA_{FO}		GA_{RO}		GA_{SO}	
	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t
1	2	1326.5	2	549.9	2	1.7	2	7.0	2	4.8	2	1.6
2	4	785.8	4	384.0	4	1.7	4	7.1	4	4.7	4	1.8
3	8	1610.8	8	861.5	8	1.8	5	6.6	6	5.2	8	1.9
4	16	TO	16	TO	16	2.0	6	6.7	7	7.5	16	2.1
5	32	TO	32	TO	32	2.1	6	10.4	6	5.3	32	2.3

Table 1: Experimental results for two sets of parameters. $|p|$ is the number of plans, t time in seconds. TO denotes times longer than 2000 seconds. For GA, 10 instances have been run, and we report the maximal number of plans found.

The conclusion is that the sub-plans are found very quickly, because they are much shorter than the complete plan.

3 Conclusion

As the experiments testify, the presented algorithm can shorten the planning times by several orders of magnitude, when it is applicable. Conceptually, it is different from the well-known partial-order reductions used in formal verification. The key difference is that the latter approaches deal with global states, possibly pruning some executions not relevant for preserving the requested properties. Here we identify independent sub-systems at the basis of their local behavior.

The future research will consist in proposing weaker independence conditions, relaxing the requirement for disjoint OTDGs. Another research direction

is to encode OTDGs directly into a planner. This would enable finding plans of different lengths for a fixed depth bound. An ultimate aim is a fully graph-based planner directly exploating the structure of the problem.

References

[D+11] Dariusz Doliwa et al. PlanICS - a web service compositon toolset. *Fundam. Inform.*, 112(1):47–71, 2011.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[HM06] Seyyed Vahid Hashemian and Farhad Mavaddat. A graph-based framework for composition of stateless web services. In *ECOWS*, pages 75–86. IEEE Computer Society, 2006.

[KG05] Matthias Klusch and Andreas Gerber. Semantic web service composition planning with owls-xplan. In *Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[Lal15] Mahesh Lal. *Neo4J Graph Data Modeling*. Packt Publishing, 2015.

[LOKX10] Zheng Li, Liam O'Brien, Jacky Keung, and Xiwei Xu. Effort-oriented classification matrix of web service composition. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, pages 357–362, 2010.

[NP13] Artur Niewiadomski and Wojciech Penczek. Towards SMT-based Abstract Planning in PlanICS Ontology. In *Proc. of KEOD 2013 International Conference on Knowledge Engineering and Ontology Development*, pages 123–131, September 2013.

[NPS15] Artur Niewiadomski, Wojciech Penczek, and Jaroslaw Skaruz. Hybrid approach to abstract planning of web services. In *Service Computation 2015 : The Seventh International Conferences on Advanced Service Computing*, pages 35–40, 2015.

[RWE13] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[SNP13] Jaroslaw Skaruz, Artur Niewiadomski, and Wojciech Penczek. Automated abstract planning with use of genetic algorithms. In *GECCO (Companion)*, pages 129–130, 2013.

[SPAS03] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1:27–46, 2003.

[Szr15] Maciej Szreter. Bounded abstract planning in PlanICS based on graph databases. Technical Report 1031, ICS PAS, Ordona 21, 01-237 Warsaw, December 2015.

TripICS - a Web Service Composition System for Planning Trips and Travels (Extended Abstract)

Artur Niewiadomski[1], Wojciech Penczek[2]

[1] Institute of Computer Science, Siedlce University, Poland
artur.niewiadomski@uph.edu.pl
[2] Institute of Computer Science PAS, Warsaw and Siedlce University, Poland
wpenczek@gmail.com

1 Introduction

Automatic composition of Web services is a very active area of research which has provided a lot of important results [10, 1, 15, 6] as well as many implemented approaches [11, 12, 2, 3, 13]. In this paper we present the system TripICS - a real-life application of our Web service composition system PlanICS [8, 9, 13] to planning trips and travels around the world. While there are systems offering some support for planning excursions and travels [4, 5], our system uses advanced automated concrete planning methods [13, 16, 14]. TripICS is a specialization of the concrete planning viewed as a constrained optimization problem to the ontology containing services provided by hotels, airlines, railways, museums etc. The system finds an optimal plan (solution) satisfying the requirements of the user by applying the most efficient concrete planners of PlanICS based on a combination of an SMT-solver [7] with the nature inspired algorithm: GA, SA, and GEO [16, 14]. Contrary to PlanICS, the first phase of planning, called abstract planning, is realized by TripICS in a semi-automatic way by giving the user a possibility to choose the elements of an abstract plan using a Graphical User Interface (GUI). In the remainder we present: the description of the system TripICS, the theory behind it, and the implementation followed by some experimental results and conclusions.

2 TripICS Description

Our system is to allow the user for an easy and user-friendly planning of visits to interesting cities and places around the world in combination with travels in and out, arranged in the way satisfying the user's requirements. The general assumption is that the user would like to receive an optimal plan of a travel starting and ending in given locations and offering a possibility of visiting some specified cities within some specified dates. A plan is optimal if its quality value is the highest according to the given criteria. Below, we make the above description much more precise by giving three lists of requirements: 1) the user has to set (obligatory requirements), 2) the user can optionally set (optional requirements), and 3) the predefined quality requirements.

2.1 Obligatory Requirements

1. Trip starts and ends in two given locations (cities),
2. Trip starts from a given date (or a period of time) and lasts for a given number of days (optionally can be shorter or longer by a specified number of days),
3. Trip involves visiting given cities, each city within a specified minimal/maximal number of days,
4. Hotels with the free cancellation option are booked (optionally free cancellation is not required if this reduces the price by a given factor in %),
5. In each city to be visited, the attractions specified in the optional rules, are available within the period of stay.

2.2 Optional Requirements

1. In each city, attractions (museums, exhibitions, matches, concerts, restaurants etc.) to attend are specified,
2. Quality of hotels is specified by giving a minimal number of stars (0 - 5) and a minimal score (0 - 10),
3. Travels do not last longer than a given number of hours.

Clearly, a plan should be optimal in the sense that the travels should conveniently fit to the stays and the prices should be as low as possible for a required quality of hotels. The aim of TripICS is to return such plans if they exist. Formally, these plans need to satisfy the user requirements as well as the quality requirements specified below.

2.3 Quality Requirements

1. A travel connection between two cities is always direct if it exists,
2. Costs, durations, and the numbers of breaks of the travels are minimized,
3. Costs of the visits are minimized while their standards and durations are maximized.

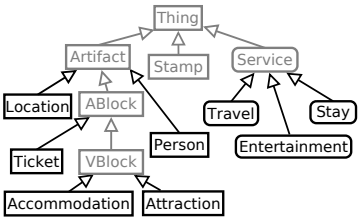
3 Theory behind TripICS

Typically, PlanICS realizes planning in three well defined phases called: abstract planning, offer collecting, and concrete planing, after receiving a user query specifying the requirements. In TripICS we depart from using an abstract planner, which does free the user from formulating a user query in the specification language. Instead, the user is given a possibility to set the obligatory and optional requirements about expected plans using GUI, described briefly in Section 3.2. All the user's choices, as well as the quality requirements, are automatically encoded as a user query and passed to the offer collector and the concrete planners. The available options result from the underlying ontology.

3.1 Ontology

This section discusses the ontology exploited by TripICS. Fig. 1 shows a part of the ontology corresponding to a travel domain. The ontology defines three service types $Travel$, $Stay$, and $Entertainment$ aimed at providing instances of the $Ticket$, $Attraction$, and $Accommodation$ object types (and operating also on objects of type $Person$ and $Location$) which are the trip elements constituting (among others) the abstract plan.

A ticket represents a journey from one location to another, for a certain price. An accommodation corresponds to a stay in some location, for a certain price as well. An attraction represents an admission ticket for an event, a reservation, or a confirmation that the attraction is available at the specified time. All these objects contain attributes describing contexts and details of the particular trip elements. We introduce also two auxiliary object types: $ABlock$ and $VBlock$. This is to avoid duplication of common attributes using the inheritance mechanism.



Object type	**Attributes' names and types**
$Location$	$continent$, $country$, $city$: $String$
$Person$	$name$: $String$; loc: $Location$
$ABlock$	$begin$, end: $Date$; $price$: $Real$; $type$: $String$
$Ticket$	$from, to$: $Location$; $breaks$: int
$VBlock$	$reviews$: $Real$; loc: $Location$
$Accommodation$	$stars$: Int; $freeCancel$: $Bool$
$Attraction$	$reserved$: $Bool$

Fig. 1. The TripICS ontology. The rectangles stand for object types while the rounded rectangles correspond to the service types. The types irrelevant for the working example are marked grey. The table describes the object types and their attributes.

For example, all the mentioned trip elements are described by the attributes $begin$, end, $price$, and $type$, and therefore they are inherited from the object type $ABlock$. The $type$ attribute defines the transportation type (bus, train, plane, ship, etc.), the accommodation type (hotel, guest house, hostel, apartment, etc.), or the attraction type (museum, exhibition, match, concert, restaurant etc.), when used in the $Ticket$, $Accommodation$, or $Attraction$ object, respectively. On the other hand, the number of breaks is specific to travels only, and thus the attribute $breaks$ is introduced in the object type $Ticket$. Each accommodation (and attraction) has been assigned a number stored in the attribute $reviews$ corresponding to the average score given by people who stayed there (or enjoyed the attraction) before. Similarly, since a fixed location is a common feature of accommodations and attractions, the attribute loc of type $Location$ is introduced in the class $VBlock$ and inherited by $Attraction$ and $Accommodation$ object types. The attributes are summarized in Fig. 1. Note that their meanings follow intuitively from their names.

3.2 Specifying requirements

First, using an intuitive GUI, the user inputs information about the dates and trip duration. The next step is to select the cities to be visited by clicking on the map or searching them by name. The cities are added to the list at the left hand side (see Fig. 2). Then, the user adds accommodations and attractions to enjoy in the particular cities, and inputs his preferences in the forms attached to the list.

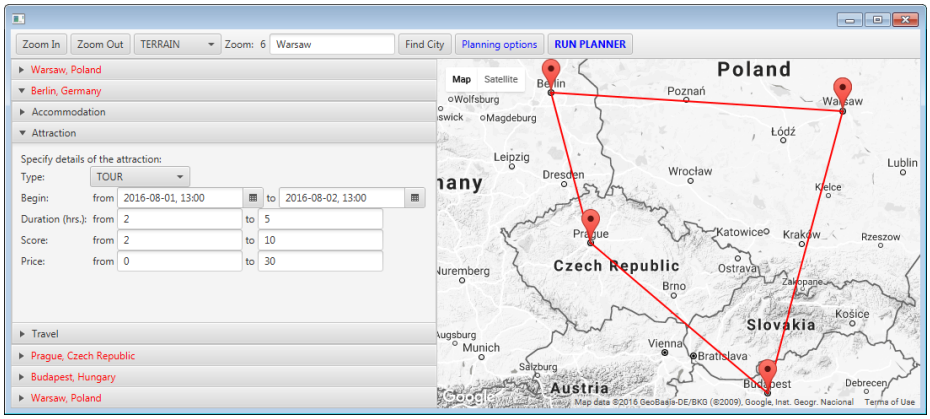


Fig. 2. TripICS GUI

Finally, the user starts the planning process using the dedicated button, and optionally sets some planner options, such as a planning algorithm (SMT, GA, IPH, SCGEO, SCSA[3] [16, 14]), timeout, maximal number of offers etc., as well as some parameters specific to the particular planning method, e.g., a population size of GA and IPH.

3.3 Collecting Offers and Planning

Basing on the city list and other data provided in the previous step, an abstract plan, i.e., a sequence of service types, is built. Next, this abstract plan is used by the *offer collector* (OC), i.e., the tool which in cooperation with the service registry queries real-world services. The service registry keeps an evidence of web services, registered accordingly to the service type system. Usually, each service type of the ontology represents a set of real-world services of similar functionality. For example, using the service type *Stay* one could register *Booking.com* as well as *Hilton* service.

OC queries web services of types present in the abstract plan and retrieves data called *offers*. An offer is a tuple of values representing a possible realization of one

[3] SMT - the SMT-based planner, GA- the GA-based planner, IPH - the initial population hybrid planner (SMT + GA), SCGEO - the SMT combined with GEO planner, SCSA - the SMT combined with SA planner, where SMT (Satisfiability Modulo Theories), GA (Genetic Algorithm), SA (Simulated Annealing), GEO (Generalised Extremal Optimization)

service type of the plan. Each value corresponds to an attribute of some object processed by the service type. The offers collected from a single service type of the plan constitute so called *offer sets*. The offers are searched by a *concrete planner* in order to find the best solution satisfying all constraints and maximizing the quality function. Thus, the concrete planning problem can be formulated as a constrained optimization problem (see [13]). Its solution consists in selecting one offer from each offer set such that all constraints are satisfied and the value of the quality function is maximized.

The constraints and the quality function result from the user requirements and preferences what is shown in the next subsection.

3.4 Constraints and Quality Function

The constraints and the quality function play a crucial role in the planning process. In this section, using a simple example, we show how the user requirements and preferences in combination with several general rules (described in Sec. 2) result in a set of constraints and a quality function.

Example 1. Assume that the user wants to make a trip on the 15th of August from Warsaw (W) to Berlin (B) and then back in a few days. In Berlin, he prefers to stay in a 3-star hotel for 3 days and during the visit he plans to take a city tour and attend a concert. The specified requirements result in an abstract plan consisting of the following 5 service types: ($Travel$, $Stay$, $Entertainment$, $Entertainment$, $Travel$). Then, OC searches for the matching offers, and retrieves the following example five offer sets ($O^1, \ldots, O^5$).

$O^1(Travel)$

id	begin	end	price	type	from	to	breaks
1	15.08, 10:40	15.08, 12:05	565	plane	W	B	0
2	15.08, 06:20	15.08, 07:45	565	plane	W	B	0
3	15.08, 07:20	15.08, 12:05	533	plane	W	B	1
4	15.08, 14:05	15.08, 19:18	170	train	W	B	0
5	15.08, 18:05	15.08, 22:58	276	train	W	B	0

$O^2(Stay)$

id	begin	end	price	type	score	loc	stars	freeCanc
1	15.08, 14	18.08, 11	1044	hotel	9.1	B	3	yes
2	15.08, 15	18.08, 11	1211	hotel	9.1	B	3	yes
3	15.08, 15	18.08, 12	1729	hotel	9.0	B	3	yes
4	15.08, 15	18.08, 11	1032	hotel	7.0	B	3	yes
5	15.08, 15	18.08, 12	1259	hotel	8.9	B	3	yes

$O^3(Entertainment)$

id	begin	end	price	type	score	loc	reserv.
1	15.08, 18	15.08, 21	84	tour	8.5	B	yes
2	16.08, 12	16.08, 15	84	tour	8.5	B	yes
3	16.08, 15	16.08, 18	84	tour	8.5	B	yes
4	17.08, 12	17.08, 15	79	tour	7.2	B	yes
5	17.08, 15	17.08, 18	79	tour	7.2	B	yes

$O^4(Entertainment)$

id	begin	end	price	type	score	loc	reserv.
1	16.08, 20	16.08, 23	280	concert	7.2	B	yes
2	16.08, 21	17.08, 1	130	concert	8.1	B	yes
3	17.08, 21	18.08, 1	110	concert	3.0	B	yes
4	17.08, 18	17.08, 22	580	concert	9.3	B	yes
5	16.08, 20	16.08, 23	164	concert	7.9	B	yes

$O^5(Travel)$

id	begin	end	price	type	from	to	breaks
1	18.08, 11:50	18.08, 16:30	429	plane	B	W	1
2	18.08, 15:10	18.08, 19:30	524	plane	B	W	1
3	18.08, 08:50	18.08, 10:10	561	plane	B	W	0
4	18.08, 09:37	18.08, 15:19	170	train	B	W	0
5	18.08, 14:37	18.08, 20:36	276	train	B	W	0

This example deals with a plan of length 5 where every service of the plan has 5 possible realizations. Thus, the search space is of size $5^5 = 3125$ as there is so many possible offer combinations. However, the number of plans (solutions) is much lower if we take constraints into account.

For example, assume that the user wants to synchronise travels and hotel in such a way that the time between arrival and hotel check-in is not longer than 3 hours. Similarly, the return travel should be not later than 3 hours after the hotel check-out time. After adding these two constraints the number of the possible solutions decreases to 2200. Another constraint could be to have at least a three-hour break between attractions. When this constraint is taken into account, there are only 1408 possible solutions. The underlying constraints are encoded by the following expressions: $(o_2.begin - o_1.end \leq 3)$, $(o_5.begin - o_2.end \leq 3)$, $(o_4.begin - o_3.end \geq 3)$, where o_i represents an offer from the i-th offer set.

As to the quality function, the user can choose between several schemes, but he can also enable/disable some of the function components. For example, if the user prefers only to minimize the total price, the quality function is expressed by $W_{price} * \sum_{i=1..5} o_i.price$, and the optimal solution is $(4, 4, 5, 3, 4)$ with the price 1561, where W_{price} is some negative constant (a weight). The numbers in the sequence correspond to the numbers of the offers in the corresponding offer sets. That is, both the travels are by train for the price of 170 each, the stay is in the cheapest hotel, and the cheapest tour and concert are chosen. However, if the user also wants to maximize the reviews of the stay and attractions, the quality function is then as follows: $W_{price} * \sum_{i=1..5} o_i.price + W_{score} * \sum_{i=2..4} o_i.score$. Assuming $W_{price} = -1$ and $W_{score} = 10$, we obtain the optimal solution $(4, 1, 3, 2, 4)$ where the accommodation and attractions with a low price and a high score are chosen. Fig. 3 presents the resulting plan.

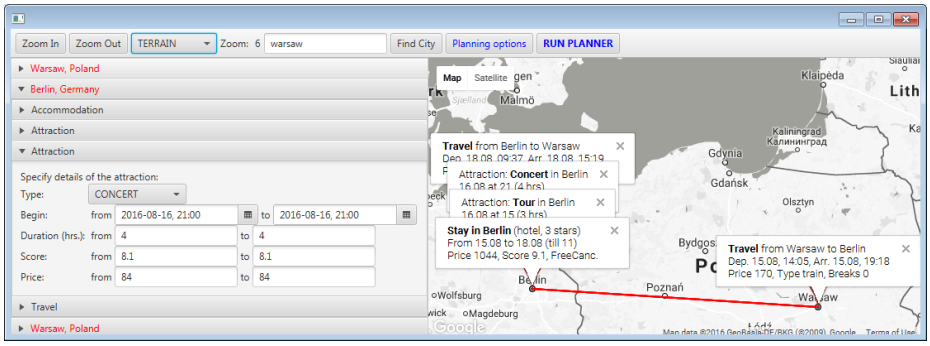


Fig. 3. The example plan

4 TripICS Implementation and Experiments

The TripICS application is implemented in Java. It consists of several planning engines, Offer Collector, and the GUI module. GUI exploits GMapsFX [17] project which provides a wrapper to the Google Map's Javascript API, allowing to exploit and interact with maps using a pure Java. The map is the central component in GUI. It is used to

Table 1. The experimental results for the travel benchmarks. In each entry of the table having three rows of values, the first row contains the average values (bold), the second row contains the best values (normal font), the third row contains the standard deviation (italic).

	SMT	**SCSA**	**SCGEO**	**GA**		**IPH$_1$**	**IPH$_{500}$**
Instance	**t[s] / Q**	**t[s] / Q**	**t[s] / Q**	**t[s] / Q**	**P [%]**	**t[s] / Q**	**t[s] / Q**
T1	**52.4 / 228.3**	**0.9 / 222.0**	**0.8 / 221.4**	**1.7 / 201.5**	80	**1.8 / 202.5**	**1.9 / 222.8**
	47 / 228.3	0.8 / 222.0	0.7 / 222.0	1.5 / 222.0		1.6 / 222.0	1.6 / 228.3
	3.6 / 0	*0.1 / 0.0*	*0.0 / 4.1*	*0.1 / 13*		*0.1 / 18.5*	*0.1 / 6.3*
T2	**400.6 276.2**	**1.2 / 268.9**	**1.0 / 268.4**	**1.8 / 222.7**	92	**1.9 / 223.6**	**1.7 / 277.1**
	400.3 / 276.2	1.0 / 276.2	0.9 / 276.2	1.7 / 284.2		1.8 / 284.2	1.6 / 284.2
	0.3 / 0	*0.1 / 13.8*	*0.1 / 10.9*	*0 / 37.3*		*0.1 / 38.5*	*0.1 / 7.6*
T3	**400.3 / 79.3**	**1.5 / 270.3**	**1.2 / 249.7**	**1.8 / 235.7**	72	**2.1 / 209.7**	**2.8 / 266.2**
	400.3 / 79.3	1.3 / 290.4	1.0 / 290.4	1.6 / 306.1		2 / 269.3	2.5 / 290.3
	0 / 0	*0.1 / 16.8*	*0.1 / 26.0*	*0.1 / 33.8*		*0.1 / 39.6*	*0.2 / 4.9*
T4	**400.2 / 106.9**	**1.7 / 217.3**	**1.8 / 216.7**	**3.3 / 178.5**	70	**3.2 / 169**	**3.4 / 173.6**
	400.2 / 106.9	1.6 / 217.3	1.7 / 217.3	3.1 / 238.8		3.1 / 214.8	2.7 / 210.4
	0 / 0	*0.1 / 0.0*	*0.1 / 4.1*	*0.1 / 27.4*		*0.1 / 22.5*	*0.4 / 10.5*
T5	**400.3 / 73.4**	**2.1 / 191.5**	**2.2 / 190.8**	**3.2 / 211.2**	76	**3.5 / 193.8**	**7.1 / 331.3**
	400.3 / 73.4	1.9 / 191.7	2.0 / 191.7	3 / 281.4		3.4 / 336.9	3.7 / 344.8
	0 / 0	*0.1 / 0.5*	*0.1 / 1.6*	*0.1 / 33.7*		*0.1 / 39.3*	*1.9 / 9.2*
T6	**400.4 / 12.5**	**2.5 / 277.6**	**2.7 / 262.0**	**3.2 / 206.9**	80	**3.8 / 199.7**	**8 / 304.1**
	400.4 / 12.5	2.3 / 320.3	2.4 / 320.3	3.1 / 286.1		3.7 / 264.4	7.8 / 308.5
	0 / 0	*0.1 / 19.6*	*0.2 / 16.4*	*0.1 / 36.7*		*0.1 / 31.5*	*0.2 / 3.1*
T7	**400.3 / 19.6**	**2.6 / 210.9**	**3.6 / 210.1**	**4.7 / 158.8**	80	**5.6 / 186.2**	**4.3 / 181.2**
	400.3 / 19.6	2.5 / 214.2	3.4 / 214.2	4.6 / 258.6		5.1 / 211.1	4.1 / 191.9
	0 / 0	*0.1 / 5.8*	*0.1 / 6.2*	*0.1 / 38.1*		*0.2 / 14.3*	*0.1 / 5.8*
T8	**400.4 / 38.1**	**3.0 / 162.0**	**4.0 / 155.4**	**4.6 / 177.3**	60	**6.1 / 95.1**	**9.3 / 226.4**
	400.3 / 38.1	2.8 / 180.4	3.8 / 180.4	4.3 / 237.0		5.7 / 143.2	9.1 / 242.3
	0.1 0	*0.1 18.6*	*0.1 20.9*	*0.1 38.8*		*0.2 23.5*	*0.1 11.2*
T9	**400.5 / 35**	**3.6 / 258.5**	**4.5 / 255.0**	**4.5 / 180.2**	62	**6.2 / 187,4**	**10.8 / 187.1**
	400.4 / 35	3.4 / 266.2	4.2 / 266.2	4.4 / 285.6		5.4 / 250.3	10.4 / 227.7
	0.1 / 0	*0.1 / 7.6*	*0.1 / 10.6*	*0 / 38.2*		*0.6 / 33*	*0.2 / 19.9*

specify user requirements as well as to visualize plans. The application is still being extended and improved.

We have evaluated the efficiency of TripICS focusing on the planning modules. At the moment our Offer Collector supports only a limited number of services. Thus, we have used several benchmarks generated by our software Offer Generator. They have been scaled by the length of a plan and the number of offers. Plans of length 5 have been found for the benchmarks T1, T2, and T3, of length 9 for T4, T5, and T6, and of length 13 for the remaining examples. The number of offers equals $2^8 = 256$ for the benchmarks T1, T4, and T7; $2^9 = 512$ for T2, T5, and T8; and $2^{10} = 1024$ for the other cases. This gives a number of the potential solutions varying from $256^5 = 2^{40}$ for T1 to $1024^{13} = 2^{130}$ for T9. The tested examples involved from 13 to 43 constraints. We have compared several planning algorithms taking into account the computation time and the quality of the solutions found. The methods combining SMT with nature-

inspired algorithms appear to be the most efficient. They are able to find solutions of very high quality within a few seconds only, which makes them acceptable for the user. The detailed results are given in Table 1 and summarised in Fig. 4.

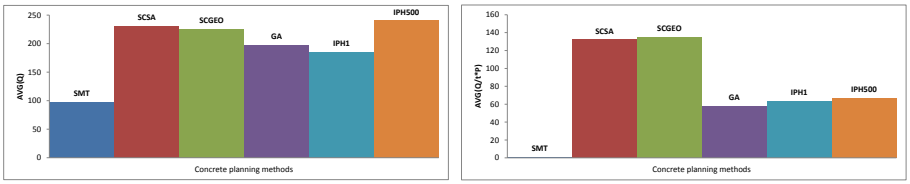


Fig. 4. A comparison of the average quality of solutions produced by the concrete planning methods (left) and average efficiency comparison of the concrete planning methods (right). By efficiency we mean $quality/time * probability$.

5 Conclusions

We have presented a preliminary version of our system TripICS, which can be used for planning trips and travels around the world. Our motivation for developing this system was twofold. Firstly, we wanted to show that web service composition can be successfully used in practice for real-life applications, and secondly our aim is to offer a new useful tool, which could be publicly used.

References

1. S. Ambroszkiewicz. Entish: A language for describing data processing in open distributed systems. *Fundam. Inform.*, 60(1-4):41–66, 2004.
2. D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.
3. D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, and F Patrizi. Automatic service composition and synthesis: the roman model. *IEEE Data Eng. Bull.*, 31(3):18–22, 2008.
4. D. Damljanovic and V. Devedzic. Applying semantic web to e-tourism. In *In Ma, Z.,Wang, H. (Eds.), The Semantic Web for Knowledge and Data Management: Technologies and Practices*, pages 243–265. IGI Global, New York, 2008.
5. D. Damljanovic and V. Devedzic. Applying semantic web to e-tourism. In *In Mehdi Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology*, pages 3426–3432. IGI Global, Hershey, 2009.
6. G. De Giacomo, M. Mecella, and F. Patrizi. Automated service composition based on behaviors: The roman model. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 189–214. Springer, 2014.
7. L. De Moura and N. Bjorner. Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54(9):69–77, September 2011.

8. 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.
9. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, and A. Zbrzezny. Planics - a web service composition toolset. *Fundam. Inform.*, 112(1):47–71, 2011.
10. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
11. Z. Li, L. O'Brien, J. Keung, and X. Xu. Effort-oriented classification matrix of web service composition. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, pages 357–362, 2010.
12. W. Nam, H. Kil, and D. Lee. Type-aware web service composition using boolean satisfiability solver. In *Proc. of CEC'08 and EEE'08*, pages 331–334, 2008.
13. A. Niewiadomski, W. Penczek, J. Skaruz, M. Szreter, and M. Jarocki. SMT versus Genetic and OpenOpt Algorithms: Concrete Planning in the PlanICS Framework. *Fundamenta Informaticae*, 135(4):451–466, 2014.
14. A. Niewiadomski, J. Skaruz, P. Switalski, and W. Penczek. Concrete Planning in PlanICS Framework by Combining SMT with GEO and Simulated Annealing. *To appear in Fundamenta Informaticae*, 2016.
15. 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, 2005.
16. J. Skaruz, A. Niewiadomski, and W. Penczek. Hybrid Planning by Combining SMT and Simulated Annealing. In Z. Suraj and L. Czaja, editors, *Proceedings of the 24th International Workshop on Concurrency, Specification and Programming, Rzeszow, Poland, September 28-30, 2015.*, volume 1492 of *CEUR Workshop Proceedings*, pages 173–176. CEUR-WS.org, 2015.
17. R. Terpilowski. GMapsFX - a JavaFX API for Google Maps, 2016.

An Algorithmic Way to Generate Simplexes for Topological Data Analysis

Krzysztof Rykaczewski[1,2], Piotr Wiśniewski[2], and Krzysztof Stencel[2,3]

[1] DLabs, Lęborska 3B, 80-230 Gdańsk
krykaczewski@gmail.com
[2] Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Chopina 12/18,
87-100 Toruń, Poland
pikonrad@mat.umk.pl
[3] Faculty of Mathematics, Informatics and Mechanics
University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
stencel@mimuw.edu.pl

Abstract. In this article we present a new algorithm for creating simplicial Vietoris-Rips complexes that is easily parallelizable using computation models like MapReduce and Apache Spark. The algorithm does not involve any computation in homology spaces.

Keywords: data analysis, topology, simplicial complex

1 Introduction

The article of Silva and Grist [2] showed that the algebraic topology was a very practical tool. In this paper its authors analyse the coverage of an area with a network of sensors. This network is interpreted as a simplicial complex. Its homology type is then computed and exploited. If this complex is homotopy equivalent with point, the coverage is complete. Therefore, the authors translated a technical problem into a mathematical problem.

In the article [3] Grist searches for topology structures in big data sets. Again, this results in building simplicial complexes and analysing them.

The abovementioned works inspire to ask the question how to build simplicial complexes efficiently and how to analyse them. In this article w propose a novel algorithm to build the simplicial complex. This algorithm has the unique property that it can be implemented within massive parallel computational models like MapReduce and Apache Spark.

2 Background

Hereafter we assume that the data set given is a finite set of points $P := \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$. This data can come up as a result of some physical experiments, psychological tests etc., and can generally be high-dimensional. We

have no insight into how the data actually look like and how it is embedded, but it would be enough to know their "shape" to say something about how it is are arranged in the original space.

In recent papers of Carlsson and collaborators [1] presented ideas to explore some properties of high-dimensional data sets. They use algebraic topology tools to find certain characteristics of the data or its simpler representation. This approach allows to discover objects and features that are immersed in high-dimensional space.

We start presentation of basic facts from the notion of a simplex. Colloquially speaking, simplex is a generalization of a segment, a triangle and a tetrahedron to any dimension. More formally, suppose the $k+1$ points $p_0, \ldots, p_k \in \mathbb{R}^k$ are affinely independent, which means $p_1 - p_0, \ldots, p_k - p_0$ are linearly independent. Then, *k-simplex* determined by them is the set of points

$$\sigma := \left\{ \lambda_0 p_0 + \cdots + \lambda_k p_k \mid \lambda_i \geq 0,\ \ 0 \leq i \leq k,\ \ \sum_{i=0}^{k} \lambda_i = 1 \right\}, \tag{1}$$

i.e. k-simplex is a k-dimensional polytope which is the convex hull of its $k+1$ vertices. We often write $\sigma = [p_0, \ldots, p_k]$ for short. The points $p_0, \ldots, p_k$ are called *vertices* of the k-simplex. Each simplex is uniquely determined by its vertices. Any subset of vertices constitutes simplex with dimension smaller that the whole one, called a *face* of simplex.

We say two vertices v and w are *neighbors* if there is 1-simplex (edge) connecting them.

Definition 1. *A simplicial complex $\mathcal{K}$ is a set of simplexes that satisfies the following conditions:*

1. *Any face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$.*
2. *The intersection of any two simplexes $\sigma_1,\ \sigma_2 \in \mathcal{K}$ is either $\emptyset$ or a face of both σ_1 and σ_2.*

Having data set P, we can use it to build a variety of complexes. We shall now formulate two most frequent definitions.

Definition 2. *For a discrete set of points $P \subset \mathbb{R}^d$ and a parameter $\epsilon > 0$, we define the* Čech complex *of radius ϵ as $\check{C}(P, \epsilon)$ by*

$$\check{C}(P, \epsilon) := \left\{ [p_1, p_2, \ldots, p_k] \mid \{p_1, p_2, \ldots, p_k\} \subset P,\ \bigcap_i B(p_i, \epsilon/2) \neq \varnothing \right\}, \tag{2}$$

where $B(p, \epsilon)$ is the closed ball of radius ϵ centered at p.

The nerve theorem states that Čech complex $\check{\mathrm{C}}(P, \epsilon)$ has homotopy type of the union of closed balls with radius $\epsilon/2$ around data P This means that the Čech complex gives faithful representation of thickened data (they are common "shape").

Vietoris-Rips complex is closely related to the Čech complex.

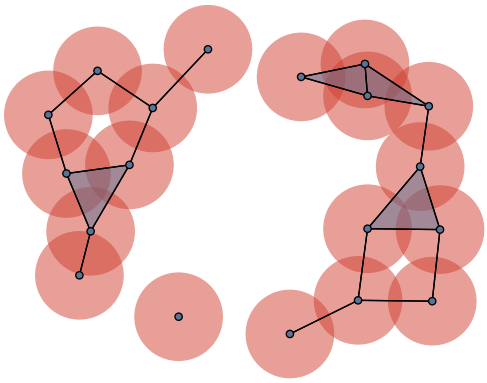


Fig. 1. Example of a Čech complex.

Definition 3. *For a given $\epsilon > 0$, the* Vietoris-Rips complex *(later called Rips complex) $R(P, \epsilon)$ is the largest simplicial complex that shares the same 1-skeleton (graph) as the Čech complex $\check{C}(P, \epsilon)$. More explicitly,*

$$R(P, \epsilon) := \{\sigma := [p_1, p_2, \ldots, p_k] \mid \{p_1, p_2, \ldots, p_k\} \subset P, diam(\sigma) \leq \epsilon\}, \quad (3)$$

where $diam(\sigma)$ is the diameter of simplex σ.

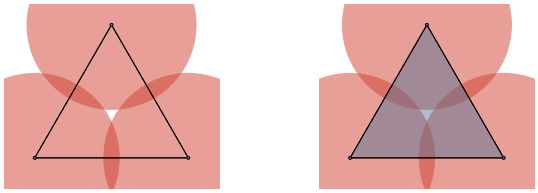


Fig. 2. Comparison of Čech (left) and Vietoris-Rips (right) complexes.

Lemma 1 (Vietoris-Rips). *For every finite set of points $P \subset \mathbb{R}^d$ and $r \geq 0$,*

$$\check{C}(P, \epsilon) \subset R(P, \epsilon) \subset \check{C}(P, 2\epsilon). \quad (4)$$

Thus, if the Čech complex at the same time for ϵ and 2ϵ approximates the data in a good way, then Vietoris-Rips complex do it as well. This estimate can be further improved [2].

Remark 1. Čech complex is difficult to calculate, but it is quite small. However, Vietoris-Rips complex is easy to calculate, but is usually very big. Both Čech

and Vietoris-Rips complexes can produce simplicial complex of dimension greater than the considered space. Therefore, there are considered many alternatives for them: Delaunay Complex, Alpha Complex, (Lazy) Witness Complex, Mapper Complex, Sparsified Rips complex, Graph Induced Complex (GIC).

For small $\epsilon > 0$ we have disjoint set of balls, whereas for big ϵ the covering by balls is *simply connected* (without holes), so we are totally losing the knowledge about the data structure. Since (in general) we do not know which radius ϵ to take, we consider them all and obtain in this way a filtration of simplicial complexes, i.e.

$$\check{\mathrm{C}}(P, \epsilon_1) \subset \check{\mathrm{C}}(P, \epsilon_2) \subset \ldots \subset \check{\mathrm{C}}(P, \epsilon_n), \tag{5}$$

for $0 < \epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_n$. With the change of radius also changes the topology of the data: some holes are created, and some disappear. Roughly speaking, those holes that last longest represent the shape of data. It is usually represented in the form of the so-called *barcode* [3].

Let us recall the following definition from topology.

Definition 4. *Two topological spaces X and Y are called* homotopy equivalent *if there exist continuous maps $f\colon X \to Y$ and $g\colon Y \to X$, such that the composition $f \circ g$ is homotopic (i.e. it can be deformed in a continuous way) to the identity* id_Y *on Y, and $g \circ f$ is homotopic to* id_X.

3 Related Work

Data anaylsis is a process of modeling data in order to discover useful information. Unfortunately, mining high-dimensional data is very challenging. Therefore, a number of methods was created so as to make it easier for researchers. One of the youngest but rapidly growing field now is topological data analysis (TDA).

TDA is an approach to analyse data sets using techniques from topology. This method relies on calculation of some properties of the space/data, which maintain (are invariant) under certain transformations. Although most of topological invariants are difficult to calculate, there is an algorithm which can calculate homology groups and barcodes [4, 5]. These groups are often used in applications [3].

Topological methods due to their nature can be used to cope with many problems where traditional methods fail. It has broad application in coverage problem in sensor network [2, 3], detecting breast cancer [6], computer vision [7], detection of periodicity in time series (behaviour classification) [8] etc.

Another issue addressed in this subject is the speed of algorithms. In paper [9] Dłotko et al. presented distributed algorithm for homology computation over a sensor network. Their technique involve reduction and coreduction of simplicial complexes.

Regardless of the algebraic approach, there are created algorithms that do not use calculations on homology groups to obtain some approximation of the shape of data [10]. Algorithm proposed in the last paper can have problems with

more complicated structures, like the sphere, but it is shown that it does well with coverage in sensor networks.

Notice that apart from using Čech and Rips complexes there are other directions in computational topology, but we do not discuss them here [2].

4 A Method to Build Complex using MapReduce

In this Section we present an efficient method to build Vietoris-Rips complexes. We make the following assumptions:

- $\mathcal{R}$ is a Euclidean space.
- A fixed number of dimensions n is given.
- $\mathcal{M} \subset \mathcal{R}^n$ is the set of input data points.
- We assume that each input data point has a unique identifier $id(x)$. Those identifiers are items of an linearly ordered set.
- For any $i \in \{1, 2, \ldots, n\}$ we define $\Pi_i : \mathcal{R}^n \to \mathcal{R}$ to be the projection to the i-th dimension.
- In $\mathcal{R}^n$ we assume the Euclidean distance. For any two $x, y \in \mathcal{R}^n$ let $|x, y|$ denote the Euclidean distance of x and y.
- Let us choose $\epsilon > 0$. We are going to build the Vietoris-Rips complex $R(\mathcal{M}, \epsilon)$

The first step to build a Vietoris-Rips complex is to find all pair of points $x, y \in \mathcal{M}$ such that $|x, y| < \epsilon$. If the set $\mathcal{M}$ is large, computing distances between all pairs of points is too time consuming. If the number of dimensions n is equal to 1, we can assign all points to segments of the form $\langle k\epsilon, (k+1)\epsilon)$. Then, instead of computing distances for all pairs, we consider only those pairs of points that fall into the same segment or two neighbouring segments. As the result, each point is paired only with points from three segments. This can significantly accelerate the computation. Of course, in a one-dimensional space, a simpler method that encompasses sorting and sequential scanning will be faster. However, this method cannot be extended to more dimensions.

For a two-dimensional space we can apply a method similar to the former one described above. Instead of segments, we have then squares with sides of length ϵ. Eventually, each point will be paired for distance computation only with points from nine other squares. The three-dimensional case is similar, but we use cubes with side of length ϵ and each point is paired with points from 27 cubes.

Unfortunately, enormous growth of the number of dimensions prohibits direct usage of this method. In case of a 100-dimentional space, the "segments" will be 100-dimensional hypercubes with ϵ-side. Each point will be paired for distance computation with points from 3^{100} hypercubes. Obviously the naïve method will amount to be better for sure in this case.

However, we can apply the abovementioned three-dimensional method in multi-dimensional spaces *indirectly*. If we choose three dimensions i, j, k, we can consider projecting the whole space to them and further search for close pairs like

in a three-dimensional space. Let us consider the projection $\Pi = \Pi_i \times \Pi_j \times \Pi_k : \mathcal{R}^n \to \mathcal{R}^3$. Obviously for each pair $x, y \in \mathcal{R}^n$ the following inequality holds:

$$|\Pi(x), \Pi(y)|_3 \leq |x, y|,$$

By $|\ldots|_3$ we denote the Euclidean distance in $\mathcal{R}^3$.

We will assign all points to ϵ-sided cubes according to their projections onto the chosen dimensions. Then, it is enough to pair each point for distance computation only with points residing in the same cube or in 26 neighbouring cubes. If two points are not assigned to the same cube or neighbouring cubes, the distance of their projections onto the chosen dimensions is larger than ϵ. Therefore, so is their distance in $\mathcal{R}^n$ and their pair does not have to be considered.

The efficiency of this method notably depends on the choice of the three projected dimension. Chosen dimensions may reduce the number of distance computations required. Our first idea was to compare the projection onto each single dimension with the uniform distribution using the algorithm based on *Kullback–Leibler divergence* [11]. However, this approach required two scans of input data. Moreover, it is hard to distinguish more dense and more sparse distributions. The problem lies rather in sparseness (with respect to ϵ) that uniformity. As the result, we invented the following quality measure of dimensions.

For a given dimension i and an integer t we define:

$$\sigma(i, t) = |\{x \in \mathcal{M}; t\epsilon \leq \Pi_i(x) < (t+1)\epsilon\}|$$

The number $\sigma(i, t)$ tells how many points will fall into the segment identified by t if we project to the dimension i. Note that if a point x satisfies the condition in the definition above, then $t = \lfloor \Pi_i(x)/\epsilon \rfloor$. $\lfloor a \rfloor$ is the largest integer not greater than a.

For a dimension $i \in \{1, 2, \ldots, n\}$ and an input data point $x \in \mathcal{M}$ we define $t_i(x)$ to be the integer:

$$t_i(x) = \lfloor \Pi_i(x)/\epsilon \rfloor.$$

Observe that the computation of all non-zero values of σ is possible using only a single scan of input data. Moreover, it is easily implementable in the MapReduce computation model. The mapper emits $\langle i, t_i(x) \rangle$ for each input data point $x \in \mathcal{M}$ and for each dimension i. The reducer simply counts the number of occurrences of pairs $\langle i, t \rangle$.

Non-dispersion measure of a dimension i is the number

$$\Delta_i = \sum_{t \in \mathcal{Z}} \sigma(i, t)^2$$

Now, for each dimension we compute its non-dispersion measure by means of the MapReduce procedure presented above. Then we chose three dimensions with smallest non-dispersions.

Assume i, j, k to be the three dimensions with smallest non-dispersion. The algorithm to find all pairs of points closer that ϵ using these dimensions can be easily parallelized using the MapReduce computation model.

The mapper reads all input data points and for each point x emits 27 key-value pairs:

$$\langle\langle t_1, t_2, t_3\rangle, x\rangle$$

where t_1, t_2, t_3 satisfy the following conditions:

$$\begin{aligned} t_1 &\in \{\ t_i(x) - 1,\ t_i(x),\ t_i(x) + 1\ \} \\ t_2 &\in \{\ t_j(x) - 1,\ t_j(x),\ t_j(x) + 1\ \} \\ t_3 &\in \{\ t_k(x) - 1,\ t_k(x),\ t_k(x) + 1\ \} \end{aligned}$$

A single reducer collects all pairs that came with a given key $\langle t_1, t_2, t_3\rangle$. If a pair x, y satisfies the conditions $id(x) < id(y)$, $t_1 = t_i(x)$, $t_2 = t_j(x)$, $t_3 = t_k(x)$, then the distance of points x, y is computed. The three dimension equalities assure that x lies in the very ϵ-sided cube $\langle t_1, t_2, t_3\rangle$. If $|x, y| < \epsilon$, the pair $\langle x, y\rangle$ is added to the result.

The collection of all pairs $\langle x, y\rangle$ that satisfy $|x, y| < \epsilon$ is the most time-consuming phase of the construction of the Vietoris-Rips complex. Those pairs are single-dimensional simplexes. The second phase computes triplets $\langle x, y, z\rangle$ such that the three points are pairwise closer than ϵ and $id(x) < id(y) < id(z)$. Following phases consist in identifying longer and longer lists of points that satisfy analogous conditions. It can be done as in the Apriori algorithm to find frequent subsets, because if all points in a set are closer than ϵ that the same must hold for all its subsets. If ϵ is not too big, these phases are not time-consuming. However, it is much simpler to build higher-dimensional simplexes using the following lemma.

Lemma 2 (Raising the dimension of simplexes). *Let R be a Vietoris-Rips complex. A simplex $a := \langle x_1, ..., x_n, x_{n+1}, x_{n+2}\rangle$ belongs to R if and only if all the three following simplexes are also in R:*

$$\begin{aligned} b &:= \langle x_1, ..., x_n, x_{n+1}\rangle \\ c &:= \langle x_1, ..., x_n, x_{n+2}\rangle \\ d &:= \langle x_{n+1}, x_{n+2}\rangle \end{aligned}$$

Proof. The "only if" part is straightforward. If $a \in R$, then all its subsimplexes obviously belong to R. Thus, so do b, c, d.

The proof of the "if" part is based on the following property of Vietoris-Rips complexes. If a simplex a belongs to R, then all its one-dimensional subsimplexes also belong to R. Assume a one-dimensional simplex $e := \langle x_j, x_k\rangle \subset a$, where $j, k \in \{1, \ldots, n+2\}$. We will show that $e \in R$.

We have to consider three cases. Firstly, if $j = n + 1 \wedge k = n + 2$, then $e = d \in R$. Secondly, if $j < n + 1 \wedge k = n + 2$, then $e \subset c \in R$, thus $e \in R$. Thirdly, if $k < n + 2$, then $j < n + 1$, thus $e \subset b \in R$ and also $e \in R$.

Since the choice of j and k is arbitrary, the "if" part has been proven. So has been the whole lemma. □

As the result of the first phase of the algorithm, we have a collection *pairs* that contains all pairs $\langle x_i, x_j\rangle$ such that

$$|x_i, x_j| < \epsilon \wedge id_j < id_k$$

We recall that id_i is the identifier of the point x_i.

In the second phase of the algorithm, we build collections of simplexes of subsequent dimensions n that belong to the Vietoris-Rips complex R. The algorithm stops when for a given n we get the empty list of simplexes.

For a given n, we search for all b, c and d that satisfy the constraints of Lemma 2. The symbols introduced in Lemma 2 will prove to be handy again. Therefore, we perform the following steps:

1. We convert the collection of simplexes $\langle x_1, x_2, \ldots, x_{n+1}\rangle$ of the dimension n to the collection of pairs $\langle\langle x_1, x_2, \ldots, x_n\rangle, x_{n+1}\rangle$.
2. We compute a natural self-join of this set of pairs using the first coordinate. As the result we get the set of all triplets: $\langle\langle x_1, x_2, \ldots, x_n\rangle, x_j, x_k\rangle$ (a as in Lemma 2) such that there are pairs $\langle\langle x_1, x_2, \ldots, x_n\rangle, x_j\rangle$ (b) and $\langle\langle x_1, x_2, \ldots, x_n\rangle, x_k\rangle$ (c) in the previous step.
3. We leave (select) only those triples that satisfy $id_j < id_k$.
4. We equi-join the remaining triplets with the set of all pairs using the last two coordinates of the triplets. We do it to assure that the last two coordinates of triplets (d) form a one-dimensional simplex belonging to the complex.
5. We flatten remaining triplets to get complexes of the dimension $n+1$ of the form $\langle x_1, x_2, \ldots, x_n, x_j, x_k\rangle$.

Note, that this procedure contains only operations that are expressible in relational algebra (or calculus). Therefore, the whole algorithm is formulated in a highly abstract language that is known to be easy to optimise and execute in parallel. We exploited this possibility and implemented it in the Apache Spark for efficient execution on large computing infrastructures of commodity computers. An example Python code that implements the whole algorithm is shown below.

```
komplex = []
komplex.append(points.map(lambda p: p[0]))
komplex.append(pairs)

# initial
base = pairs
pairsAsKey = pairs.map(lambda p: (p, 1))
noAnylizedObjects = pairs.count()

#iterated
while noAnylizedObjects > 0:
    base = base.map(lambda a:(a[:-1], a[-1]))
    base = base.join(base) \
               .filter(lambda a: a[1][0] < a[1][1]) \
               .map(lambda a: (a[1] , a[0])) \
               .join(pairsAsKey) \
```

```
            .map(lambda a:rigthTupleSum(a[1][0], a[0]))
noAnylizedObjects = base.count()
if noAnylizedObjects > 0:
    komplex.append(base)
```

5 Conclusions

In this article we have presented an efficient method to build Vietoris-Rips complexes. Moreover, this method is easily parallelizable using the MapReduce computation model or Apache Spark. Further research will focus on algorithms that reduce constructed complexes. It will facilitate finding homology types of complexes. As a result it will also allow assessing of the correctness of the selection of the ϵ value.

References

1. Carlsson, G., Ishkhanov, T., De Silva, V., Zomorodian, A.: On the local behavior of spaces of natural images. International journal of computer vision **76** (2008) 1–12
2. De Silva, V., Ghrist, R.: Coverage in sensor networks via persistent homology. Algebraic & Geometric Topology **7** (2007) 339–358
3. Ghrist, R.: Barcodes: the persistent topology of data. Bulletin of the American Mathematical Society **45** (2008) 61–75
4. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. Discrete and Computational Geometry **28** (2002) 511–533
5. Zomorodian, A., Carlsson, G.: Computing persistent homology. Discrete & Computational Geometry **33** (2005) 249–274
6. Nicolau, M., Levine, A.J., Carlsson, G.: Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. Proceedings of the National Academy of Sciences **108** (2011) 7265–7270
7. Freedman, D., Chen, C.: Algebraic topology for computer vision. Computer Vision (2009) 239–268
8. Vejdemo-Johansson, M., Pokorny, F.T., Skraba, P., Kragic, D.: Cohomological learning of periodic motion. Applicable Algebra in Engineering, Communication and Computing **26** (2015) 5–26
9. Dłotko, P., Ghrist, R., Juda, M., Mrozek, M.: Distributed computation of coverage in sensor networks by homological methods. Applicable Algebra in Engineering, Communication and Computing **23** (2012) 29–58
10. Ćwiszewski, A., Wiśniewski, P.: Coverage verification algorithm for sensor networks. In: Computer Applications for Bio-technology, Multimedia, and Ubiquitous City. Springer (2012) 397–405
11. Amari, S.I., Cichocki, A., Yang, H.H., et al.: A new learning algorithm for blind signal separation. Advances in neural information processing systems (1996) 757–763

Extrapolation of an Optimal Policy using Statistical Probabilistic Model Checking

Artur Rataj[1] and Bożena Woźna-Szcześniak[2]

[1] IITiS, Polish Academy of Sciences, ul. Bałtycka 5, 44-100 Gliwice, Poland
arturrataj@gmail.com
[2] IMCS, Jan Długosz University
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
b.wozna@ajd.czest.pl

Abstract. We show how to extrapolate an optimal policy controlling a model, which is itself too large to find the policy directly using probabilistic model checking (PMC). In particular, we look for a global optimal resolution of non–determinism in several small Markov Decision Processes (MDP) using PMC. We then use the resolution to find a respective set of decision boundaries representing the optimal policies found. Then, a hypothesis is formed on an extrapolation of these boundaries to an equivalent boundary in a large MDP. The resulting hypothetical extrapolated decision boundary is statistically approximately verified, whether it indeed represents an optimal policy for the large MDP. The verification either weakens or strengthens the hypothesis. The criterion of the optimality of the policy can be expressed in any modal logic that includes the probabilistic operator $\mathtt{P}_{\sim p}[\cdot]$, and for which a PMC method exists.

Keywords: probabilistic model checking, statistical model checking, non–determinism, optimal policy, extrapolation.

1 Introduction

Probabilistic model checking (PMC) [4] refers to a range of techniques for a formal analysis of a stochastic system, which is usually a state transition system with transitions labelled by probability values.

A policy of a decision maker (an agent), controlling a Markov Decision Process (MDP), resolves a non–deterministic choice, which exist in each state of an MDP in the form of a number of probability distributions over states, of which one is arbitrarily chosen (for details see Sec. 2). An optimal policy [12], in respect to a given property, may in particular correspond to either the minimum or maximum value of the property. In this paper we consider an MDP with properties specified in any modal logic that includes the probabilistic operator $\mathtt{P}_{\sim p}[\cdot]$, for which exists a PMC method. A common example of such a logic, for which efficient model checkers exist, is *Probabilistic Computation Tree Logic* (PCTL) [3].

Statistical probabilistic model checking (SPMC) [13, 8], including Monte Carlo simulation and sampling, involves a generation of a large number of random paths in a stochastic model, evaluating a given property on each path, and finally statistically aggregating all these evaluations in order to approximate a correct value of a property.

We address the issue of an estimation of an optimal policy in a model, which is too large to have that policy found using PMC, and also too large to have that policy estimated using SPMC, if an initial, sufficiently precise approximation ζ of the policy is unknown. In order to obtain ζ, we first find a number of equivalent optimal policies in several scaled-down versions of the large model. Then, we pose a hypothesis on an extrapolation of these policies to the large model. Finally, we strengthen or weaken the hypothesis using SPMC, which approximately verifies, whether the extrapolated policy is optimal by checking whether the policy has the largest fitness, when compared to a number of its close variants.

In particular, we begin with searching for a global optimal resolution of non–determinism [7] in several small MDPs, that model smaller versions of the system we are interested in. We then estimate equations of decision boundaries, each representing one of the obtained optimal policies. A hypothesis is then formed on extrapolating the equations to a large MDP. The resulting hypothetical extrapolated decision boundary is finally approximately verified by estimating if its fitness is locally maximal using a Monte Carlo DTMC simulator.

The paper is constructed as follows. In Sec. 2 we define the formalism used. In Sec. 3 we propose a technique which extrapolates and verifies an optimal policy. In Sec. 4 we present a case study. In the last section we conclude the paper.

2 Preliminaries

Let us define the formalism used throughout the paper. It is fairly standard and follows [4], where the reader will find an in–depth description.

2.1 Discrete–Time Markov Chains

A *discrete-time Markov chain* (DTMC) consists of states that represent instantaneous snapshots of the system at a given time, and has transitions labelled by (discrete) probability distributions over the target states.

Definition 1. *A DTMC is a tuple $\mathcal{D} = (S, s^\iota, T, \mathrm{AP}, L)$, where S is a finite set of states, s^ι is the initial state, $T : S \times S \to [0, 1]$ is a transition probability function such that $\sum_{s' \in S} T(s, s') = 1$ for all $s \in S$, AP is a set of atomic propositions, and $L : S \to 2^{\mathrm{AP}}$ is a valuation function which assigns to every state $s \in S$ a set $L(s)$ of atomic propositions that are assumed to be true at that state.*

Observe that each transition represents the possibility to evolve from one state to another. Moreover, for a state $s \in S$ of $\mathcal{D}$, the probability of moving to a state $s' \in S$ in one discrete step is given by $T(s, s')$. Further, a *path* of $\mathcal{D}$ is

an infinite sequence $\omega = s_0, s_1, s_2, \ldots$ of states such that $T(s_i, s_{i+1}) > 0$ for all $i \geq 0$. Each path of $\mathcal{D}$ provides one possible evolution of the Markov chain.

Properties of DTMCs can be written in *Probabilistic Computation Tree Logic* (PCTL) [3], a probabilistic extension of the temporal logic CTL [1].

Definition 2 (Syntax). *Let $a \in \mathrm{AP}$ be an atomic proposition, $p \in [0,1]$ a probability bound, $k \in \mathbb{N}$ and $\sim \in \{<, \leq, \geq, >\}$. The syntax of PCTL is defined inductively as follows:*

$$\phi ::= \mathit{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{P}_{\sim p}[\psi], \quad \psi ::= \mathtt{X}\phi \mid \phi\,\mathtt{U}\phi \mid \phi\,\mathtt{U}^{\leq k}\phi$$

PCTL formulae are interpreted over the states of a DTMC or *Markov decision processes* (see next section). We say that a state $s \in S$ satisfies a PCTL formula ϕ, denoted $\mathcal{D}, s \models \phi$, if ϕ is true at the state s. Intuitively, a state s satisfies the basic state formula $\mathtt{P}_{\sim p}[\psi]$ if the probability of taking a path from s satisfying *path formula* ψ meets the bound $\sim p$. Further, the path formula $\mathtt{X}\phi$ (operator *neXt*) is true, if ϕ is satisfied in the next state; the path formula $\phi_1 \mathtt{U} \phi_2$ (operator *until*) is true, if ϕ_2 is eventually satisfied and ϕ_1 is true until then; the path formula $\phi_1 \mathtt{U}^{\leq k} \phi_2$ (operator *bounded until*) is true, if ϕ_2 is satisfied within k discrete steps and ϕ_1 is true until then.

In practice, it is common to write formulae of the following kind: $\mathtt{P}_{=?}[\psi]$, which asks "*what is the probability of ψ to be true*". Also, the following useful operators can be derived from the above PCTL syntax: $\mathtt{F}\phi ::= \mathit{true}\,\mathtt{U}\,\phi$ (*eventually* ϕ becomes true) and $\mathtt{G}\phi ::= \neg\mathtt{F}\neg\phi$ (ϕ is true *globally*), and a bounded variants of these.

2.2 Markov decision processes

A *Markov decision process* (MDP), like DTMC, consists of states, representing possible configurations of the system being modelled, and transitions between states occur in discrete time-steps. However, at each state the system (decision maker) may choose any action that is available in this state, and then non–deterministically move into a new state, while providing the decision maker a corresponding probability.

Definition 3. *An MDP is a tuple $\mathcal{M} = (S, s^\iota, \mathit{Act}, \mu, \mathrm{AP}, L)$, where:*

- *S, s^ι, AP and $L : S \to 2^{\mathrm{AP}}$ are defined as for DTMCs,*
- *Act is a finite set of actions,*
- *$\mu : S \times \mathit{Act} \to \mathit{Dist}(s)$ is the (partial) transition probability function, with $\mathit{Dist}(S)$ denoting the set of all discrete probability distributions over S.*

Observe that for each state $s \in S$, the successor state is determined in two stages: firstly, an available action $a \in \mathit{Act}$ (i.e. one for which $\mu(s,a)$ is defined) is non–deterministically selected; secondly, the successor is randomly chosen according to the probability distribution $\mu(s,a)$.

To reason formally about the behaviour of MDPs, normally, the notation of *policies* is used. A *policy* resolves all of the non–deterministic choices in an MDP.

Moreover, under the control of a particular policy, the behaviour of an MDP is fully probabilistic and, as is for DTMCs, one can define a probability space over the possible paths through the model. Further, it is possible to reason about the best– or worst–case system behaviour by quantifying over all possible policies: for example, it is possible to compute the minimum or maximum probability of a PCTL property. Finally, the notion of an *optimal policy* can be used with a property value optimised by a model checker. For example, `Prism` [5] can find an optimal policy with regard to the minimum or maximum possible probability of a PCTL property [6].

Properties of MDPs can also be written in PCTL, yet with an implicit quantification over policies. For example, the $\mathtt{P}_{=?}$ operator used for DTMCs is replaced with two variants $\mathtt{P}_{\min=?}$ (the minimum probability) and $\mathtt{P}_{\max=?}$ (the maximum probability).

3 Approximate extrapolation of policy

Let $\mathcal{M} = (S, s^\iota, Act, \mu, \mathrm{AP}, L)$ be an MDP. We first define an MDP *with classified binary choices* (MDPCBC) as follows.

Definition 4. *An MDPCBC is a tuple $\mathcal{X} = (S, s^\iota, Act, \mu, \mathrm{AP}, L)$, where:*

- *S, s^ι,* AP *and $L : S \to 2^{\mathrm{AP}}$ are defined as for MDPs,*
- *$Act = \bigcup_{i=1}^{\alpha} Act_i$ is a finite set of actions that is divided into α disjoint classes Act_i according to their meaning as understood in the modelled phenomenon. Moreover, each class Act_i is divided into two disjoint sets of the same size: $Act_i{\downarrow}$ and $Act_i{\uparrow}$.*
- *$\mu : S \times Act \to Dist(s)$ is the (partial) transition probability function, with $Dist(S)$ denoting the set of all discrete probability distributions over S, and the following property: any non–deterministic binary choice q contains exactly two actions $a_q{\downarrow}$ and $a_q{\uparrow}$ such that $a_q{\downarrow} \in Act_i{\downarrow}$ and $a_q{\uparrow} \in Act_i{\uparrow}$.*

For example, if $Act = Act_1 \cup Act_2$, then Act_1 might represent choices of either a black or a white ball and Act_2 might represent a decisions if to continue a loop of choosing the balls or, on the contrary, stop the process. Further, if a non–deterministic binary choice represents a class of actions "a choice of either a black or a white ball", then $Act_1{\downarrow}$ would contain only choices of the black ball and $Act_1{\uparrow}$ would contain only choices of the white ball.

Now we define the notation of a decision boundary.

Definition 5. *Let $\mathcal{X} = (S, s^\iota, Act, \mu, \mathrm{AP}, L)$ be an MDPCBC, and let the binary non–deterministic choices between actions belonging to Act_i be available at certain states $S_i \subseteq S$. A* decision boundary *is a function $D_i : S_i \to \{false, true\}$ such that if there is a non–deterministic choice between actions $a_q{\downarrow}$ and $a_q{\uparrow}$, then the agent chooses $a_q{\downarrow}$ if $D_i(s) = false$, and $a_q{\uparrow}$ otherwise.*

Thus, a decision boundary determines a certain policy controlling an MDPCBC, which then becomes a DTMC, further called a *Markov chain with classified binary choices* (MCCBC).

As seen, thanks to the division of *Act* into classes, we have a number of class–specific decision boundaries, which individually may be easier to describe mathematically. On the contrary, mixing actions with vastly different meanings might produce a common decision boundary which is hard to analyse.

3.1 Method

Let there be a set $\mathscr{X} = \{X^1_{\text{small}}, \ldots, X^J_{\text{small}}, X_{\text{large}}\}$ of MDPCBCs, instantiated from a common template, but with different values of the parameter N, equal respectively to $\{N^1_{\text{small}}, \ldots, N^J_{\text{small}}, N_{\text{large}}\}$. The parameter does not influence on the nature of the problem, but merely represents a scale of the problem. N can be e.g. the number of philosophers in the Dining Philosophers Problem [2]. We want to estimate an optimal policy of X_{large}. Yet it is impossible to find X_{large} directly using PMC (e.g. implemented in `Prism`), due to extensive computational complexity and memory requirements. Therefore, we will attempt to extrapolate J optimal policies of J respective X^j_{small}, $j = 1, \ldots, J$.

Let the decision boundary for class i in X^j_{small} be $D^j_i(s)$, $s \in S_i$, and let us pose a hypothesis on how $D^j_i(s)$ can be merged into a single function. The hypothesis is represented by $D_i(s, N)$, which generalises all $D^j_i(s)$ so that $D^j_i(s) = D_i(s, N^j_{\text{small}})$, $j = 1, \ldots, J$. This allows for obtaining a hypothetical extrapolated decision boundary $D_i(s, N_{\text{large}})$, controlling X_{large}. Finally, we strengthen or refute the posed hypothesis, by locally verifying the fitness of $D_i(s, N_{\text{large}})$ using SPMC.

3.2 Arbitrariness

Let any state in S be represented by a tuple $V = (x_1, \ldots, x_d)$, a so–called state vector, each of its elements represents some specific phenomenon in a modelled system. For example V might have an interpretation (temperature, precipitation). Let us build a real coordinate metric space $\mathscr{S}$ such that any state s is mapped to a point V_s in $\mathscr{S}$, having coordinates V. We do that because we hope that points close in $\mathscr{S}$ may intuitively represent similar situations in the modelled system, thus it is less likely that a decision boundary goes between them. This hopefully simplifies both the shape and semantic interpretation of $D_i(s, N)$.

Due to, amongst others, a finite $||S||$, the extrapolation to $D_i(s, N_{\text{large}})$ might be imprecise. Consider the following. See that $||V_s|| = ||S||$ is finite, and thus we can find some $\epsilon > 0$ which is equal to the closest distance between all possible pairs (V_{s_1}, V_{s_2}), $s_1, s_2 \in S, s_1 \neq s_2$. Therefore, there is an infinite number of decision boundaries representing a single policy. Let any decision boundary be represented by a vector of parameters $C(N) = (y^N_1, \ldots, y^N_\beta)$, where $\beta \in \mathbb{N}^+$ is a constant specific to a method of the representation. For any class i, we can extrapolate $D_i(s, N)$ by a respective extrapolation of J vectors $C_i(N^j_{\text{small}})$ to a single vector $C_i(N_{\text{large}})$, the latter determining $D_i(s, N_{\text{large}})$. Yet, as $C(N)$ is arbitrary due to $\epsilon > 0$, so is $C_i(N_{\text{large}})$. Moreover, as the extrapolation may

augment the arbitrariness, the multiple possible values of $C_i(N_{\text{large}})$ may in turn represent multiple $D_i(s, N_{\text{large}})$.

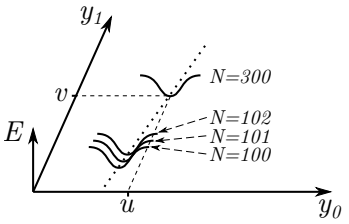


Fig. 1. A schematic example of a trajectory of $C_1(N) = (y_0, y_1)$ in a parameterised Euclidean space $\mathbb{R}^2$. Schematically depicted minimum error E of a decision maker translates to an optimal policy of an MDPCBC instantiated with a given N.

Consider the example in Fig. 1(a). $C_1(100)$, $C_1(101)$ and $C_1(102)$ determine a common segment. We extrapolate that segment with a line in order to find $C_1(300) = (u, v)$, which in turn determines hypothetical $D_1(s, 300)$. Yet, as MDPCBCs have a finite number of states, $C_1(300)$ can not be determined precisely: minor arbitrariness in the placement of the segment scale up roughly affinely with the distance to the segment. We thus see, that there is at least a single reason for $D_i(s, N_{\text{large}})$ to be a hit–and–miss when it comes to an estimation of a strategy for X_{large}.

4 Case study

Let us study an example – an optimal policy of choosing a coin. There are two coins, a fair one and an unbalanced one. They have the probabilities of the outcome of heads equal to respectively 0.5 and 0.6. A decision maker flips a coin N times, deciding before each flip, which of the two coins to choose (for simplicity, N is even). What is the best policy of maximising the probability of drawing $N/2$ heads within these flips? The criterion of optimality is thus

$$P_{\text{opt}} = \mathtt{P}_{\text{max=?}}[\mathtt{F}(f = N \wedge h = \frac{N}{2})] \tag{1}$$

Let N be the scaling parameter discussed in Sec. 3.

Let us first verify a small variant of the model, with $N = N_{\text{small}} = 100$, using `Prism`'s PMC capabilities. `Prism` is able to compute the optimal policy of a decision maker which has a full knowledge about the system, and the computed policy is given as a transition matrix of a DTMC, which in the case of our model is also an MCCBC. The policy is visualised in Fig. 2(a), where f and h are parts of the vector state which are equal to, respectively, the number of tosses so far and the number of heads drawn so far.

Let $Act_1\uparrow$ be a choice of the fair coin. The decision boundary, approximated by visually interpreting Fig. 2(a), is

$$h \gtrsim 0.55\,(f - (9 \pm 0.5)) \tag{2}$$

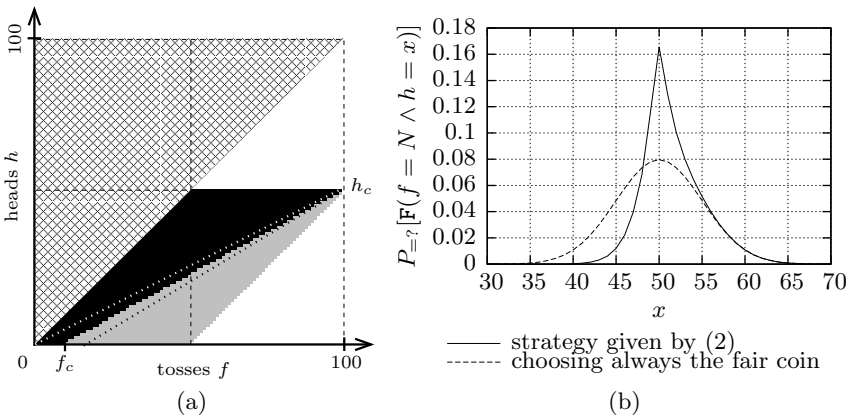


Fig. 2. (a) Visualisation of the optimal policy for $N = N_{\text{small}} = 100$. Gridded region depicts unreachable states, white region designates, that a successful policy is no more possible or a maximum number of tosses is reached, black and grey regions mean respectively "chose the fair or the unbalanced coin". White dashed line represents $h = f/2$, black dashed line shows an example variation of the decision boundary for a different f_c. (b) Probability distributions of h for two different strategies at $f = N = N_{\text{small}}$.

Fig. 2(b) depicts probability distributions of h after all N_{small} tosses. As seen, (2) makes (1) almost twice as large if compared to a state–agnostic approach of always choosing the fair coin.

4.1 A single small MDPCBC

Firstly, we will attempt to extrapolate from only (2), i.e. let $J = 1$. Assuming limited computational resources for finding optimal policies of small models, $J = 1$ enables us to use the largest possible N^J_{small}.

Extrapolation. In order to extrapolate from a single point, we will form some supporting hypotheses. It is easy to see that at the state $(f = 0, h = 0)$ the decision maker should choose the fair coin for any N. This is because he is more afraid of an excessively large h, rather than of the number of heads drawn being too small, as the single available unbalanced coin leans towards heads, and thus it can be used to reduce the deficiency of h. The latter also says, that a high deficiency of h leads to the choice of the unbalanced coin. Therefore, we know that at some $(f \geq f_c, h = 0)$, $f_c > 0$, the unbalanced coin is chosen, and that for any N, the fair coin is chosen for $(0 \leq f < f_c, h = 0)$. We also know that at $(f = N - 1, h = N/2 - 1)$ the decision maker should maximise the probability of drawing a head in order to reach $(f = N, h = N/2)$ (the success is assured only if a head will be drawn, an unbalanced coin is chosen), and that he would minimise that probability for $(f = N - 1, h = N/2)$ (the success is assured only if a head will not be drawn, a fair coin is thus chosen). Therefore, we know that for any N the decision boundary goes between these two states.

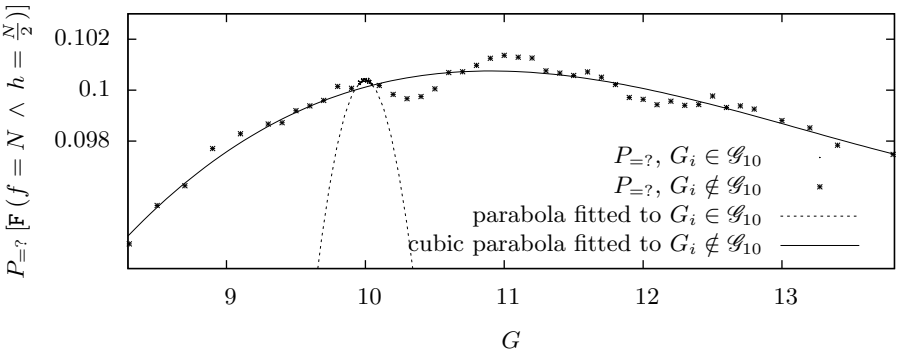


Fig. 3. Estimation of P_{opt} for $N = N_{\text{large}}$, $5 \cdot 10^7$ samples per MCCBC.

On the basis of (2) we can guess that the decision boundary is a segment. Let us guess that the placement of the segment scales affinely with N in the sense that $f_c \approx N/G$, where G is a constant. This hypothesis is trivially represented by the following decision boundary:

$$h > h_1(f) = \left(\frac{N}{2} - 1/2\right) \frac{f - f_c}{N - 1 - f_c} = \left(\frac{N}{2} - 1/2\right) \frac{f - N/G}{N - 1 - N/G}$$

thus

$$D_i(s, N) = \begin{cases} G < \dfrac{N}{N-1} \dfrac{2h + 1 - N}{2h - f} & \text{if } h < f/2 \\ \mathit{true} & \text{if } h \geq f/2 \end{cases} \tag{3}$$

Verification. Using (2) we can estimate $G = N^1_{\text{small}}/f_c \approx 100/(9 \pm 0.5) \underset{\sim}{\in} \mathscr{G} = \langle 10.5; 11.8 \rangle$. Checking statistically P_{opt} in an MDPCBC controlled by (3) for $N = N_{\text{large}} = 5000$ and $G \approx 11 \in \mathscr{G}$ yields the diagram in Fig. 3. An MCCBC with the highest P_{opt} is found for $G = G_1 = 11$, which agrees with the estimation, and we may thus strengthen the hypothesis.

Local maxima are seen in the diagram. For example, we statistically checked MCCBCs for a dense set of values of G in a set $\mathscr{G}_{10}$ such that $\forall_{G_i \in \mathscr{G}_{10}} 0.95 \leq G_i \leq 10.05, ||\mathscr{G}_{10}|| = 51$, then we fitted a parabola as seen in the figure, to show that a local gradient–descent optimiser might be trapped in a local maximum around $G \approx 10$, thus strengthening a respective false hypothesis.

We thus see, that an interval of G wider than the one spanned over $\mathscr{G}_{10}$ should be sampled by such an optimiser. This leads to a considerable numerical complexity of the resulting SPMC, as we need to use as much as $5 \cdot 10^7$ samples per a single estimation in order to get a 99% confidence level of $\approx 2.3 \cdot 10^{-4}$, estimated using Asymptotic Confidence Interval [10].

A less precise extrapolation might increase the said complexity even more. For example,

– a less precise initial estimation of G might result in a larger number of samples to be gathered, in order to localise the maximum around $G = 11$;
– if we would merely assume, that the decision boundary is a segment, whose both ends are unknown and with no known relation, and not that only f_c is unknown as we did so far, we would then need to search within at least a two–dimensional optimisation space. For example one with the optimised parameters (G, H) such that the linear boundary of $D_1(N_{\text{large}})$ intersects a pair of points $(f_c = N_{\text{large}}/G, 0)$, $(N_{\text{large}} - 1, H)$.

Obviously, in general, an MDPCBC with $N = N_{\text{large}}$ might become unverifiable using both PMC and SPMC, if the extrapolation from PMC–checked models were insufficiently precise or unknown at all.

4.2 Several small MDPCBCs

We will attempt to estimate $D_1(N_{\text{large}})$ using several small MDPCBCs. We won't use the reasoning from Sec. 4.1, but instead, an optimiser will analyse a trajectory of parameters representing different decision boundaries.

Extrapolation. We apply a Nelder–Mead Simplex gradient–descent optimiser [9] to a linear combination of f, h, N and 1, with the goal of minimising the number of wrong choices in models with $N = 80, 100$ and 120, i.e. $J = 3$. We obtain a hypothetical generalised decision boundary

$$550.876 f - 1001.63\, h - 50.1596\, N - 15.0067 \lesssim 0$$

thus for $N = N_{\text{large}}$

$$h \gtrsim h_2(f) = 0.549980(f - 455.298), \quad G \approx G_2 = 10.982 \tag{4}$$

Testing (4) against the three MCCBC matrices returned by `Prism`, it turns out that this boundary always allows for a right choice within the three MDPCBCs in question. It may be a hint, that the said linear combination has been a right choice.

Verification. Due a finite number of states in the MDPCBCs from which we have extrapolated, we may expect imprecisions in (4) as discussed in Sec. 3, especially that we extrapolate from $N^j_{\text{small}} \approx 100$ to $N_{\text{large}} \approx 5000$, i.e. to a model with a number of tosses about 50 times as large on average.

We know from the reasoning in Sec. 4.1, that $N/2 - 1 \leq h(N-1) < N/2$. For $N = N_{\text{large}}$, $h_2(N_{\text{large}} - 1) \approx 2498.95$, and is thus too small by ≈ 0.05, a difference which seems to be fairly precise for an extrapolation that distant. Yet, we will correct that imprecision by placing the segment correctly at $f = N_{\text{large}} - 1$, and then extracting from (4) merely the value of G_2. This boils down to the reuse of the diagram in Fig. 3. Given the low number of statistically verified MCCBCs in the diagram and the considerable size of the 99% confidence interval, given in Sec. 4.1, it can be stated that $G_2 \approx G_1$. We may thus strengthen the hypothesis.

5 Discussion

We are working on a tool for automating the presented extrapolation. As opposed to the example in the case study, it would apply a number of extrapolating functions beside a linear combination, in order to choose the best extrapolated policy. For example, the tool could support an extrapolation of oscillating functions like in [11], in order to deal with models of physical systems involving periodicity. For example, the scaling parameter might represent a rotational speed of an element, and the policy would minimise a standing wave in the supporting construction.

References

1. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT press (1999)
2. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Informatica **1**(2), 115–138 (1971)
3. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal aspects of computing **6**(5), 512–535 (1994)
4. Kwiatkowska, M., Norman, G., Parker, D.: Advances and challenges of probabilistic model checking. In: Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on, pp. 1691–1698. IEEE (2010)
5. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 585–591. Springer (2011)
6. Kwiatkowska, M., Parker, D.: Advances in probabilistic model checking. In: T. Nipkow, O. Grumberg, B. Hauptmann (eds.) Software Safety and Security - Tools for Analysis and Verification, *NATO Science for Peace and Security Series - D: Information and Communication Security*, vol. 33, pp. 126–151. IOS Press (2012)
7. Kwiatkowska, M., Parker, D.: Automated verification and strategy synthesis for probabilistic systems. In: D.V. Hung, M. Ogawa (eds.) Proc. 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13), *LNCS*, vol. 8172, pp. 5–22. Springer (2013)
8. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: International Conference on Runtime Verification, pp. 122–135. Springer (2010)
9. Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. The Computer Journal **7**(4), 308–313 (1965)
10. Nimal, V.: Statistical Approaches for Probabilistic Model Checking. Master's thesis, Oxford University (2010)
11. Rataj, A.: Fractional genetic programming for a more gradual evolution. In: Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, pp. 371–382 (2013)
12. Sondik, E.J.: The optimal control of partially observable markov processes. Tech. rep., DTIC Document (1971)
13. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer (STTT) **8**(3), 216–228 (2006)

A GPGPU–based Simulator for Prism: Statistical Verification of Results of PMC [extended abstract]

Marcin Copik[1] and Artur Rataj[2] and Bożena Woźna-Szcześniak[3]

[1] RWTH Aachen University,
Schinkelstr. 2, 52062 Aachen, Germany
mcopik@gmail.com
[2] IITiS, Polish Academy of Sciences
ul. Bałtycka 5, 44-100 Gliwice, Poland
arturrataj@gmail.com
[3] IMCS, Jan Długosz University
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
b.wozna@ajd.czest.pl

Abstract. We describe a GPGPU–based Monte Carlo simulator integrated with Prism. It supports Markov chains with discrete or continuous time and a subset of properties expressible in PCTL, CSL and their variants extended with rewards. The simulator allows an automated statistical verification of results obtained using Prism's formal methods.
Keywords: GPGPU, Monte Carlo simulation, Prism, probabilistic model checking, statistical model checking, probabilistic logics.

1 Introduction

We present a GPGPU–based simulator which extends the model checker Prism [9]. The simulator uses the Monte Carlo method for a statistical probabilistic model checking [14, 10] (SPMC). SPMC involves a generation of a large number of random paths (i.e. samples) in a probabilistic Markov chain, evaluating a given property on each path, and finally finding an average of these evaluations, which approximate a correct value of the property. Monte Carlo methods typically are able to precisely compute confidence intervals (CI) around the approximated value.

The GPGPU simulator (further called $\mathcal{S}^G$) is integrated with Prism, which allows to check a single model implementation using either one of Prism's probabilistic model checking (PMC) methods, or $\mathcal{S}^G$. $\mathcal{S}^G$ supports the same models and properties as the Prism's CPU–based simulator (further denoted $\mathcal{S}^C$), yet the latter, lacking GPGPU acceleration and on–the–fly compilation of the model, is considerably slower.

Beside the simulator itself, we present its simple application: an automated method of verifying property values computed using Prism's formal methods. Namely, the user may request, that certain property classes be computed in two steps:

1. A PMC step. A property is computed using one of Prism 's formal PMC methods;
2. An automated statistical verification (ASV) step. The obtained property value v is statistically evaluated using $\mathcal{S}^G$; it is then checked if v fits into a number of confidence intervals (CI) of various confidence levels.

Because a significant imprecision in a PMC method is typically caused by some mechanism for reducing computational complexity, the user might tend to disable such a mechanism, rather than wait for a statistical verification. This is why it is crucial that the verifying simulator be fast: it should effectively save user's time, by providing data in tight CI within a short time.

The paper is constructed as follows. Section 2 describes what is currently supported by $\mathcal{S}^G$. In Sec. 3 we provide a description of some implementation details of $\mathcal{S}^G$. In Sec. 4 ASV is discussed Section 5 presents a case study. Finally, the last section concludes the paper.

2 Supported models and properties

$\mathcal{S}^G$ accepts a specific set of properties, for two finite probabilistic classes of Markov chains: a *discrete-time Markov chain* (DTMC) and a *continuous-time Markov chain* (CTMC). Unlike DTMC, where each transition corresponds to a discrete time-step, in a CTMC transitions occur in continuous time given by a negative exponential distribution. Both of the classes can be enriched with rewards structures, resulting respectively in rDTMC and rCTMC. A reward structure allows to specify two distinct types of rewards: state (instantaneous) and transition (cumulative) ones, assigned respectively to states and transitions by means of a reward function. Formal definitions of all of the above systems can be found e.g. in [8].

The temporal logics *Probabilistic Computation Tree Logic* (PCTL) [6] and *Continuous Stochastic Logic* (CSL) [1] can be used to specify properties for respectively DTMCs and CTMCs. $\mathcal{S}^G$ recognises only flat subsets of each logic. We will refer to these subsets as respectively FlatPCTL and FlatCSL.

Definition 1 (Syntax of FlatPCTL). *Let $a \in \mathcal{AP}$ be an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ a probability bound, and k is a non–negative integer or ∞. The syntax of FlatPCTL is defined inductively as follows:*

$$\phi ::= \mathtt{P}_{\sim p}[\psi], \quad \psi ::= \mathtt{X}\phi_1 \mid \mathtt{G}^{\leq k}\phi_1 \mid \mathtt{F}^{\leq k}\phi_1 \mid \phi_1 \mathtt{U}^{\leq k}\phi_1 \mid \phi_1 \mathtt{R}^{\leq k}\phi_1,$$
$$\phi_1 ::= a \mid \phi_1 \wedge \phi_1 \mid \neg\phi_1.$$

In the syntax above, we distinguish between state formulae ϕ, ϕ_1 and path formulae ψ, which are evaluated over states and paths, respectively. A property of a model is always expressed as a state formula. The path modalities (i.e., *next state* – $\mathtt{X}$, *bounded globally* – $\mathtt{G}$, *bounded eventually* – $\mathtt{F}^{\leq k}$, *bounded until* – $\mathtt{U}^{\leq k}$, and *bounded release* – $\mathtt{R}^{\leq k}$), which are standard in temporal logics, can occur only within the scope of the *probabilistic operator* $\mathtt{P}_{\sim p}[\cdot]$.

Intuitively, a state s satisfies $\mathtt{P}_{\sim p}[\psi]$ if the probability of taking a path from s satisfying path formula ψ meets the bound $\sim p$. Next, $\mathtt{X}\phi$ is true if ϕ is satisfied in the next state; $\mathtt{G}^{\leq k}\phi$ is true if ϕ holds for all time-steps that are less or equal to k; $\mathtt{F}^{\leq k}\phi$ is true if ϕ is satisfied within k time-steps; $\phi_1 \mathtt{U}^{\leq k}\phi_2$ is true if ϕ_2 is satisfied within k time-steps and ϕ_1 is true from now on until ϕ_2 becomes true. $\phi_1 \mathtt{R}^{\leq k}\phi_2$ is true if either ϕ_1 is satisfied within k time-steps and ϕ_2 is true from now on up to the point where ϕ_1 becomes true, or

ϕ_2 holds for all time-steps that are less or equal to k. The formal semantics over DTMC can be found e.g. in [6, 8].

$\mathcal{S}^G$ supports also an extension of FlatPCTL allowing specifications over reward structures by means of the following state formulae: $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}] \mid \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \mid \mathtt{R}_{\sim r}[\mathtt{F}\phi]$ where $\sim \in \{<, \leq, \geq, >\}$, $r \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$, and ϕ is a FlatPCTL formula.

The formal semantics over rDTMC can be found in [8]. Here we only provide an intuition. Namely, a state s of an rDTMC satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}]$, if from state s the expected reward *cumulated* after k time-steps satisfies $\sim r$. Next, a state s of an rDTMC satisfies $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$, if from state s the expected state reward at time-step k satisfies $\sim r$. Finally, a state s of an rDTMC satisfies $\mathtt{R}_{\sim r}[\mathtt{F}\phi]$, if from state s the expected reward cumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

Definition 2 (Syntax of FlatCSL). *Let a and p be as in Definition 1, and I be an interval of $\mathbb{R}_{\geq 0}$. The syntax of FlatCSL is defined inductively as follows:*

$$\phi ::= \mathtt{P}_{\sim p}[\psi], \quad \psi ::= \mathtt{X}\phi_1 \mid \mathtt{G}^I\phi_1 \mid \mathtt{F}^I\phi_1 \mid \phi_1 \mathtt{U}^I \phi_1 \mid \phi_1 \mathtt{R}^I \phi_1, \quad \phi_1 ::= a \mid \phi_1 \wedge \phi_1 \mid \neg\phi_1.$$

Satisfying $\mathtt{P}_{\sim p}[\psi]$ and path modalities are the same for FlatCSL as for FlatPCTL, except that the parameter of the modalities is an interval I of the non-negative reals, rather than an integer upper bound. For example, the path formula $\phi_1 \mathtt{U}^I \phi_2$ holds if ϕ_2 is satisfied at some time instant in the interval I and always earlier ϕ_1 holds.

$\mathcal{S}^G$ supports an extension of FlatCSL allowing specifications over reward structures in a manner similar to FlatPCTL, the only difference are the time bounds: $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq t}] \mid \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] \mid \mathtt{R}_{\sim r}[\mathtt{F}\phi]$ where $\sim \in \{<, \leq, \geq, >\}$, $r,t \in \mathbb{R}_{\geq 0}$, and ϕ is a FlatCSL formula.

The extension of CTMC with rewards is analogous to that of DTMC, given above, barring the mentioned differences in time bounds. The formal semantics over rCTMC can be found in [8], see though that $\mathcal{S}^G$does not support therein mentioned steady state.

3 Implementation of $\mathcal{S}^G$

In a case of Markov models implemented in the `Prism` language, a generation of random simulation paths is not computationally expensive. A single transition consists of an evaluation of its guards, an enumeration of updates if viable, a random selection of subsequent transitions and finally an estimation of properties. The syntax of guards and updates allows simple arithmetical and logical operations and a few basic mathematical functions, such as a power or a logarithm. `Prism` comes already with the mentioned CPU–based simulator $\mathcal{S}^C$, well–suited to debugging tasks but slow, as it is sequential and generates each path by reinterpreting a model specification.

Instead of such an on–the–fly reinterpretation, the tool Ymer [15] compiles expressions to a form which is faster to evaluate repeatedly. Another tool APMC [7], in, turn provides a translation of `Prism` models to C programs which are later compiled and executed.

Another obstacle preventing the $\mathcal{S}^C$ from achieving a reasonable performance is its inherent sequentiality. A Monte Carlo simulation is considered to be embarrassingly parallel – it samples the model by generating a large number of independent random

paths. Prism has approached this problem by providing the ability to perform distributed sampling, Ymer and APMC support distributed sampling as well. The latest version of Ymer implement multi–threaded sampling as well [16].

The improvement in parallel and distributed sampling is limited by the number of threads supported on multi–core CPU systems. A processor with a rich set of instructions and multiple cache levels is a perfect tool for complex and general problems but using it for a simple simulation of a moderate Markov model would be a very expensive over–engineering, both financial cost– and energy–wise. On the contrary, recent advances in GPGPUs made them a very efficient replacement for such computations, which benefit from massive parallelism. This simultaneous execution of hundreds and thousands lightweight threads on a GPGPU comes at a price: well-known limitations include a restriction of a group of threads to execute the same instruction at the same time or a burdensome memory model with a high cost of non–regular memory access patterns. We believe though, that those restrictions do not play a significant role in a Monte Carlo simulation of Prism models. For that purpose we have chosen OpenCL [13] as a framework and programming language for GPGPU simulation.

3.1 Architecture

Our decision to implement the simulator engine for Prism using OpenCL has been driven by its capability of supporting many types of devices offered by different vendors, the most common being graphic cards and multi–core CPU servers. We will further use an OpenCL–specific terminology.

To simplify the implementation we have used OpenCL bindings for Java, the main source language of Prism, provided through the JavaCL library [2].

Fig. 1 presents a general scheme of the simulator. Within Prism there is no significant difference between starting a simulation in using $\mathcal{S}^C$ or $\mathcal{S}^G$. In both cases a model and a list of properties is required. Then, a just–in–time source-to-source translation of the model and properties produces a dedicated *compute kernel* (block Kernel generator in Fig. 1) which is basically a program for a number of vector processors, which use a single–instruction–multiple–data paradigm. The approach is very different from the reinterpretation scheme implemented in $\mathcal{S}^C$, which stores the model and properties in memory structures, and not as an automatically generated program. $\mathcal{S}^G$ executes kernels implemented in OpenCL C language, a modified version of C99, which has been adapted to OpenCL's device abstraction and stripped from features usually not allowed on a device, such as recursion or function pointers. Those simple programs can be effectively compiled into native bytecode of the device (typically, a graphic card). The syntax of Prism language makes the translation process rather straightforward due to the similarity between its expressions and the OpenCL C language. Only minor and automatically applied changes allow to obtain an expression valid in the latter language.

3.2 Method

A scheme of generating a single random path (sample) is presented in Algorithm 1. Capitalised identifiers indicate constants which are injected into kernel source. Many

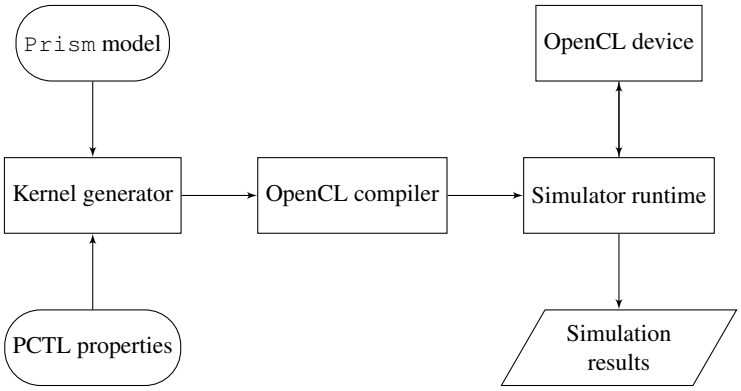


Fig. 1. The OpenCL-based simulator engine in `Prism`.

details have been omitted in the case of continuous time models. For example, in CTMC both times of entering and leaving state may be required in certain situations, such as a bounded until or a property with cumulative reward. The fargument `idx` is an unique identifier of kernel instance, `offset` specifies how many paths have been processed in kernels previously computed on the device. The argument `seed` seeds the generator of pseudorandom numbers. As the scheme is an equivalent of an OpenCL kernel, the two last arguments are OpenCL–specific storage buffers in the device memory, where the kernel is allowed to save results of property verification, and also the length of created path used to display sampling statistics. The first loop (lines 5–7) resets the collected statistical data about properties, the second loop (lines 8–27) generates the path until any of the following: its maximum length is reached (line 8), a deadlock (line 14) or a self–loop (line 18) occurs, or all properties are verified precisely enough (line 24). The last loop (lines 29–31) copies the collected data into the global memory.

The implementation had to be specially adapted for GPGPU devices, given that there can be a significant slow–down if the kernel diverges from the SIMD paradigm. Another example of such change is choosing always the smallest, most space efficient integer type for holding a state variable, which is possible as `Prism` models specify ranges for each such variable. For efficiency, if a simulated model updates a variable with a value exceeding these ranges, then the behaviour is undefined. A large speed–up can be achieved by handling an update synchronised between `Prism` modules in one step. If such update is performed on the same copy of state vector, it may induce a race condition of the type Read-After-Write. $\mathcal{S}^G$ detects such situations and creates additional copies of the affected variables, if necessary, instead of relying on the OpenCL compiler, which might handle the issue less effectively. Creating only one instance of a state vector is crucial for performance because it decreases significantly memory space used by the kernel, as discussed later with memory complexity of the algorithm. The simulation kernel is also capable of detecting when there is only one transition available, and it does not change the state. Such a behaviour indicates that there is only a single self–loop in the current state, which is interpreted by $\mathcal{S}^G$ as a stop condition. If there is an unbounded property which has not been satisfied yet, its value is not going

to change. If there is a property with a lower bound, which has not been reached yet, it can be evaluated immediately and the process of simulating the current path ends.

Our pseudorandom number generator of choice is Random123 [12], which provides a performance satisfying our needs. For more details on model conversion into a form for GPGPU computation see [3].

Algorithm 1 A generic path generation algorithm for OpenCL kernel.

```
1:  procedure KERNEL(idx, offset, seed, path_lengths, property_results)
2:      prng ← initialize_prng(seed, idx, offset)          ▷ A distinct seed for each path
3:      state_vector ← INITIAL_STATE_VECTOR
4:      time ← 0
5:      for each property p ∈ PROPERTIES do
6:          reset(results_p)
7:      end for
8:      for i < MAX_PATH_LENGTH do
9:          active_updates ← evaluate_guards(state_vector)      ▷ Single and synchronised
10:         time_update(time)                                   ▷ More complex for CTMC
11:         for each property p ∈ PROPERTIES do
12:             active_properties ←property_p_update(state_vector, results_p, time)
13:         end for
14:         if active_updates = 0 then
15:             break                                           ▷ Deadlock detected
16:         end if
17:         no_change ← update(prng, state_vector, active_updates)
18:         if no_change ∧ active_updates = 1 then
19:             for each property p ∈ PROPERTIES do
20:                 active_properties ←property_p_update(state_vector, results_p, time)
21:             end for
22:             break                                           ▷ Loop detected
23:         end if
24:         if ¬active_properties then
25:             break                                           ▷ Stop sampling
26:         end if
27:     end for
28:     path_lengths[idx + offset] = i                      ▷ Save results in global memory
29:     for each property ∈ property_results do
30:         property[idx + offset] = results_p              ▷ Save results in global memory
31:     end for
32: end procedure
```

A generated kernel is passed as a string of source code to a specific device compiler (block `OpenCL compiler` in Fig. 1). This compilation does not add a significant overhead on modern OpenCL platforms – we have found that it typically takes less than one second for tested models. A kernel represented in device bytecode is sent to simulator's runtime (see again Fig. 1), where a range of *work items* is enqueued on the device, as described in more details in the next section. Each one of them is responsible

for producing exactly one random path through the model and its identifier *idx*, is paired with an *offset*, in order to produce a unique key across many separate kernel enqueued on the device. The key is necessary for correct accessing memory storage and unique random seeds for each generated path.

4 Automatic statistical verification

The ASV step is optionally triggered at the user request, in one of the following ways:

- unconditionally;
- if a quantitative property is being computed, like the probability value;
- if a steady state is detected prematurely in an iterative PMC method.

The last criterion is discussed in more detail in the example in Sec. 5.

The ability to process an extensive number of samples in a very short time often allows for a reliable and fast ASV of a property value v obtained using PMC. In the ASV step, in order to save the user's time, $\mathcal{S}^G$ must finish within a predefined time $T_{\max}$.

After the ASV step, the user is presented with a set of diagnostics, so that he can estimate the correctness of v. Let the simulator estimate a value w of the same property. Let R_{CI} be the ratio of the width of CI at confidence level 90% to w. The following independent diagnostics can be presented:

- $T_{\max}$ too low to reach $R_{CI} < R^{CI}_{\max}$; $R^{CI}_{\max}$ has a default value of $1 \cdot 10^{-2}$ and can be customised;
- v within a CI of a confidence level x, outside a CI of a confidence level y, where $x \in L = \{90\%, 95\%, 99\%\}$, $y \in L \vee \{< 90\%\}$.

5 Case study

An instantaneous reward in a CTMC can be estimated by weighting the reward function over a probabilistic distribution of states at a time instant t. `Prism` computes the distribution by uniformisation (Jensen's method) [5] which discretises the CTMC with respect to a constant rate. Then, probabilities are approximated by a finite summation Z of Poisson–distributed steps of the derived DTMC. The number of these steps depends on the precision required, and is computed using the Fox–Glynn method [4]. Yet, in order to shorten computation time, when performing that summation, `Prism` also tests, if a steady state has been reached, by finding a maximum difference, either relative or absolute, between elements in solution vectors from two successive steps. If the difference is smaller than a constant threshold ε, the summation is terminated early.

`Prism`'s default criterion of the termination is to use a relative difference and $\varepsilon = 10^{-6}$. The criterion can be customised in order to set a compromise between precision and computational complexity.

To illustrate the ASV, we will discuss a model where the detection of a steady state is premature if the default termination criterion is used. In effect, were the automatic SV not enabled, the user would obtain incorrect results without a warning.

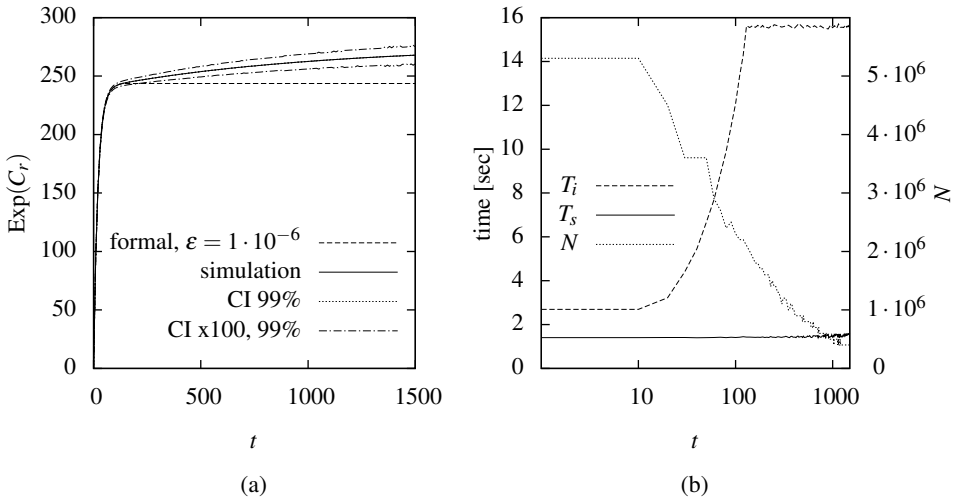


Fig. 2. (a) Estimates of $\mathrm{Exp}(C_r)$, CIs are narrow enough to be hardly seen, thus their magnification by 100 is also show. (b) elapsed CPU time and the number of samples per single property, T_i and T_s are total times of, respectively, the formal method performing Z, and the statistical estimation on $\mathcal{S}^G$, N is the number of samples chosen by $\mathcal{S}^G$ in order to fit within $T_{\max}$.

We will use a modified Model 2 from [11]. It is a simple CTMC with multiple clients and servers, in which some of the servers can occasionally be broken. To trigger the said premature termination of Z, we modify the model by using constants $r_s = 1, r_l = 500$ (see [11] for details), i.e. the server is slower at initiating a connection with a client, but faster at processing a request. We ask for an expected instantaneous reward value C_r, equal to the number of clients requesting at a given time instant t.

Let the user choose the default termination criterion, and let he also request, that SV be used for up to $T_{\max} = 1.5$ sec. after a PMC step such that a steady state detection has terminated early Z (or any other formal iterative method which uses ε). $\mathrm{Exp}(C_r)$ against t, found using both a formal PMC and $\mathcal{S}^G$, is depicted in Fig. 2(a). We see that the model undergoes a fast change at $t \approx 100$, during which the increase of requests becomes rapidly slower. The constant $T_{\max}$ allows for a fairly narrow CI at 99% confidence level – its width never reaches 0.1. In the case of the discussed diagram $R_{CI} < 4 \cdot 10^{-4}$ for any t – such a narrow CI makes it probable that following the said change at $t \approx 100$, the PMC results become less and less precise. This would trig respective warnings, that values from the results of the PMC step fall outside a CI of a high confidence level.

The relative temporal overhead of SV for the chosen $T_{\max}$ is illustrated in Fig. 2(b). It is largest for small t and makes `Prism` run for about 50% longer. For $t \gtrsim 200$ `Prism` needs less than 10% of an additional CPU time to perform the SV step. The figure also shows the number of samples which can be computed within $T_{\max}$; wee see that the number decreases for larger t due to longer paths which need to be generated by the simulator.

In order to obtain the CPU times in Fig. 2(b), we have used an AMD R9 Nano, a modern mid-range GPU with 4096 stream processors. Let us compare that GPU to

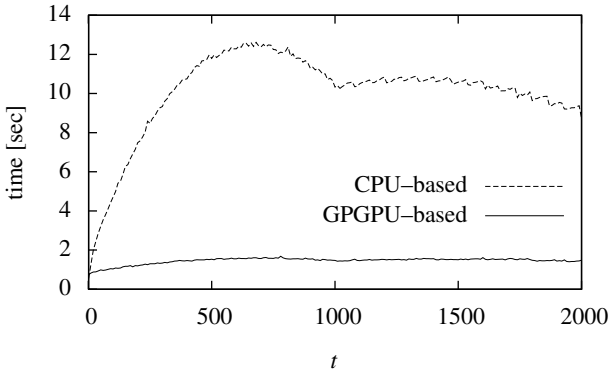


Fig. 3. Computation time on two OpenCL devices.

another OpenCL device, containing two Intel Xeon E5-2630 v3 CPUs with clock frequency 2.40GHz, each of them providing 8 cores with HyperThreading, resulting in 32 threads for OpenCL. Fig. 3 compares the simulation time from Fig. 2(b) to that of the CPU OpenCL device, both evaluating the same number of samples. In the case of the latter device, we see times in the range of 8 and 12 seconds, i.e. it is several times slower. This shows the advantage of streams processors in the case of a large number of independent, non–memory intensive tasks.

$\mathcal{S}^C$ has been excluded from the chart as it is not optimised for speed. At $t = 2000$, where long paths make it especially advantageous to use the on–the–fly compilation, it took $\mathcal{S}^C$ over 130 minutes to evaluate the same reward property on the same CPU, thousands of times slower comparing a respective simulation on a GPGPU.

6 Discussion

The SV limits itself now to diagnostics, but it could be straightforwardly extended to influence on the PMC step. For example, if a property p_i turns out to be computed imprecisely in the PMC step, and the following property to compute p_{i+1} differs only in the time instant t, the SV could automatically decrease ε for the computation of p_{i+1}.

We expect to release an open-source version of the tool in the following months. The further development would be focused on a parallelisation across multiple OpenCL devices, with a dynamic and automatic load balancing.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. Transactions on Computational Logic (TOCL) **1**(1), 162–170 (2000)
2. Chafik, O.: Javacl. `https://github.com/nativelibs4java/JavaCL` (2016)
3. Copik, M.: GPU-accelerated stochastic simulator engine for PRISM model checker. Bachelor's Thesis, Silesian University of Technology, Gliwice, Poland (2014)

4. Fox, B.L., Glynn, P.W.: Computing poisson probabilities. Commun. ACM **31**(4), 440–445 (1988)
5. Gross, D., Miller, D.: The randomization technique as a modeling tool and solution procedure for transient Markov processes. Operations Research **32**(2), 926–944 (1984)
6. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal aspects of computing **6**(5), 512–535 (1994)
7. Herault, T., Lassaigne, R., Peyronnet, S.: Apmc 3.0: Approximate verification of discrete and continuous time markov chains. In: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, QEST '06, pp. 129–130. IEEE Computer Society, Washington, DC, USA (2006). DOI 10.1109/QEST.2006.5. URL http://dx.doi.org/10.1109/QEST.2006.5
8. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07), *LNCS*, vol. 4486, pp. 220–270. Springer (2007)
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 585–591. Springer (2011)
10. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: International Conference on Runtime Verification, pp. 122–135. Springer (2010)
11. Pourranjbar, A., Hillston, J., Bortolussi, L.: Don't just go with the flow: Cautionary tales of fluid flow approximation. In: Computer Performance Engineering - 9th European Workshop, pp. 156–171 (2012)
12. Salmon, J.K., Moraes, M.A., Dror, R.O., Shaw, D.E.: Parallel random numbers: As easy as 1, 2, 3. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 16:1–16:12. ACM, New York, NY, USA (2011). DOI 10.1145/2063384.2063405. URL http://doi.acm.org/10.1145/2063384.2063405
13. Stone, J.E., Gohara, D., Shi, G.: Opencl: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering **12**(1-3), 66–73 (2010)
14. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer (STTT) **8**(3), 216–228 (2006)
15. Younes, H.L.S.: Ymer: A statistical model checker. In: Proceedings of the 17th International Conference on Computer Aided Verification, CAV'05, pp. 429–433. Springer-Verlag, Berlin, Heidelberg (2005). DOI 10.1007/11513988_43. URL http://dx.doi.org/10.1007/11513988_43
16. Younes, H.L.S.: Ymer. https://github.com/hlsyounes/ymer (2016)

Process Environment Opacity

Damas P. GRUSKA

Institute of Informatics, Comenius University,
Mlynska dolina, 842 48 Bratislava, Slovakia,
gruska@fmph.uniba.sk.

Abstract. A security property called *process environment opacity* is formalized and studied. Properties of process's inputs (environments) are expressed by predicate and it is required that an intruder cannot learn their validity by observing process's behavior. This basic idea can be formulated in various ways. The resulting security properties are compared.

Keywords: security, opacity, process algebras, information flow

1 Introduction

Information flow based security properties (see [GM82]) assume an absence of any information flow between private and public systems activities. This means, that 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 and among them opacity (see [BKR04,BKMR06]) could be considered as the most general one and many other security properties could 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 the both traces are indistinguishable for an observer (which is expressed by an observation function). This means that the observer (intruder) cannot be sure whether a trace for which the predicate holds has been performed or not. In [Gru15] opacity is modified (the resulting property is called process opacity) in such a way that instead of process's traces we focus on properties of reachable states. Hence we assume an intruder who is not primarily interested whether some sequence of actions performed by a given process has some given property but we consider an intruder who wants to discover whether this process reaches a state which satisfied some given (classified) predicate. It turns out that in this way we could capture some new security flaws. Both opacity and process opacity are based on fixed (static) security policy which is not changed during system computation. In [Gru16] process opacity for dynamic security policies is formalized. All of these properties are defined in

the framework of process algebras, but neither opacity, which represents security of process's traces nor process opacity, which represents security of resulting states, can capture security of process's inputs, similarly as security of inputs is defined for programming languages.

The aim of this paper is to formalize security of process's inputs. We adopt the approach which appeared in [SS15] for programming languages but we reformulate it for timed process algebras. Inputs will be modeled by processes which we call process's environments. The resulting security property will be called *process environment opacity*. It assumes that for every process's environment for which a given predicate holds there exists equivalent one, for which it does not hold and the process with these environments leads to equivalent states. This means that an intruder cannot learn validity of the predicate over process's inputs. We will study this security property as well as its variants. Moreover, we compare them with other security properties known in the literature.

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 some notion on information flow security based on opacity and in the next section process environment opacity is defined, studied and compared to other security properties. Section 5 contains discussion and plans for future work.

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" (i.e. for example, the process $a.Nil$ can perform t action since it is not contained the process specification). But in both cases internal communications have priority to action t in the parallel composition. 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 determinancy* 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 $Act = A \cup \{\tau\}, At = A \cup \{t\}, Actt = Act \cup \{t\}$. 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

$$\begin{aligned}
&\Sigma_0 = \{Nil\}\\
&\Sigma_1 = \{x. \mid x \in A \cup \{t\}\} \cup \{[S] \mid S \text{ is a relabeling function}\}\\
&\qquad \cup \{\backslash M \mid M \subseteq A\}\\
&\Sigma_2 = \{|, +\}
\end{aligned}$$

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 subterm $t.A$ i.e. between any two t actions only finitely many non timed actions can be performed) are called TPA processes.

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 $\to$ is a subset of TPA $\times$ $Actt$ $\times$ TPA. We write $P \xrightarrow{x} P'$ instead of $(P, x, P') \in \to$ 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:

$$\frac{}{Nil \xrightarrow{t} Nil}\ A1 \qquad \frac{}{u.P \xrightarrow{t} u.P}\ A2$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \not\xrightarrow{\tau}}{P \mid Q \xrightarrow{t} P' \mid Q'}\ Pa \qquad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'}\ S$$

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) i.e. performing of t action does not lead to the choice between summands of $+$. 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].

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. The set of all traces of P will be denoted by $Tr(P)$. By ϵ we will denote the empty sequence of actions, by $Succ(P)$ we will denote the set of all successors of P i.e. $Succ(P) = \{P' | P \xrightarrow{s} P', s \in Actt^*\}$. If set $Succ(P)$ is finite we say that P is finite state process. We define modified transitions $\stackrel{x}{\Rightarrow}_M$ which "hide" actions from M. Formally, we will write $P \stackrel{x}{\Rightarrow}_M P'$ for $M \subseteq Actt$ iff $P \xrightarrow{s_1}\xrightarrow{x}\xrightarrow{s_2} P'$ for $s_1, s_2 \in M^\star$ and $P \stackrel{s}{\Rightarrow}_M$ instead of $P \stackrel{x_1}{\Rightarrow}_M\stackrel{x_2}{\Rightarrow}_M \ldots \stackrel{x_n}{\Rightarrow}_M$. 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 $x \in M$. Note that $\stackrel{x}{\Rightarrow}_M$ is defined for arbitrary action but in definitions of security properties we will use it for actions (or sequence of actions) not belonging to M. We can the extend the definition of $\Rightarrow_M$ for sequences of actions similarly to $\xrightarrow{s}$. By $Sort(P)$ we will denote the set of actions from A which can be performed by P. The set of weak timed traces of process P is defined as $Tr_w(P) = \{s \in (A \cup \{t\})^\star | \exists P'.P \stackrel{s}{\Rightarrow}_{\{\tau\}} P'\}$. Two process P and Q are weakly timed trace equivalent ($P \simeq_w Q$) iff $Tr_w(P) = Tr_w(Q)$. We conclude this section with definitions M-bisimulation and weak timed trace equivalence.

Definition 1. *Let $(TPA, Actt, \rightarrow)$ be a labelled transition system (LTS). A relation $\Re \subseteq TPA \times TPA$ is called a* 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 M-bisimulation relating P and Q.*

3 Information flow and opacity

In this section we will present motivations for new security concepts which will be introduced in the next section. First we define an absence-of-information-flow property called Strong Nondeterministic Non-Interference

(SNNI, for short, see [FGM00]). Suppose that all actions are divided into two groups, namely public (low level) actions L and private (high level) actions H. It is assumed that $L \cup H = A$. Process P has SNNI property (we will write $P \in SNNI$) if $P \setminus H$ behaves like P for which all high level actions are hidden (by action τ) 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 $a \in M \cup \bar{M}$. Formally, we say that P has SNNI property, and we write $P \in SNNI$ iff $P \setminus H \simeq_w P/H$. SNNI property assumes an intruder who tries to learn whether a private action was performed by a given process while (s)he can observe only public ones. If this cannot be done then the process has SNNI property. A generalization of this concept is given by opacity (this concept was exploited in [BKR04], [BKMR06] and [Gru07] in a framework of Petri Nets, transition systems and process algebras, respectively. Actions are not divided into public and private ones at the system description level, as it is done for example in [GM04], but a more general concept of observations and predicates is exploited. 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 the both traces are indistinguishable for an observer (which is expressed by an observation function). This means that the observer (intruder) cannot say whether a trace for which the predicate holds has been performed or not.

First we assume an observation function i.e. a function $\mathcal{O} : Actt^\star \to \Theta^\star$, where Θ is a set of elements called observables (note that we have no other requirements on $\mathcal{O}$ except that it has to be total, i.e. defined for every sequence of actions). 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 a predicate ϕ over sequences. 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 system of interest. The observer cannot deduce the validity of ϕ if there are two sequences $w, w' \in Actt^\star$ such that $\phi(w), \neg\phi(w')$ and the sequences cannot be distinguished by the observer i.e. $obs(w) = obs(w')$. We formalize this concept by the notion of opacity.

Definition 2 (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 $Op(\phi, \mathcal{O})$.

$$P \stackrel{w}{\Longrightarrow}$$

$$\mathcal{O}(w) = \mathcal{O}(w') \text{ and } \phi(w), \neg\phi(w')$$

$$P \stackrel{w'}{\Longrightarrow}$$

Fig. 1. Opacity

Now let us assume that an intruder tries to discover, instead of a property over process's traces, whether a given process can reach a state with some given property expressed by a (total) predicate. This might be process deadlock, capability to execute only traces s with time length less then n, capability to perform at the same time actions form a given set, incapacity to idle (to perform t action) etc. Again we do not put any restriction on such predicates but we only assume that they are consistent with some suitable behavioral equivalence. The formal definition follows.

Definition 3. *We say that the predicate ϕ over processes is consistent with respect to relation $\cong$ if whenever $P \cong P'$ then $\phi(P) \Leftrightarrow \phi(P')$.*

As consistency relation $\cong$ we could take bisimulation ($\approx_\emptyset$), weak bisimulation ($\approx_{\{\tau\}}$) or any other suitable equivalence. A special class of such predicates are such ones (denoted as $\phi^Q_\cong$) which are defined by a given process Q and equivalence relation $\cong$ i.e. $\phi^Q_\cong(P)$ holds iff $P \cong Q$.

We suppose that the intruder can observe only some activities performed by the process. Hence we suppose that there is a set of public actions which can be observed and a set of hidden (not necessarily private) actions. To model observations we exploit the relation $\stackrel{s}{\Rightarrow}_M$. The formal definition of process opacity (see [Gru15]) is the following.

Definition 4 (Process Opacity). *Given process P, a predicate ϕ over processes is process opaque w.r.t. the set M if whenever $P \stackrel{s}{\Rightarrow}_M P'$ for $s \in (Actt \setminus M)^*$ and $\phi(P')$ holds then there exists P'' such that $P \stackrel{s}{\Rightarrow}_M P''$ and $\neg\phi(P'')$ holds. The set of processes for which the predicate ϕ is process opaque w.r.t. to the M will be denoted by $POp(\phi, M)$.*

Note that if $P \cong P'$ then $P \in PPOp(\phi, M) \Leftrightarrow P' \in POp(\phi, M)$ whenever ϕ is consistent with respect to $\cong$ and $\cong$ is such that it a subset of the trace equivalence (defined as $\simeq_w$ but instead of $\stackrel{s}{\Rightarrow}_{\{\tau\}}$ we use $\stackrel{s}{\Rightarrow}_\emptyset$).

$$P \quad \stackrel{s}{\Longrightarrow}_M \quad \phi(P')$$

$$P \quad \stackrel{s}{\Longrightarrow}_M \quad \neg\phi(P'')$$

Fig. 2. Process opacity

Properties opacity and process opacity are depicted in Fig. 1 and 2, respectively. In the former case, a set of actions performed by a process is of interest, say classified, in the later case, it is a property of reachable states which is protected by process opacity.

4 Process environment opacity

Now we will formulate security properties which protect inputs of a given process. We start with property called *process environment opacity* which is a modification of the the property formulated for programming languages which appeared in [SS15]. Process's inputs will be modeled also by processes, called environments, running in parallel with the process. Moreover, we force (by restriction operator) the process to communicate exclusively with its environment by means of channels contained in given set M. Formally, we assume a set of environments $\mathcal{E}$, i.e. processes which represent possible input environments for a given process. We extend definition of predicates over environments to sets of environments $\mathcal{K}, \mathcal{K} \subseteq \mathcal{E}$ in the following way. $\phi(\mathcal{K})$ holds iff $\phi(E)$ holds for every $E \in \mathcal{K}$. Moreover, we assume two equivalence relations over processes, $\cong^1, \cong^2$, which express capability of an intruder to distinguish among processes.

Definition 5 (Process Environment Opacity). *Given process P and set of environments $\mathcal{E}$, a predicate ϕ over processes is environment opaque w.r.t. the set M and $\cong^1, \cong^2$ if whenever $(P|E) \setminus M \stackrel{s}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds then there exists E' such that $(P|E') \setminus M \stackrel{s}{\Rightarrow} R'$, $E \cong^1 E'$, $\neg\phi(E')$ holds and $R \cong^2 R'$. The set of processes for which the predicate ϕ is environment opaque w.r.t. to the M and $\cong^1, \cong^2$ will be denoted by $PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$.*

Note that process environment opacity says nothing about security of inputs if $\phi(\mathcal{E})$ or $\neg\phi(\mathcal{E})$ holds. As regards choice of relations $\cong^1, \cong^2$ it depends on capability of possible intruders. For example, it could be bisimulation, trace equivalence or just the relation equal to $TPA \times TPA$. Schematically, process environment opacity is depicted in Fig. 3.

$$
\begin{array}{lcl}
(P|E) \setminus M \stackrel{s}{\Longrightarrow} & R, \texttt{ and } \phi(E) & E \\
 & \cong^2 & \cong^1 \\
(P|E') \setminus M \stackrel{s}{\Longrightarrow} & R' \texttt{ and } \neg\phi(E') & E'
\end{array}
$$

Fig. 3. Process environment opacity

4.1 Properties of process environment opacity

In this subsection we will formulate some properties of process environment opacity.

Proposition 1. *Let* $\phi_1 \Rightarrow \phi_2$. *Then*

$$PEOp(\phi_2, \mathcal{E}, M, \cong^1, \cong^2) \subseteq PEOp(\phi_1, \mathcal{E}, M, \cong^1, \cong^2).$$

Proof. Let $P \in PEOp(\phi_2, \mathcal{E}, M, \cong^1, \cong^2)$ and $(P|E) \setminus M \stackrel{s}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi_1(P')$ holds. Then also $\phi_2(P')$ holds since $\phi_1 \Rightarrow \phi_2$. We know that there exists E' such that $(P|E') \setminus M \stackrel{s}{\Rightarrow} R'$, $E \cong^1 E'$ and $\neg\phi_2(E')$ holds and $R \cong^2 R'$. Since $\neg\phi_2 \Rightarrow \neg\phi_1$ we have that also $\neg\phi_1(E')$ holds and hence $P \in PEOp(\phi_1, \mathcal{E}, M, \cong^1, \cong^2)$. □

Proposition 2. *Let* $\mathcal{E}_1 \subseteq \mathcal{E}_2$ *and* $\neg\phi(\mathcal{E}_2 \setminus \mathcal{E}_1)$. *Then*

$$PEOp(\phi, \mathcal{E}_1, M, \cong^1, \cong^2) \subseteq PEOp(\phi, \mathcal{E}_2, M, \cong^1, \cong^2).$$

Proof. Sketch. Let $P \in PEOp(\phi, \mathcal{E}_1, M, \cong^1, \cong^2)$ and $(P|E) \setminus M \stackrel{s}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds for $E \in \mathcal{E}_1$. Since $\neg\phi(\mathcal{E}_2 \setminus \mathcal{E}_1)$ all environments for which ϕ holds are from $\mathcal{E}_1$. We know that there exists $E' \in \mathcal{E}_1$, $E \cong^1 E'$, such that $(P|E') \setminus M \stackrel{s}{\Rightarrow} R'$ and $\neg\phi(E')$ holds and $R \cong^2 R'$. Since $\mathcal{E}_1 \subseteq \mathcal{E}_2$ we have that also $P \in PEOp(\phi, \mathcal{E}_2, M, \cong^1, \cong^2)$. □

Proposition 3. *Let* $\cong^1 \subseteq \cong^{1'}$. *Then*

$$PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2) \subseteq PEOp(\phi, \mathcal{E}, M, \cong^{1'}, \cong^2).$$

Proof. Let $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$ and $(P|E) \setminus M \stackrel{s}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds for $E \in \mathcal{E}$. We now that there exists E' $E' \in \mathcal{E}$ such that $(P|E') \setminus M \stackrel{s}{\Rightarrow} R'$, $E \cong^1 E'$ and $\neg\phi(E')$ holds and $R \cong^2 R'$. Since $\cong^1 \subseteq \cong^{1'}$ we have that also $E \cong^{1'} E'$ and hence we have also $P \in PEOp(\phi, \mathcal{E}, M, \cong^{1'}, \cong^2)$. □

Proposition 4. *Let* $\cong^2 \subseteq \cong^{2'}$. *Then*

$$PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2) \subseteq PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^{2'}).$$

Proof. Let $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$ and $(P|E) \setminus M \stackrel{s}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds for $E \in \mathcal{E}$. We now that there exists E' $E' \in \mathcal{E}$ such that $(P|E') \setminus M \stackrel{s}{\Rightarrow} R'$, $E \cong^1 E'$ and $\neg\phi(E')$ holds and $R \cong^2 R'$. Since $\cong^2 \subseteq \cong^{2'}$ we have that also $R \cong^{2'} R'$ and hence we have also $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^{2'})$. □

As regards compositionality properties of process environment opacity, only a few of them hold. In general, process environment opacity is not compositional even for the prefix operator. Let $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$ for ϕ such that $\phi(A)$ holds iff $A \stackrel{\bar{x}}{\rightarrow}$ and $x \in M$. Clearly, $(x.P|A') \setminus M$ cannot perform any action for A' such that $\neg\phi(A')$, i.e. $x.P \notin PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$. A bit more sophisticated argument leads to the same result also for $x \notin M$. On the other side, process environment opacity is compositional with respect to non-deterministic choice as it is stated by the following Proposition.

Proposition 5. *Let* $P, Q \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$. *Then*

$$P + Q \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2).$$

Proof. Let $(P+Q|E)\setminus M \stackrel{s}{\Rightarrow}_M R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds. Then without loss of generality we can assume that $(P|E) \setminus M \stackrel{s}{\Rightarrow}_M R$. Since $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$ there exists E' such that $(P|E') \setminus M \stackrel{s}{\Rightarrow}_M R'$, $E \cong^1 E'$ and $\neg\phi(E')$ holds and $R \cong^2 R'$. Hence also $(P+Q|E') \setminus M \stackrel{s}{\Rightarrow}_M R'$ what implies $P + Q \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$.

□

As regards the rest of TPA operators, the situation is similar as for the prefix operator.

4.2 Process environment opacity and Non-Deducibility

In this subsection we will compare process environment opacity with Non-Deducibility on Composition (NDC for short, see in [FGM03]) which is a widely studied security property. It is based on the idea of checking the system against all high level potential interactions, representing every possible high level process. System is NDC if for every high level user A, the low level view of the behaviour of P is not modified (in terms of trace equivalence) by the presence of A. The idea of NDC can be formulated as follows.

Definition 6. (NDC) $P \in NDC$ *iff for every* $A, Sort(A) \subseteq H \cup \{\tau, t\}$

$$(P|A) \setminus H \simeq_w P \setminus H.$$

Now we will formulate relationship between NDC and $PEOp(\phi, \mathcal{E}, H, \cong^1, \cong^2)$ properties. Note that this relationship is similar to the one which appeared in [SS15] for programming languages.

Proposition 6. *$P \in NDC$ iff $P \in PEOp(\phi, \mathcal{E}, H, \cong^1, \cong^2)$ for $\mathcal{E} = \{E \in TPA | Sort(A) \subseteq H\}$, $\cong^2 = TPA \times TPA$ and for every ϕ such that every kernel of $\cong^1$ contains with every process E from $\mathcal{E}$ for which $\phi(E)$ holds also process E' for which $\neg\phi(E')$ holds.*

Proof. Sketch. Let $P \in NDC$ that means that $(P|E) \setminus H \simeq_w (P|E') \setminus H$ for every two E, E' such that $Sort(E) \subseteq H, Sort(E') \subseteq H$. But by the assumption we know that for every ϕ such that for every E for which $\phi(E)$ holds there exists E' for which $\neg\phi(E')$ holds and $E \cong^1 E'$, and this implies that $P \in PEOp(\phi, \mathcal{E}, H, \cong^1, \cong^2)$.

Now suppose that $P \notin NDC$ i.e. there exist E, E' such that $(P|E) \setminus H \not\simeq_w (P|E') \setminus H$. Without loss of generality we can assume that $(P|E) \setminus H \stackrel{s}{\Rightarrow}$ but $(P|E') \setminus H \stackrel{s}{\not\Rightarrow}$. Hence we can choose predicate ϕ such that $\phi(E)$ holds and only environment for which ϕ does not hold is equal to E'. Hence $P \notin PEOp(\phi, \mathcal{E}, H, \cong^1, \cong^2)$. □

4.3 Variants of process environment opacity

In this subsection we will formulate several variants of process environment opacity. We will start with its persistent variant.

Definition 7 (Persistent Process Environment Opacity). *Given process P and set of environments $\mathcal{E}$, a predicate ϕ over processes is environment persistently opaque w.r.t. the set M and $\cong^1, \cong^2$ if whenever for every $P', P' \in Succ(P)$ we have $P' \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$.*

The set of processes for which the predicate ϕ is persistent environment opaque w.r.t. to the M and $\cong^1, \cong^2$ will be denoted by $PPEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$.

The relationship between process environment opacity and its persistent variant is formulated in by the following Proposition.

Proposition 7. $PPEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2) \subseteq PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$

Proof. The proof follows directly from Definitions 5 and 7. In general, we cannot guarantee that this inclusion is proper. □

Process environment opacity expresses security of inputs for which given predicate holds but does not say anything about those ones for which it does not hold, i.e. says nothing about security of $\neg\phi$. So we define a stronger variant of process environment opacity.

Definition 8 (Strong Process Environment Opacity). *Given process P and set of environments $\mathcal{E}$, a predicate ϕ over processes is strongly environment persistently opaque w.r.t. the set M and $\cong^1, \cong^2$ if whenever $P \in PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$ and $P \in PEOp(\neg\phi, \mathcal{E}, M, \cong^1, \cong^2)$.*

The set of processes for which the predicate ϕ is strongly environment opaque w.r.t. to the M and $\cong^1, \cong^2$ will be denoted by $SPEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$.

Proposition 8. $SPEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2) \subseteq PEOp(\phi, \mathcal{E}, M, \cong^1, \cong^2)$

Proof. Again the proof follows directly from Definitions 5 and 8. and we cannot guarantee that the inclusion is proper as well. □

The most of properties stated for process environment opacity can be formulated also for its strong variant, some with a slight modification.

Till now we have not expected that environments are fully "consumed" by processes. For cases when an intruder can see only results of complete computations we have to modify definition of process environment opacity. Suppose that all process in $\mathcal{E}$ are finite and moreover, the last action performed before reaching (sub)state Nil is a new auxiliary action $\surd$. That means that for every environment E every occurrence of Nil is preceded by $\surd$ i.e. every Nil is always a subterm of a term $\surd.Nil$. For example, $E = a.b.c.\surd.Nil + b.a.\surd.Nil$ is such a process. We indicate corresponding environments by $\mathcal{E}_{\surd}$.

Definition 9 (Termination Process Environment Opacity). *Given process P and set of environments $\mathcal{E}_{\surd}$, a predicate ϕ over processes is environment opaque w.r.t. the set M and $\cong^1, \cong^2$ if whenever $(P|E)\backslash M \stackrel{s.\surd}{\Rightarrow} R$ for $s \in (Actt \setminus M)^+$ and $\phi(E)$ holds then there exists E' such that $(P|E') \setminus M \stackrel{s.\surd}{\Rightarrow} R'$, $E \cong^1 E'$ and $\neg\phi(E')$ holds and $R \cong^2 R'$. The set of processes for which the predicate ϕ is termination environment opaque w.r.t. to the M and $\cong^1, \cong^2$ will be denoted by $TPEOp(\phi, \mathcal{E}_{\surd}, M, \cong^1, \cong^2)$.*

As regards decidability of proposed security properties, in general they are undecidable. To obtain properties which are decidable, we need restrictions on predicates which should be decidable, limitation to finite set of environments and/or restrictions on equivalences $\cong^1, \cong^2$.

5 Discussion and further work

We have presented the new security concept called process environment opacity and we have formalized it in the timed process algebra framework.

It expresses different aspects of processes behaviour as opacity or process opacity. Instead of sequences of actions or resulting processes, it focuses on process's inputs and their properties. We have presented some properties of the resulting security property as well as its variants. Since we worked with timed process algebras we can model security with respect to limited time length of an attack, with a limited number of attempts to perform an attack and so on. Besides investigation of decidable variants of the proposed properties we also plan to study variants of process environment opacity which assume intruders which are not only observers but can actively interact with the systems to be attacked.

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.

[DHS15] van Delft B., S. Hunt, D. Sands: Very Static Enforcement of Dynamic Policies. In Principles of Security and Trust, LMCS 9036, 2015.

[FGM03] Focardi R., R. Gorrieri and F. Martinelli: Real-Time information flow analysis. IEEE Journal on Selected Areas in Communications 21 (2003).

[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 Programming, 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.

[GM04] Gorrieri R. and F. Martinelli: A simple framework for real-time cryptographic protocol analysis with compositional proof rules. Science of Computer Programming, Volume 50, Issues 13, 2004.

[Gro90] Groote, J. F.: Transition Systems Specification with Negative Premises. Proc. of CONCUR'90, Springer Verlag, Berlin, LNCS 458, 1990.

[Gru15] Gruska D.P.: Process Opacity for Timed Process Algebra. In Perspectives of System Informatics, LNCS 8974, 2015.

[Gru16] Gruska D.P.: Dynamic Security Policies and Process Opacity, In Proc of Perspectives of System Informatics, to appeare in LNCS, 2016.

[Gru10] Gruska D.P.: Process Algebra Contexts and Security Properties. Fundamenta Informaticae, vol. 102, Number 1, 2010.

[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.

[SS15] Schoepe D. and A. Sabelfeld: Understanding and Enforcing Opacity. Proc. of IEEE 28th Computer Security Foundations Symposium, 2015.

Coordinator Synthesis for Hierarchical Structure of Artificial Neural Network

Stanislaw Placzek

Engineering Department
Vistula University
Stoklosy Street, 02-787 Warsaw, Poland
stanislaw.placzek@wp.pl

Abstract. Two important issues have to be dealt with when implementing the hierarchical structure [1] of the learning algorithm of an Artificial Neural Network (ANN). The first one concerns the selection of the general coordination principle. Three different principles are described. They vary with regard to the degree of freedom for first-level tasks. The second issue concerns the coordinator structure or coordination algorithm. The ANN learning process can be examined as a two-level optimization problem. Importantly all problems and sub-problems are unstructured minimization tasks . The article concentrates on the issue of the coordinator structure. Using the interaction prediction principle as the most suitable principle for two-level ANN structures, different coordinator target functions are defined. Using classification task examples, the main dynamic characteristics of the learning process quality are shown and analyzed.

Keywords: Artificial Neural Network (ANN), hierarchy, decomposition, coordination, coordination principle, coordinator structure

1 Computational task complexity

Large-scale multidimensional classification, interpolation and extrapolation are complex tasks that can require long calculation times. For these tasks one can make use of the ANN learning process using input and output data vectors with a defined network architecture. There is no theoretical solution to the architecture selection process, including the definition of the number of hidden layers and neuron distribution between layers. From a calculation point of view, the ANN learning process approaches a local or a global minimum asymptotically and is very time-consuming. A multi-layered ANN with one input layer, a set of hidden layers and an output layer can be sectioned off [1]. Every layer has its own input and output vectors. For a standard two-layer network both the hidden layer and the output layer can be described as sub-networks. These sub-networks form the first-level of the hierarchical structure. So the network consists of two sub-networks, and the local target function for each of them is defined $\Phi = (\Phi_1, \Phi_2)$.

Similarly to the ANN structure decomposition, learning algorithm using error backpropagation can also be decomposed. It can be decomposed into:

- The first-level task, searching for the minimum of the local target functions $\Phi = (\Phi_1, \Phi_2)$,
- The second-level task, coordinating the first-level tasks.

Unfortunately, the first level optimization tasks in such learning algorithm are non-linear. In practice, only standard procedures exist to solve these optimization problems . But in a two-level learning algorithm structure, the coordinator is not responsible for solving the global task. In [2], the most popular interaction prediction principle was implemented. According to this principle, the coordinator plays an active role in the current ANN learning process. The main interaction prediction principle is shown in Fig. 1. With each iteration, the coor-

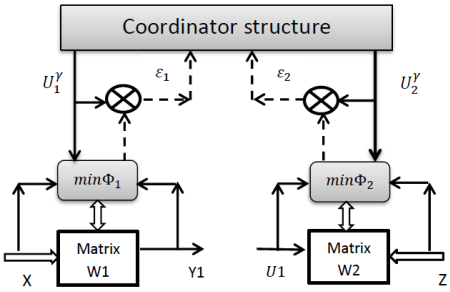


Fig. 1. Interaction Prediction Principle

dinator and all of the first-level sub-tasks interchange information. The first-level sub-tasks are optimal searching tasks. Usually, they look for the minimum of the Mean Squared Error (MSE). The coordination algorithm structure is primary related to the interaction prediction principle and two signals are used: primary discerning $U^\gamma = (U_1^\gamma, U_2^\gamma)$ and the feedback signal ascending $\epsilon = (\epsilon_1, \epsilon_2)$. The primary signals are known as coordination signals and are sent from the coordinator to all of the first-level sub-tasks. Thereby the coordinator assists in optimizing the first-level sub-tasks. The forecast coordination signals may not be precisely correct and the sub-tasks calculate their own value of the coordination signals. These signals are sent up into the coordinator. The coordinator then uses its own gradient method to calculate the new value of the coordination signals (improving upon its own previous estimate). The process can be continued until the coordinator and all the first-level sub-tasks have solved their own tasks.

1.1 Coordination aspects

Coordination as an operation process is related to three types of decision problems [3]. Finding a solution for:

– The global task as the primary task for the entire ANN structure,
– Local tasks as the decomposition's result of the global target function,
– Coordinator task that should be synthesized or find a solution procedure.

In effect , three different specific problems have to be solved:

1. Coordinator Synthesis. Decompose the global target function Φ into two sub-tasks with their own local target functions Φ_1, Φ_2; the upper-level coordinator task should be defined in such a way that all the sub-tasks are coordinated.
2. Modification Problem. In a two-level ANN learning algorithm; both the first-level tasks and the coordinator task are defined. Unfortunately, the ANN is not coordinated by the coordinator tasks. Before this problem can be dealt with, one should modify the first-level tasks in such a way that the coordinator coordinates the modified local target functions $\Phi = (\Phi_1, \Phi_2)$. This can be formalized using the following predicate formula:

$$(\exists\gamma)(\exists W)[\mathbf{P}(W, \Phi) and \mathbf{P}(U, \Psi(U^{\gamma}, \epsilon))] \tag{1}$$

   Where: The predicate $\mathbf{P}(W, \Phi)$ is true, if Φ is a problem (task) and W is one of its solution. $\Psi(U, \epsilon)$- coordinator target function. The first-level tasks are coordinated with the coordination task when the coordination task has a solution, all the first-level sub-tasks also have the solution for same coordination input U^{γ}.
3. Decomposition. Given only the global target task Φ for ANN - decompose this task Φ into the two sub-tasks Φ_1, Φ_2 and find a coordinator structure and a coordination procedure. This is formalized as:

$$(\exists\gamma)(\exists W)[\mathbf{P}[W = (W_1, W_2), (\Phi_1(U), \Phi_2(U) and \mathbf{P}(W, \Psi(X, Z, W))]] \tag{2}$$

   Where: X - input file, Z - learning file, $W = (W_1, W_2)$ - ANN's weight coefficients.

The first-level tasks are coordinated with a given global target task when the global task has a solution and as do some of the first-level tasks. The coordinator has to influence the first-level tasks in such a way that the resulting action guarantees the solution of the global target task.

1.2 Decomposition and coordination

Using Fig. 1, one can define the set of target functions:

– The global target function:

$$\Phi(W1, W2, Y, Z) = \frac{1}{2} \cdot \sum_{k=1}^{N_2} (y_k - z_k)^2 \tag{3}$$

  Where: $Y[1 : N_2]$ - the ANN output value, $Z[1 : N_2]$ - the the vector of teaching data, N_2 - the number of output neurons.

– The local target function Φ_1

$$\Phi_1(W1, X, U^\gamma) = \frac{1}{2}\sum_{i=1}^{N_1} f(\sum_{j=0}^{N_0} W1_{ij} \cdot x_j) - u_i^\gamma)^2 \tag{4}$$

Where: $U^\gamma[1 : N_1]$ - the coordination matrix as an input variable, N_1 - the number of hidden neurons, N_0 - the number of input neurons, $f(*)$ - a sigmoid function.

– The local target function Φ_2.

$$\Phi_2(W2, Z, U^\gamma) = \frac{1}{2}\sum_{k=1}^{N_2} f(\sum_{i=0}^{N_1} W2_{ki} \cdot u_i^\gamma) - z_k)^2 \tag{5}$$

Where: $U^\gamma[1 : N_1]$ - the coordination matrix as an input variable, N_2 - the number of output neurons, $f(*)$ - a sigmoid function

Using (4), one can calculate the feedback signal $\epsilon 1_i$ and the new value of matrix W1.

$$\epsilon 1_i = f(\sum_{j=0}^{N^0} W1_{ij}\dot{x}_j) \tag{6}$$

$$\frac{\partial \Phi 1}{\partial W1_{ij}} = (v1_i - u_i^\gamma) \cdot f' \cdot x_j \tag{7}$$

$$W1_{ij}(n+1) = W1_{ij}(n) - \alpha 1 \cdot \frac{\partial \Phi 1}{\partial W1_{ij}} \tag{8}$$

For the second sub-network using (5), one can calculate the new value of $\epsilon 2_i$ and the new value of matrix $W2_{ki}$.

$$\frac{\partial \Phi 2}{\partial W1_{ki}} = (v2_k - z_k) \cdot f' \cdot u_k^\gamma \tag{9}$$

$$W2_{ki}(n+1) = W2_{ki}(n) - \alpha 2 \cdot \frac{\partial \Phi 2}{\partial W1_{ki}} \tag{10}$$

$$\frac{\partial \Phi 2}{\partial u_i^\gamma} = \sum_{k=1}^{N_1}(v2_k - z_k) \cdot f' \cdot W2_{ki} \tag{11}$$

$$\epsilon 2_i(n+1) = u_i^\gamma(n) - \alpha 3 \cdot \frac{\partial \Phi 2}{\partial u_i^\gamma} \tag{12}$$

Where: $\alpha 1, \alpha 2, \alpha 3$ - learning coefficients, $u_i^\gamma = u1_i^\gamma = u2_i^\gamma$ - coordination signals, $\epsilon 1_i, \epsilon 2_i$ - feedback signals.

To summarize the first-level includes two sub-networks and two optimization tasks. The first sub-network calculates the new coefficient matrix $W1_{ik}(n + 1)$ and the feedback signal $\epsilon 1_i$ value by taking the parameter $u1_i^\gamma$ from the coordinator and using the optimization procedure. The feedback signal is sent

into the coordinator. For the second sub-network, the coordination signal $u2_i^\gamma$ sets the optimization procedure in motion and calculates the new coefficient matrix $W2_{ki}(n+1)$ and the feedback signal $\epsilon 2_i$ value. The feedback signal is also sent into the coordinator. After that, the coordinator procedure has to calculate the new coordinator signal $u1_i^\gamma(n+1)$ *and* $u2_i^\gamma(n+1)$. Thus, design of the coordinator structure is the main problem in a two-level learning algorithm.

2 Coordinator structure

In a two-level learning algorithm, the coordinator plays the main role. Therefore, the choice of the coordinator principle is paramount. The principle should specify strategies for the coordinator and determines the structure of the coordinator. Three different approaches to how the interaction could be performed were introduced [3].For an ANN learning algorithm, the Interaction Prediction Principle is the most suitable. According to it, the coordinator predicts the interface inputs. Success in coordinating all the first-level tasks depends on the accuracy of the prediction of the interface inputs or the effect of the prediction inputs. Therefore, to obtain responses from the first-level tasks, the coordinator could measure:

1. The interface inputs value. This principle will be known as the Interface Interaction Balance.
2. The target function value. This principle will be known as the Performance Prediction Principle.

2.1 Interface Interaction Balance

The coordination input may involve a prediction of the interface between the first and the second sub-networks. For the first sub-network, the coordinator signal U_1^γ is sent into the target function Φ_1 as a teacher data parameter. For the second sub-network, the coordinator signal U_2^γ is treated as an input variable. Thanks to it, both tasks are fully specified and algorithms can find the minimum value of their target functions Φ_1, Φ_2 . The Interface Interaction Balance idea is shown in Fig. 2. The coordinator target function Ψ could be a linear or a nonlinear

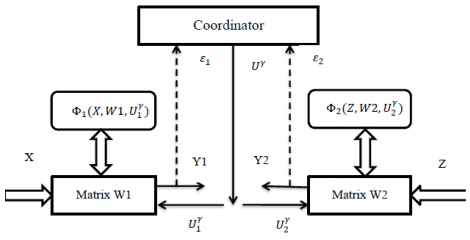


Fig. 2. Coordination algorithm structure for Interface Prediction Principle

function.

$$\Psi(U^\gamma, \epsilon^\gamma) = \Psi(U^\gamma, U^\gamma - Y) \tag{13}$$

Equation (13) uses the linear relation between two feedback signals $\epsilon 1, \epsilon 2$ and the prediction interface value U^γ. Where: U^γ- prediction interface value Y - real interface value

$$\Psi = \frac{1}{2}\sum_{i=1}^{N_1}(u_i^\gamma - \epsilon 1_i^\gamma)^2 + \frac{1}{2}\sum_{i=1}^{N_1}(u_i^\gamma - \epsilon 2_i^\gamma)^2 \tag{14}$$

Where: $U^\gamma = [u_1^\gamma, u_2^\gamma ... u_{N_1}^\gamma]$

Using formula (14), the first derivative is calculated

$$\frac{\partial \Psi}{\partial u_i^\gamma} = (u_i^\gamma - \epsilon 1_i^\gamma) + (u_i^\gamma - \epsilon 2_i^\gamma) \tag{15}$$

$$u_i^\gamma(n+1) = u_i^\gamma(n) - \lambda 1 \cdot \frac{\partial \Psi}{\partial u_i^\gamma} \tag{16}$$

Where: $\lambda 1$ - learning coefficient,

The coordinator and the first-level sub-networks work in an iterative scheme. When the coordinator signal $U^\gamma(n)$ is applied and the first-level optimization tasks find their own solution, a new coordination signal can be calculated (16). Using the Interface Interaction Balance, the two vectors signals are measured: the predicted interface input U^γ and the real interface value Y_1, Y_2 (Fig.2). In a real situation this requirement could be difficult to implement.It is usually possible to measure the first-level tasks and to send into the coordinator the target function (performance) Φ_1, Φ_2 value and their derivatives.

2.2 Linear Performance Balance Principle

The target functions values and their derivatives are of a more generalized form.The coordination scheme that uses this idea is shown in Fig. 3. The coor-

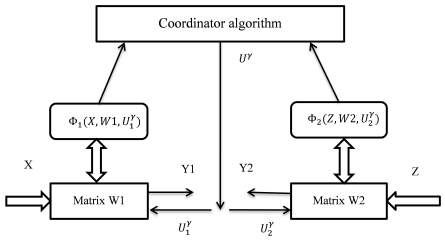


Fig. 3. Coordination algorithm structure for Performance Prediction Principle

dinator can receive both the target function value and the derivative value from

the first-level sub-systems . The coordinator function can be defined as a linear or a nonlinear function of the two parameters $\Phi 1, \Phi 2$

$$\Psi = \Psi(\Phi 1, \Phi 2) \tag{17}$$

Then the local target functions can be defined by :

$$\Phi 1(Y1, W1, U^{\gamma}) = \frac{1}{2}\sum_{i=1}^{N_1}(y1_i - u_i^{\gamma})^2 \tag{18}$$

$$\Phi 2(Y2, W2, Z) = \frac{1}{2}\sum_{k=1}^{N_2}(y2_k - z_k)^2 \tag{19}$$

In this subsection, we examine the coordinator function (17) as linear relation:

$$\Psi = \Phi 1 + \Phi 2 \tag{20}$$

Using (18) and (19) the first partial derivatives are calculated:

$$\frac{\partial \Phi 1}{\partial u_i^{\gamma}} = (y1_i - u_i^{\gamma}) \tag{21}$$

$$\frac{\partial \Phi 2}{\partial u_i^{\gamma}} = \sum_{k=1}^{N_2}(y2_k - z_k) \cdot f' \cdot W2_{ki} \tag{22}$$

Using the same gradient algorithm for the coordinator as above, the new coordination signal $U^{\gamma}(n+1)$ is calculated using the formula

$$u_i^{\gamma}(n+1) = u_i^{\gamma}(n) - \lambda 1 \cdot \frac{\partial \Psi}{\partial u_i^{\gamma}} \tag{23}$$

The global target function Ψ is a nonlinear function that should take into account the sigmoid activation functions for both the first and the second sub-network. Because of this, one can say that the coordination function described by formula (20) is simplistic and does not consider the non - linearity of the coordinator structure.

2.3 Nonlinear Performance Balance Principle

In the article, non - linearity will be demonstrated by two coordinator target functions

$$\Psi_m = \Phi 1 + \Phi 2 + c \cdot \Phi 1 \cdot \Phi 2 \tag{24}$$

$$\Psi_p = \Phi 1 + \Phi 2 + c \cdot (\Phi 1^2 + \Phi 2^2) \tag{25}$$

Where:
Ψ_m - indicates non-linear multiplication part,
Ψ_p - indicates non-linear sum of power part.

parameter "c" describes the impact of the non-linear part on the coordinator algorithm.

The structure of target functions Φ_1, Φ_2 are the same as in formula (18) and (19), respectively. The final formulas for derivatives have more complicated structures

$$\frac{\partial \Psi_m}{\partial u_i^\gamma} = \frac{\partial \Phi 1}{\partial u_i^\gamma} + \frac{\partial \Phi 2}{\partial u_i^\gamma} + c \cdot \left[\frac{\partial \Phi 1}{\partial u_i^\gamma} \cdot \Phi 2 + \frac{\partial \Phi 2}{\partial u_i^\gamma} \cdot \Phi 1 \right] \tag{26}$$

$$\frac{\partial \Psi_p}{\partial u_i^\gamma} = \frac{\partial \Phi 1}{\partial u_i^\gamma} + \frac{\partial \Phi 2}{\partial u_i^\gamma} + 2 \cdot c \cdot \left[\frac{\partial \Phi 1}{\partial u_i^\gamma} \cdot \Phi 1 + \frac{\partial \Phi 2}{\partial u_i^\gamma} \cdot \Phi 2 \right] \tag{27}$$

In practice, calculation developers are obliged to find the optimal "c" value. This parameter will have a significant impact on the quality and stability of the coordinator learning process.

3 Classification task example

The main dynamic characteristics of the learning process can be shown using the following example. Emphasis is put on the characteristics of the first-level local target functions, Φ_1, Φ_2, and the second level, coordinator target function Ψ . Optimal ANN learning characteristics depend on two connected tasks. Firstly, a network structure that includes a number of hidden layers needs to be created.Secondly, neurons need to be distributed between layers. A single hidden layer is chosen in this example. The structure of the ANN is simple and can be described as ANN $(N_0 - N_1 - N_2)$. In literature, one can only find suggestions regarding optimal numbers of neurons using the Vapnik - Cervonenkis dimension [4][5]. However, the hidden layer structure could also be determined using Kolmogorov's theorem [4][6]. For an ANN with one hidden layer, a sigmoid activation function can be chosen for classification tasks and N_0 input neurons,

$$VCdim = N_0 + 1 \tag{28}$$

Therefore, we can use this measure to define the number of neurons in the hidden layers

$$N_1 = VCdim \tag{29}$$

For a continuous function with N_0 input vector dimension and N_2 output vector, the number of neurons according to Kolmogorov's theorem can be calculated as

$$N_1 = 2 \cdot N_0 + 1 \tag{30}$$

In practice, the number of neurons in the hidden layer will be chosen according to the formula

$$N_0 + 1 \leq N_1 \leq 2 \cdot N_0 + 1 \tag{31}$$

Sigmoid activation functions are implemented in both the hidden and output layers. For the classification task using 6 - dimension input vectors $N_0 = 6$ and $N_2 = 1$, the number of hidden neurons is calculated according to formula

$$7 \leq N_1 \leq 13 \tag{32}$$

3.1 Example for Interface Interaction Balance

The quality of the leaning process depends on the key learning parameters for the first-level as well as the coordinator level. Fig. 4 shows the dynamic characteristics of the learning process.

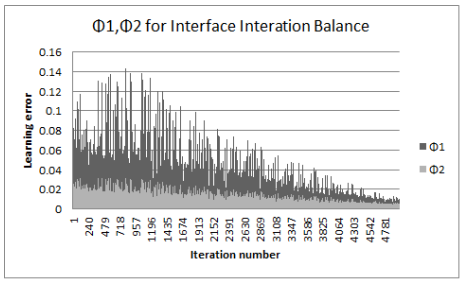


Fig. 4. The first sub-networks learning error. $\lambda_1 = 0.5$

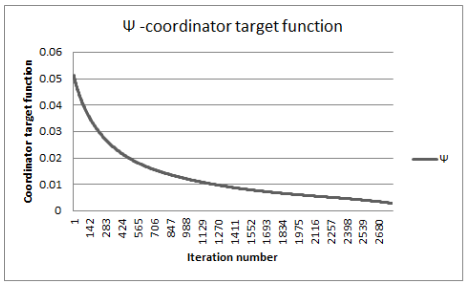


Fig. 5. The coordinator learning error. $\lambda = 0.2$

The main learning parameters, $\alpha 1 = 0.3, \alpha 2 = 0.3$, guarantee that every sub-task can find its own minimum target function value. The coordinator has its own algorithm described by equation (23). If parameter λ_1 is too large, the learning process is not stable, especially at the end of the iterative process.

The first sub-network includes 6 input neurons and 13 output neurons. Matrix W1 includes 7*13 = 91 neurons, but matrix W2 only 14*1=14 neurons. At each stage the first sub-network calculates a lower target function $\Phi 1$ as the second sub-network. In the middle and the final part of the iterative process, the characteristics are the same. This means that the sub-networks achieved only small corrections to their matrix coefficients and the standard gradient algorithm is not efficient (Fig.6).

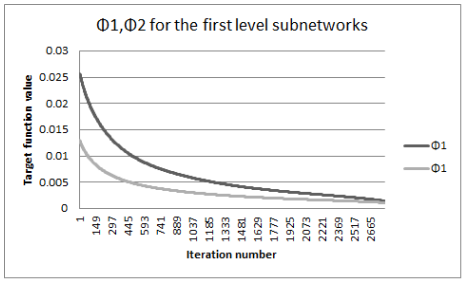


Fig. 6. Learning error with regards to the iteration number. The learning parameter $\lambda_1 = 0.2$

3.2 Linear Performance Balance Principle

As stated above, the performance prediction principle is more general that interface prediction. The coordinator gets information not from all of the elements of the output vectors, Y_1, Y_2, as in the previous algorithm, but only some general information, such as the function value Φ_1, Φ_2 and their derivatives.

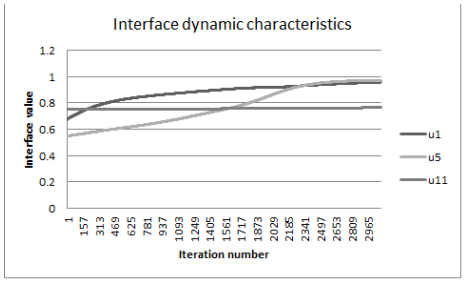


Fig. 7. Dynamic characteristics of the hidden interface value

In Fig. 6, the quality of the learning process is shown . Error functions decrease their value quite fast and the oscillation does not exist. On Fig.7. dynamic characteristics of interface value u_1, u_5, u_{11} are shown. Part of them change their value dynamically.

3.3 Nonlinear Performance Balance Principle

The sub-network transfer function is nonlinear. The coordinator structure should consider this and use a more complicated target function structure. According to formulas (24)(26), the function Ψ includes a nonlinear part: the multiplication or sum of the second power target functions. In Fig. 8 the quality of the learning process is shown. A flex point for all of the sub-networks characterizes this

learning process. After that, the learning process achieves the minimum value in an asymptotic way. Learning process quality is different for $\lambda_1 - 0.2$. It is stable and smooth (Fig.9). The coordinator algorithm requires two parameters: the learning coefficient λ_1 and the parameter c. The learning process quality depends on the "c"-value. Fig. 10 shows this characteristic, including the iteration number for the flex point and the total iteration number.

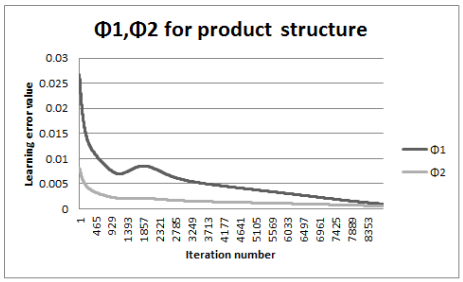


Fig. 8. The iteration number for nonlinear coordination function. $\lambda_1 = 0.3$

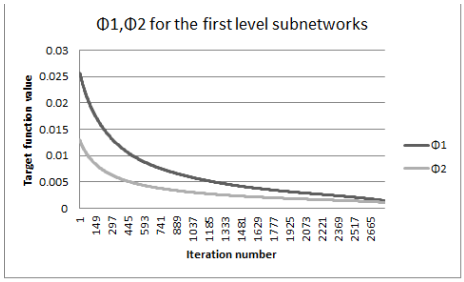


Fig. 9. The iteration number for nonlinear coordination function. $\lambda_1 = 0.2$

4 Conclusion

In this article, only a single principle was examined. The interaction balance principle can be realized by measuring two different feedback signals: the sub-network output signals Y_1, Y_2 or the sub-network performance value Φ_1, Φ_2. In the first case, the learning process is very flexible for the coordinator learning parameter λ. For $\lambda = 0.3$, the last learning process stage includes oscillation, which delays the process of convergence. For $\lambda = 0.2$ the characteristics are better. Learning processes are not simple and depend not only on an ANN structure

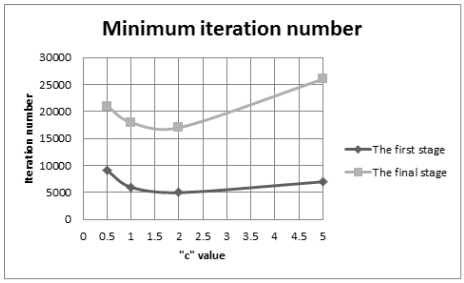


Fig. 10. Optimal parameters for Nonlinear Performance Balance Principle

but on the input data structure as well. To measure sub-network performance the coordinator can receive more general information. This has a positive impact on all the dynamic characteristics (Fig.6). In particular the oscillation seen in the middle of the learning process are smaller. From the coordinator point of view, every sub-network can be seen as a black box with the input signal U^{γ} and output Y_1 or Y_2. Response functions have to be nonlinear because the global target function is also nonlinear. The coordinator, when using a stable coordinator algorithm with constant learning parameters is not responsible for finding the optimal interaction vector U^{γ} during the entire learning process (from beginning to end). This has a negative impact on the quality and speed of convergence. Using the hierarchical principle, an additional level can be added: the adaptation level. The adaptation level, using its own identification algorithm, can predict learning parameters as λ and c. This theme should be studied in future. Finally, the nonlinear coordinator algorithm was examined. In that case all algorithms are very flexible with regarded to parameter c", which decides about the impact of the nonlinear part of the coordinator algorithm for convergence. With a small value, learning time is very long, but for a large value, oscillations are seen. The characteristic of $n = f(c)$, where: n- iteration number is shown in Fig. 10.

References

1. Placzek Stanislaw. A two-level on-line learning algorithm of Artificial Neural Network with forward connections IJARAI, vol.3, no. 12, 2014
2. Placzek Stanislaw. Decomposition and the principle of interaction prediction in hierarchical structure of learning algorithm of ANN Poznan University of Technology, Academic Journal. Electrical Engineering, no 84, Poznan 2015
3. Mesarovic M.D., Macka D., Takahara Y. Theory of hierarchical multilevel systems, Academic Press, New York and London 1970
4. Haykin S. Neural network, a comprehensive foundation. Macmillan College Publishing Company, New York 1994.
5. Vapnik V. Statistical Learning Theory Wiley, New York 1998
6. Osowski S. Sieci neuronowe w ujeciu algorytmicznym WNT Warszawa 1996

On the model checking of sequential reactive systems

D.G. Kozlova[1] V.A. Zakharov[2]

[1] Faculty of Computational Mathematics and Cybernetics,
Lomonosov Moscow State University, Moscow, RU-119899, Russia,
[2] Faculty of Computer Science,
National Research University Higher School of Economics,
Moscow, Russia
(Corresponding author: zakh@cs.msu.su)

Abstract. By sequential reactive system we mean a program which operates in the interaction with the environment permanently receiving data (requests) from it. At receiving a piece of data a program performs a sequence of actions (response) and displays the current result. Such programs usually arise at implementation of computer drivers, on-line algorithms, control procedures. Basic actions performed by these programs may be regarded as generating elements of a certain semigroup. This consideration opens the way to model sequential reactive systems by finite state transducers that operate over semigroups. This model of computation can be used for synthesis, optimization, verification and testing of sequential reactive systems. In this paper we make an attempt to originate a framework for developing verification techniques for sequential reactive systems by taking advantage of finite state transducers as a formal model. To this end we introduce a LTL-based formal language which may be suitable for specification of the behaviour of sequential reactive systems and adopt a well known LTL-based model checking techniques for verification of finite state transducers against these specifications.

1 Introduction

Finite state transducers extend the finite state automata to model functions and relations on strings or lists. They are used in many fields as diverse as computational linguistics [14] and model-based testing [1, 22]. In software engineering transducers provide a suitable formal model for various on-line algorithms and device drivers for manipulating with strings, transforming images, filtering dataflows, inserting fingerprints, sorting data, etc.

An ordinary model of finite state transducers over words can be further extended to encompass a more wide class of sequential reactive programs. These programs operate in the interaction with the environment permanently receiving data (requests) from it. At receiving a piece of data such program performs a sequence of actions. When certain control points are achieved a program outputs the current results of computation as a response. What matters is that different

sequences of actions may yield the same result. Therefore, the basic actions of a program may be viewed as generating elements of some appropriate semigroup, and the result of computation may be regarded as the composition of actions performed by the program.

Let us consider some examples. Imagine that a radio-controlled robot moves on the earth surface. It can make one step moves in any of 4 directions N, E, S, W. When such robot receives a control signal syg in a state q it must choose and carry out a sequence of steps (say, N, N, W, S), and enter to the next state q'. At some distinguished state q_{fin} robot reports its current location. Movements of the robot may be regarded as basic actions, and the most simple model of computation which is suitable for analyzing a behaviour of this robot is non-deterministic finite state transducer operating on free Abelian group of rank 2. Next, consider a network switch which receives as input packet flows alternating with control instructions. Following to its flow table a switch sends modified copies of every packet into one or another output port. A flow table is updated at receiving a control instruction. Modifications and forwardings of a data packet may be regarded as basic actions. When a switch forwards two packets from different packet flows to different ports, the corresponding actions can be performed in an arbitrary order. Therefore, such a switch can be modeled by a finite state transducer operating on a partially commutative semigroup. Semigroups of this kind are also known as traces; they are thoroughly studied in [9].

When designing sequential reactive systems software engineers want to be confident of their correct behaviour. For example, in the case of radio-controlled robot it may be required that it never appears in the north-west sector of the surface, obligatory passes via certain locations, and can be always returned to the starting point at receiving a particular sequences of control messages. When a network switch is concerned, its computations should comply with the requirements of forwarding policies (see, e.g. [6, 7]) such as the absence of forwarding loops, non-interference of certain packet flows, etc. To analyze the behaviour of sequential reactive systems one may use the concept of finite state transducer over finitely generated semigroups as a formal model of such systems and develop various verification techniques (equivalence checking, model checking, deductive verification, etc.) for these class of transducers.

Equivalence checking problem for finite state transducers has been studied in much details in many papers. Its study for classical transducers that operate on words began in the early 60s. First, it was shown that the equivalence checking problem is undecidable for non-deterministic transducers [11] even over 1-letter input alphabet [12]. But the undecidability displays itself only in the case of unbounded transduction when an input word may have arbitrary many images. At the next stage bound-valued transducers were studied. The equivalence checking problem was shown also to be decidable for deterministic [4], functional (single-valued) transducers [3, 18], and k-valued transducers [8, 23]. In a series of papers [16, 17, 19] techniques for checking bounded valuedness, k-valuedness and equivalence of finite state transducers over words were developed. Recently in [25] equivalence checking problem was shown to be decidable for finite state

transducers that operate over finitely generated semigroups embeddable in decidable groups.

There are also papers where equivalence checking problem for transducers is studied in the framework of program verification. The authors of [20] proposed models of communication protocols as finite state transducers operating on bit strings. They set up the verification problem as equivalence checking between the protocol transducer and the specification transducer. The authors of [22] extend finite state transducers with symbolic alphabets which are represented as parametric theories. They showed that a number of classical problems for extended transducers, including equivalence checking problem, are decidable modulo underlying theories. In [1] a model of streaming transducers was proposed for programs that access and modify sequences of data items in a single pass. It was shown that a number of verification problems such as equivalence checking, assertion checking, and checking correctness with respect to pre/post conditions, are decidable for this transducer model.

Unlike equivalence checking, model checking of (or related with) transducers is less well studied. Transducers found a usage in regular model checking of parameterized distributed systems. In some formal models of these systems configurations are modeled as words over finite alphabet. In such a situation a transition relation on these configurations is a binary relation on finite words which can be adequately specified by finite state transducers (see [5, 24]). In this line of research transducers play the role of verification instrument, but not an object of verification. As for verification of transducers, to the extend of our knowledge no special purpose study of model checking problem for finite state transducers has been conducted so far. In the opinion of the authors of this paper, this is due the following reason. Both the influence of the environment upon a reactive system and its response is defined in terms of a set of basic predicates. The letters of input and output alphabets of a transducer are regarded as valuations (tuples of truth values) of these predicates, and transducers are viewed as special presentation of finite labeled transition system (Kripke structure) (see [2]). From this viewpoint model checking problem for finite state transducers conforms well to standard model checking scheme for finite structures, and, therefore, are not worthy of any particular treatment.

However, these arguments become invalid when a response of a reactive system at every step of its computation is regarded as a composition of actions produced by the system so far. In this case the predicates which specify the basic properties of reactive systems behaviour are defined on finite sequences of actions, i.e. every such predicate is a language over an alphabet of output actions. More complex dynamic properties can be expressed by LTL formulae. It should be remarked that these formulae must express not only the properties of output sequences of actions but relationships between input sequences of requests from the environment (signal flows) and output sequences of responding actions (compound actions). This can be achieved through the introduction of behaviour patterns of the environment as the sets of signal flows and by parametrization of temporal operators with these patterns.

In this paper we make an attempt to introduce a LTL-based formal language for specification of the behaviour of sequential reactive systems and to adapt a well known LTL-based model checking techniques [13, 21] for verification of finite state transducers. The paper is organized as follows. In the next section a concept of finite state transducer over semigroup (see [25]) as a formal model of sequential reactive systems is defined. In section 3 we introduce $\mathcal{LP}$-LTL — a parameterized version of Linear Temporal Logics — as a formal language for specifying behaviour of sequential reactive systems and set up model checking problem for finite state transducers. In section 4 we present a $\mathcal{LP}$-LTL model checking algorithm for the case when both basic properties of reactive systems and behaviour patterns of the environment are defined by finite state automata. Finally, we briefly discuss some possible directions for further research.

2 Transducers as models of reactive systems

Let $\mathcal{C}$ and $\mathcal{A}$ be two finite sets. The elements of $\mathcal{C}$ are called *signals*; they may be viewed as abstractions of messages (control instructions, instrument or sensor readings, pieces of data, etc.) received by a reactive system from its environment. Finite sequences of signals (words over alphabet $\mathcal{C}$) are called *signal flows*. As usual, the set of all signal flows is denoted by $\mathcal{C}^*$. We write uv for concatenation of signal flows u and v, and ε for the empty signal flow.

The elements of $\mathcal{A}$ are called *basic actions*; they are the abstractions of operations (data processings, movements, etc.) performed by a reactive system in response to received signals. Finite sequences of basic actions (words over alphabet $\mathcal{A}$) are called *compound actions*.

Actions are interpreted over semigroups. Consider a semigroup $(S, e, \circ)$ generated by the set $\mathcal{A}$, where S is a set of semigroup elements, e is the neutral element, and $circ$ is a composition operation. The elements of S may be regarded as *data states*. Every basic action $a, a \in \mathcal{A}$, when been applied to a data state $s, s \in S$, yields the result $s \circ a$. Every compound action $h = a_1 a_2 \dots a_k$ is interpreted as the composition $[h] = a_1 \circ a_2 \circ \dots \circ a_k$.

A *trajectory* on a semigroup $(S, e, \circ)$ is a pair $tr = (s_0, \alpha)$ such that $s_0 \in S$ and α is an infinite sequence

$$\alpha = (c_1, s_1), (c_2, s_2), \dots, (c_i, s_i), \dots,$$

where $c_i \in \mathcal{C}, s_i \in S$ for every $i, i \geq 0$. This sequence represents a possible behaviour of a reactive system as it becomes visible to an outside observer: starting from the data state s_0 the system every time at receiving a next signal c_i performs some compound action h_i and displays its effect $s_i = s_{i-1} \circ h_i$. Given a trajectory $tr = (s_0, \alpha)$ and an integer $i, i \geq 0$, denote by $tr|^i$ the trajectory $(s_i, \alpha|^i)$, where $\alpha|^i = (c_{i+1}, s_{i+1}), (c_{i+2}, s_{i+2}), \dots$

A *finite state transducer* over a set of signals $\mathcal{C}$ and a set of basic actions $\mathcal{A}$ is a system $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$, where Q is a finite set of *control states*, $Q_0, Q_0 \subseteq Q$, is a set of *initial states*, and $T, T \subseteq Q \times \mathcal{C} \times Q \times \mathcal{A}^*$ is a *transition relation*. Every quadruple (q, c, q', h) in T is called a *transition*: when a transducer is in a control

state q and receives a signal c it passes its control to a state q' and performs a compound action h. Such transitions are usually depicted as $q \xrightarrow{c,h} q'$. It is assumed that T is a total relation: for every control state q and a signal c the set T includes at least one transition of the kind $q \xrightarrow{c,h} q'$. A *run* of π is any sequence of transitions

$$run = q_0 \xrightarrow{c_1,h_1} q_1 \xrightarrow{c_2,h_2} q_2 \xrightarrow{c_3,h_3} \cdots \tag{1}$$

which begins from some initial state q_0. We write $run|^i$ for the suffix of the sequence run which begins from the state $q_i, i \geq 0$. The size $|\pi|$ of a transducer π is the number $|Q|$ of its state.

Finite state transducers can serve as formal models of sequential reactive systems. At each step of its computation it receives a signal c from the environment and performs a transition $q \xrightarrow{c,h} q'$ by passing its control to a state q' and executing an action h. Usually behaviour of transducers is defined as transduction relation between input and output words. But it can be rather well defined in terms of trajectories as follows. Suppose that basic actions of a transducer $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$ are interpreted over a semigroup $(S, e, \circ)$. Then every run (1) of π generates a trajectory $tr(run) = (e, \alpha)$, where the sequence $\alpha = (c_1, s_1), (c_2, s_2), \ldots, (c_i, s_i), \ldots$, is such that $s_1 = e \circ h_1$, and $s_i = s_{i-1} \circ h_i$ holds for every $i, i \geq 2$. The set of all trajectories generated by the runs of π is denoted by $Tr(\pi, S)$. This set completely characterizes a behaviour of sequential reactive system modeled by a transducer π over a semigroup of actions $(S, e, \circ)$.

3 Specification language

Specification languages are intended to describe formally desirable (or erroneous) behaviour of computing systems. Since the behaviour of a sequential reactive system is presented as a set of trajectories, the expressions of an appropriate specification language should be interpreted over trajectories. Every trajectory displays how the data states from the set S changes as a reactive system receives signals and performs responding actions with the passage of time. Therefore, it is advantageous to take some variant of temporal logics as a framework of such a specification language.

The formulae of temporal logics are built of basic predicates by means of Boolean connectives and temporal operators. Basic predicates are defined on data states. In our model of sequential reactive systems data states are interpreted as elements of a semigroup $(S, e, \circ)$. Thus, basic predicates can be regarded as certain subsets of S. They can be formally specified alternatively in different ways.

1. By means of parameterized algebraic equations in a semigroup: a data state s satisfies a basic predicate $Eq(p, X)$ iff s is such a value of a parameter p that an equation $Eq(p, X)$ has a solution in a semigroup $(S, e, \circ)$. For example, an equation $p \circ X = e$ specifies a set of data states s from which a computation of a reactive system can be restarted.

2. By any means for defining formal languages over a set of basic actions $\mathcal{A}$ — formal grammars, language equations, automata of various types, etc.: a data state s satisfies a predicate L, where L is a language over $\mathcal{A}$, iff $s = [h]$ for some compound action h such that $h \in L$. For example, a finite state automaton A distinguishes a set of data states s such that $s = [h]$ for some compound action h accepted by A.

A sequential reactive system modifies data states in response to incoming signals. These signals come to an input of a system in conformity with a certain scenario (pattern) of environment's behaviour. An environment behaviour pattern characterizes a set of possible signal flows that may affect a reactive system. Therefore, a specification of its behaviour must include some references to signal flows. This can be achieved by using formal descriptions of environment behaviour patterns as parameters of temporal operators. Since a signal flow is but a word over a set of signals $\mathcal{C}$, such descriptions can be provided by any means used for defining formal languages — grammars, equations, automata.

These contemplations bring us to the following concept of formal specification language $\mathcal{LP}$-LTL for sequential reactive systems. Given a set of signals $\mathcal{C}$, a set of basic actions $\mathcal{A}$, and a semigroup $(S, e, \circ)$ generated by basic actions, we say that any set of finite words (language) over the alphabet $\mathcal{C}$ is an *environment behaviour pattern* (or, simply, a *pattern*), and any subset $S', S' \subseteq S$, is a *basic predicate*.

Select a family of patterns $\mathcal{L}$ and a family $\mathcal{P}$ of basic predicates. Then a set of $\mathcal{LP}$-LTL formulae is the minimal set $Form$ of expressions which satisfy the following rules:

1) every basic predicate $P, P \in \mathcal{P}$ is a $\mathcal{LP}$-LTL formula;
2) if φ, ψ are $\mathcal{LP}$-LTL formulae then $\neg\varphi$, $\varphi \wedge \psi$ and $\varphi \vee \psi$ belong to $Form$;
3) if $\varphi \in Form$ and $c \in \mathcal{C}$ then $X_c\varphi$, $Y_c\varphi$ belong to $Form$;
4) if $\varphi \in Form$ and $L \in \mathcal{L}$ then $F_L\varphi$, $G_L\varphi$ belong to $Form$ as well.

This definition is constructive, since $\mathcal{L}$ and $\mathcal{P}$ may be thought of as the set of names interpreted over patterns and basic predicates. The size $|\varphi|$ of a formula φ is the number of Boolean connectives and temporal operators occurred in φ.

The semantics of the specification language is defined in terms of satisfiability relation $\models$ of $\mathcal{LP}$-LTL formulae on trajectories. Let $tr = (s_0, \alpha)$ be a trajectory, where $\alpha = (c_1, s_1), (c_2, s_2), \ldots, (c_i, s_i), \ldots$, and φ be a $\mathcal{LP}$-LTL formula. Then

1) if $P \in \mathcal{P}$ then $tr \models P \iff s_0 \in P$;
2) $tr \models \neg\varphi \iff$ it is not true that $tr \models \varphi$;
3) $tr \models \varphi \wedge \psi \iff tr \models \varphi$ and $tr \models \psi$;
4) $tr \models \varphi \vee \psi \iff tr \models \varphi$ or $tr \models \psi$;
5) $tr \models X_c\varphi \iff c = c_1$ and $tr|^1 \models \varphi$;
6) $tr \models Y_c\varphi \iff c \neq c_1$ or $tr|^1 \models \varphi$;
7) $tr \models F_L\varphi \iff \exists\, i \geq 0 \ : \ c_1c_2 \ldots c_i \in L$ and $tr|^i \models \varphi$;
8) $tr \models G_L\varphi \iff \forall\, i \geq 0 \ : \ c_1c_2 \ldots c_i \in L$ implies $tr|^i \models \varphi$.

Clearly, some other parameterized temporal operators that are used in LTL like U (until), W (weak until), R (release) can be introduced in the same way. Moreover, some new temporal operators that are specific for $\mathcal{LP}$-LTL may be introduced. For example, to express some properties of trajectories one may need a weak eventuality operator $\widehat{F}_L$ which has the following semantics:

$tr \models \widehat{F}_L\varphi \iff$ either $\forall\, i \geq 0 \ : \ c_1c_2\ldots c_i \notin L$, or $tr \models F_L\varphi$.

It is easy to ascertain that parameterized temporal operators introduced above satisfy duality and fixed-point (expansion) properties.

Proposition 1. *Let φ be an arbitrary $\mathcal{LP}$-LTL formula, $c \in \mathcal{C}$, $L \subseteq \mathcal{C}^*$, and tr be an arbitrary trajectory. Then*

1) $tr \models \neg X_c\varphi \iff tr \models Y_c\neg\varphi$,
2) $tr \models \neg Y_c\varphi \iff tr \models X_c\neg\varphi$,
3) $tr \models \neg F_L\varphi \iff tr \models G_L\neg\varphi$,
4) $tr \models \neg G_L\varphi \iff tr \models F_L\neg\varphi$.

For every pattern L and a signal c denote by $Pref_1(L)$ the set $\{c : \exists w \in \mathcal{C}^* : cw \in L\}$ of 1-letter prefixes of signal flows in L, and by $Suff_c(L)$ the pattern $\{w : cw \in L\}$ which consists of maximal proper suffixes of those signal flows in L that begin with the signal c. We say that a family of patterns $\mathcal{L}$ is suffix-closed iff for every signal c and every pattern $L, L \in \mathcal{C}$, the pattern $Suff_c(L)$ also belongs to $\mathcal{L}$.

Proposition 2. *Suppose that a family of patterns $\mathcal{L}$ is suffix-closed, and let φ be a $\mathcal{LP}$-LTL formula, and tr be a trajectory. Then*

1) *if $\varepsilon \in L$ then* $tr \models F_L\varphi \iff tr \models \varphi \vee \bigvee\limits_{c\in Pref_1(L)} X_cF_{Suff_c(L)}\varphi$,
2) *if $\varepsilon \notin L$ then* $tr \models F_L\varphi \iff tr \models \bigvee\limits_{c\in Pref_1(L)} X_cF_{Suff_c(L)}\varphi$,
3) *if $\varepsilon \in L$ then* $tr \models G_L\varphi \iff tr \models \varphi \wedge \bigwedge\limits_{c\in Pref_1(L)} Y_cF_{Suff_c(L)}\varphi$,
4) *if $\varepsilon \notin L$ then* $tr \models G_L\varphi \iff tr \models \bigwedge\limits_{c\in Pref_1(L)} Y_cF_{Suff_c(L)}\varphi$.

As in the case of ordinary LTL these properties are important for building model checking and satisfiability checking procedures for $\mathcal{LP}$-LTL formulae.

4 Model checking sequential reactive systems against $\mathcal{LP}$-LTL specifications

Assume that sequential reactive systems are modeled by finite state transducers that operate over a set of signals $\mathcal{C}$ and the set of basic actions $\mathcal{A}$ interpreted in a semigroup $(S, e, \circ)$. Let $\mathcal{L}$ and $\mathcal{P}$ be families of patterns and basic predicates. Then model checking (MC) problem for sequential reactive systems against $\mathcal{LP}$-LTL specifications is that of checking, given a finite state transducer π and a

$\mathcal{LP}$-LTL formula φ, whether $tr \models \varphi$ holds for every trajectory tr in $Tr(\pi, S)$ (or, in symbols, $Tr(\pi, S) \models \varphi$).

It is evident that decidability and complexity of MC problem for sequential reactive systems against $\mathcal{LP}$-LTL specifications essentially depend on 1) a semigroup $(S, e, \circ)$ used for interpretation of basic actions, 2) a family of basic predicates $\mathcal{P}$ on the set of data states S, and 3) a family of behaviour patterns of the environment $\mathcal{L}$ used for parametrization of temporal operators. In some cases this problem has an effective solution.

We studied the most simple case of MC problem when 1) basic actions are interpreted over free monoid $(S, e, \circ)$, where S is the set of compound actions $\mathcal{A}^*$, $e = \varepsilon$, and $\circ$ is concatenation operation on compound actions, 2) a family $\mathcal{P}$ of basic predicates is the collection of all regular sets of compound actions, 3) a family $\mathcal{L}$ of behaviour patterns of the environment is the collection of all regular sets of signal flows. $\mathcal{LP}$-LTL formulae of this type will be called *Reg-LTL* formulae. The main advantage of *Reg-LTL* is that the most simple model of computation — deterministic finite state automata — can be involved to define basic predicates and patterns occurred in these formulae.

By (non-initialized) *deterministic finite state automaton* we mean a quadruple $K = (\Sigma, Z, Z_{acc}, \Phi)$, where Σ is a finite input alphabet, Z is a finite set of states, $Z_{acc}, Z_{acc} \subseteq Z$, is a subset of accepting states, and $\Phi : Z \times \Sigma \to Z$ is a total transition function. A transition function can be extended to the set Σ^* in the usual fashion: $\Phi(z, \varepsilon) = z$, and $\Phi(z, bw) = \Phi(\Phi(z, b), w)$ for every state z, a letter b in Σ and a word $w, w \in \Sigma^*$. By initialized automaton we mean a pair (K, z_0), where z_0 is a state of an automaton K. An initialized automaton (K, z_0) accepts a word w if $\Phi(z_0, w) \in Z_{acc}$; thus, it specifies a regular language $L(K, z_0) = \{w \ : \ \Phi(z_0, w) \in Z_{acc}\}$ of all accepted words.

When finite state automata are used for specification of regular basic predicates they have the set of basic actions $\mathcal{A}$ as an input language; automata of this kind will be called $\mathcal{A}$-automata. When finite state automata are employed for specification of regular patterns of the environment they have the set of signals $\mathcal{C}$ as an input alphabet; automata of this sort will be called $\mathcal{C}$-automata. Thus, every atomic formula of *Reg-LTL* is an initialized $\mathcal{A}$-automaton (A, z_0), and temporal operators used in *Reg-LTL* are those of the form $X_c, Y_c, F_{(B,z_0)}, G_{(B,z_0)}$, where c is a signal, and (B, z_0) is an initialized $\mathcal{C}$-automaton. In what follows we will use letters Z, Z_{acc} and Φ as generic names of a set states, a subset of accepting states and a transition functions in automata that specify basic predicates and patterns of the environment.

The rules of *Reg-LTL* semantics can be re-defined in terms of finite state automata. Suppose, for example, that a run of a transducer begins with a transition $q \xrightarrow{c,h} q'$. Then $tr(run) \models X_c(A, z_0) \iff h \in (A, z_0) \iff \Phi(z_0, h) \in Z_{acc}$. This effect also shows itself for other formulae. Given a $\mathcal{A}$-automaton (A, z_0) and a compound action h, we say that the $\mathcal{A}$-automaton $(A, \Phi(z_0, h))$ is *h-shift* of basic predicate (A, z_0). In more general case, a h-shift of a *Reg-LTL* formula φ is a formula $shift(\varphi, h)$ which is obtained from φ by replacing every basic predicate (A, z_0) occurred in φ with its h-shift $(A, \Phi(z_0, h))$. Consider a run (1)

of a transducer π. Then

$$\begin{array}{lll} tr(run) \models F_{(B,z_0)}\varphi & \Longleftrightarrow & \exists\ i \geq 0\ :\ \Phi(z_0, c_1c_2 \ldots c_i) \in Z_{acc} \ \text{ and} \\ & & tr(run|^i) \models shift(\varphi, h_1h_2 \ldots h_i); \\ tr(run) \models G_{(B,z_0)}\varphi & \Longleftrightarrow & \forall\ i \geq 0\ :\ \Phi(z_0, c_1c_2 \ldots c_i) \in Z_{acc} \ \text{ implies} \\ & & tr(run|^i) \models shift(\varphi, h_1h_2 \ldots h_i); \end{array}$$

These relationships are essential in the designing of *Reg-LTL* model checking algorithm in Theorem 1.

For the sake of brevity we will skip references to a semigroup $(S, e, \circ)$ in our notation till the end of the section. It is assumed that this semigroup is a free monoid of finite words over $\mathcal{A}$ and MC problem $Tr(\pi) \models \varphi$ is studied for finite state transducers against $\mathcal{LP}$-*LTL* specifications.

The main result of this section is

Theorem 1. *Let $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$ be a finite state transducer operating on a free monoid of words, and φ be a Reg-LTL formula. Suppose that every regular component (a basic predicate or a pattern of the environment) of φ is specified by a deterministic finite state automata which has N states at the most. Then there exists a generalized Büchi automaton $M[\pi, \varphi]$ such that*

- *$M[\pi, \varphi]$ has $|\pi| 2^{O(|\varphi| N^{|\varphi|})}$ states at the most;*
- *$M[\pi, \varphi]$ can be constructed effectively by π and φ in time polynomial of the size of $M[\pi, \varphi]$;*
- *$M[\pi, \varphi]$ accepts empty ω-language iff $Tr(\pi) \models \varphi$.*

Proof. (Sketch) Our algorithm for the translation of a pair (π, φ) to a Büchi automaton $M[\pi, \varphi]$ follows the well-known scheme for translation of LTL formulae to Büchi automata which was introduced in [21]. We only emphasize those aspects of this translation which are specific for *Reg-LTL*.

1. Consider the formula $\psi = \neg\varphi$ and present it in negation normal form via duality laws (see Proposition 1). It should be noted that if a basic predicate is specified by an automaton (A, z_0) then $\neg(A, z_0) \equiv (\bar{A}, z_0)$, where $\bar{A}$ is a complementation of A. Thus, we eliminate all negations in ψ.

2. Define the closure $cl(\psi)$ of ψ as the minimal set of *Reg-LTL* formulae which complies with the following rules:

- $\psi \in cl(\psi)$,
- $(A, z_0) \in cl(\psi) \ \Rightarrow\ \forall\ z \in Z :\ (A, z) \in cl(\psi)$
- $f \vee g \in cl(\psi) \ \Rightarrow\ f, g \in cl(\psi)$,
- $f \wedge g \in cl(\psi) \ \Rightarrow\ f, g \in cl(\psi)$,
- $X_c f \in cl(\psi) \ \Rightarrow\ shift(f, h) \in cl(\psi)$ for every $h \in \mathcal{A}^*$,
- $Y_c f \in cl(\psi) \ \Rightarrow\ shift(f, h) \in cl(\psi)$ for every $h \in \mathcal{A}^*$,
- $F_{(B,z_0)} f \in cl(\psi) \ \Rightarrow\ f \in cl(\psi)$ and $\forall\ c \in \mathcal{C} :\ X_c F_{(B,\Phi(z_0,c))} f \in cl(\psi)$,
- $G_{(B,z_0)} f \in cl(\psi) \ \Rightarrow\ f \in cl(\psi)$ and $\forall\ c \in \mathcal{C} :\ Y_c G_{(B,\Phi(z_0,c))} f \in cl(\psi)$.

As it can be seen from the definition of $cl(\psi)$ this set may contain $O(|\varphi|N^{|\varphi|})$ at the most.

3. Build the collection $CS(\psi)$ of all subsets of $cl(\psi)$ which are both locally consistent and saturated. A subset K of $cl(\psi)$ is called locally consistent if it satisfies the following requirements:

- if $(A, z_0) \in K$ then $z_0 \in Z_{acc}$;
- if $X_{c_1} f \in K$ and $X_{c_2} f \in K$ then $c_1 = c_2$,

and it is called saturated if it fulfills the rules listed below:

- if $f \vee g \in K$ then $f \in K$ or $g \in K$;
- if $f \wedge g \in K$ then $f \in K$ and $g \in K$;
- if $F_{(B,z_0)} f \in K$ then either $X_c F_{(B,\Phi(z_0,c))} \in K$ for some signal c, or $f \in K$ in the case of $z_0 \in F_{acc}$;
- if $G_{(B,z_0)} f \in K$ then $Y_c G_{(B,\Phi(z_0,c))} \in K$ for every signal c, and, moreover, f is also in K in the case of $z_0 \in F_{acc}$.

4. Build a generalized Büchi automaton $M[\pi, \varphi] = (Q \times CS(\psi), Init, \Delta, \mathbf{F})$ over the input alphabet $\mathcal{C} \times \mathcal{A}^*$, where

- $Q \times CS(\psi)$ is the set of states of the automaton,
- $Init = \{(q_0, K) : \ \psi \in K\}$ is the set of initial states,
- $\Delta = \Delta_1 \cup \Delta_2 \cup \Delta_3$ is a transition relation which is defined as follows:
  - $(q, K) \xrightarrow{c,h} (q', K') \in \Delta_2$ iff 1) $q \xrightarrow{c,h} q' \in T$, 2) a set K contains at least one formulae $X_c\varphi$, and 3) $\{shift(\varphi, h) : X_c\varphi \in K \text{ or } Y_c\varphi \in K\} \subseteq K'$;
  - $(q, K) \xrightarrow{c,h} (q', K') \in \Delta_1$ iff 1) $q \xrightarrow{c,h} q' \in T$, 2) a set K does not contain any X-formulae, and 3) $\{shift(\varphi, h) : X_c\varphi \in K\} \subseteq K'$;
  - $(q, K) \xrightarrow{c,h} (q', K) \in \Delta_3$ iff 1) $q \xrightarrow{c,h} q' \in T$, and 2) a set K does not contain neither X-formulae, nor Y-formulae.
- $\mathbf{F} = \{\mathbf{F}_\varphi : \varphi \text{ is a } F\text{-formula in } cl(\psi)\}$ is a family of acceptance conditions, where for every $\varphi = F_{(B,z)} f$ the acceptance condition $\mathbf{F}_\varphi$ is a set of all such pairs (q, K) that satisfy a requirement:
  $F_{(B,z')} shift(f, h) \in K \ \Rightarrow \ shift(f, h) \in K$.

5. Following the same line of reasoning as in [21] one could show that $M[\pi, \varphi]$ has an accepting computation iff the set $Tr(\pi)$ includes a trace tr such that $tr \models \psi$. Thus, $M[\pi, \varphi]$ is empty iff $Tr(\pi) \models \varphi$. □

Since emptiness of generalized Büchi automata can be checked in polynomial time we arrived at

Corollary 1. *Regular models checking of sequential reactive systems can be performed effectively in time polynomial of the size of a model (finite state transducer) and double exponential of the size of a specification (Reg-LTL formula).*

5 Conclusion

The main contribution of this paper is twofold:

1. we introduce a new framework for formal verification of sequential reactive systems; it includes a concept of finite state transducer over semigroups as a formal model of sequential reactive systems, and a formal language for specifying behaviour of transducers.
2. we set up a model checking problem for finite state transducers operating over semigroups and show that conventional model checking techniques is applicable to this problem (at least in the case of transducers over free monoids).

There are questions and problems that still remain open for further research. What is an expressive power of $\mathcal{LP}$-LTL? We surmise that some $\mathcal{LP}$-LTL-specific operators could be introduced to make this language more convenient in practice. We believe also that other temporal logics (say, CTL) could be also adapted appropriately for specification of sequential reactive systems behaviour. Model checking algorithm presented in Theorem 1 needs further improvement. To this end complexity issues of $\mathcal{LP}$-LTL need to be studied. We are sure that a more advanced on-the-fly approach used in LTL model checking [10] could be applied to efficient verification of transducers against $\mathcal{LP}$-LTL. In this paper we presented in some details a solution to verification problem for finite state transducers over free semigroups. But we believe that this result can be extended further to comprise the cases of partially commutative semigroups (traces [9]), free groups and free Abelian groups.

References

1. Alur R., Cerny P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. Proc. of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2011), p. 599-610.
2. Alur R., Moarref S., and Topcu U.: Pattern-based refinement of assume-guarantee specifications in reactive synthesis. Proc. of 21-st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2015.
3. Blattner M, Head T.: Single-valued a-transducers. Journal of Computer and System Sciences. **15** (1977), p. 310-327.
4. Blattner M, Head T.: The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences. **19** (1979), p. 45-49.
5. Bouajjani A., Jonsson B., Nilsson M., Touili T.: Regular Model Checking. Proc. of 12-th International Conference on Computer Aided Verification, LNCS **1855** (2000), p. 403-418.
6. M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford.: A NICE way to Test OpenFlow Applications. Proc. of Networked Systems Design and Implementation, April 2012.
7. Chemeritsky E. V., Smeliansky R. L., Zakharov V. A.: A formal model and verification problems for software defined networks. Automatic Control and Computer Sciences. **48** (2014), p. 398406.

8. Culik K., Karhumaki J.: The equivalence of finite-valued transducers (on HDTOL languages) is decidable. Theoretical Computer Science. **47** (1986), p. 71-84.
9. Diekert V., Metivier Y.: Partial commutation and traces. Handbook of formal languages. **3** (1997), p. 457-533.
10. Gerth R., Peled D., Vardi M. Y., Wolper P.: Simple on-the-y automatic verication of linear temporal logic. In Protocol Specification, Testing, and Verification, (1995), p 318.
11. Griffiths T.: The unsolvability of the equivalence problem for ε-free nondeterministic generalized machines. Journal of the ACM **15** (1968), p. 409-413.
12. Ibarra O.: The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. SIAM Journal on Computing, 1978, v. 4.
13. Kesten Y., Manna Z., McGuire H., Pnueli A.: A decision algorithm for full propositional temporal logic. Proc. of 5-th International Conference on Computer Aided Verification, LNCS **697** (1993), p. 97-109.
14. Mohri M.: Finite-state transducers in language and speech processing. Computational Linguistics. **23** (1997), p. 269-311.
15. Reutenauer C., Schuzenberger M.P.: Minimization of rational word functions. SIAM Journal of Computing. **30** (1991), p. 669-685.
16. Sakarovitch J., de Souza R.: On the decomposition of k-valued rational relations. Proc. of 25th International Symposium on Theoretical Aspects of Computer Science. (2008), p.621-632.
17. Sakarovitch J., de Souza R.: On the decidability of bounded valuedness for transducers. Proc. of the 33rd International Symposium on MFCS. (2008), p. 588-600.
18. Schutzenberger M. P.: Sur les relations rationnelles. Proc. of Conference on Automata Theory and Formal Languages. (1975), p. 209-213.
19. de Souza R.: On the decidability of the equivalence for k-valued transducers. Proc. of 12th International Conference on Developments in Language Theory. (2008), p. 252-263.
20. Thakkar J., Kanade A., Alur R.: A transducer-based algorithmic verification of retransmission protocols over noisy channels. Proc. of IFIP Joint International Conference on Formal Techniques for Distributed Systems, LNCS, **7892** (2013), p. 209-224.
21. Vardi M.Y., Wolper P.: Reasoning about infinite computations. Information and Computation. **115** (1994), p. 137.
22. Veanes M., Hooimeijer P., Livshits B., et al.: Symbolic finite state transducers: algorithms and applications. Proc. of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM SIGPLAN Notices. **147** (2012), p. 137-150.
23. Weber A.: Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing. **22** (1993), p. 175-202.
24. Wolper P., Boigelot B.: Verifying systems with innite but regular state spaces. Proc. 10th Int. Conf. on Computer Aided Verication (CAV-1998). LNCS. **1427** (1998), p. 8897.
25. Zakharov V.A.: Equivalence checking problem for finite state transducers over semigroups. Proc. of the 6-th International Conference on Algebraic Informatics (CAI-2015). LNCS. **9270** (2015), p. 208-221.

Computing Bisimulation-Based Comparisons

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. By using the idea of Henzinger et al. for computing the similarity relation, we give an efficient algorithm, with complexity $O((m+n)n)$, for computing the largest bisimulation-based auto-comparison and the directed similarity relation of a labeled graph for the setting without counting successors, where m is the number of edges and n is the number of vertices. Moreover, we provide the first algorithm with a polynomial time complexity, $O((m+n)^2n^2)$, for computing such relations but for the setting with counting successors (like the case with graded modalities in modal logics and qualified number restrictions in description logics).

1 Introduction

In a transition system, bisimilarity between states is an equivalence relation defined inductively as follows: x and x' are bisimilar if they have the same label, each transition from x to a state y can be simulated by a transition from x' to a state y' that is bisimilar to y, and vice versa (i.e., each transition from x' to a state y' can be simulated by a transition from x to a state y that is bisimilar to y') [5]. In modal logic, a Kripke model can be treated as a transition system, and the characterization of bisimilarity is as follows: two states are bisimilar iff they cannot be distinguished by any formula. The "only if" implication is called the invariance and does not require additional conditions, while the whole assertion is called the Hennessy-Milner property and holds for modally saturated models (including finitely-branching models) [10, 5, 1].

Simulation between states in a transition system is a pre-order defined inductively as follows: x' simulates x if they have the same label and, for each transition from x to a state y, there exists a transition from x' to a state y' that simulates y. Notice the lack of the backward direction. Similarity is an equivalence relation defined as follows: x and x' are similar if x simulates x' and vice versa. In modal logic, the characterization of simulation is as follows: x' simulates x iff x' satisfies all existential formulas (in negation normal form) satisfied by x (see, e.g., [1]). The "only if" implication is a kind of preservation and does not require additional conditions, while the whole assertion can also be called the Hennessy-Milner property and holds for modally saturated models.

Now, consider the pre-order $\lesssim$ between states in a transition system defined inductively as follows, $x \lesssim x'$ if: the label of x is a subset of the label of x';

for each transition from x to a state y, there exists a transition from x' to a state y' such that $y \lesssim y'$; and conversely, for each transition from x' to a state y', there exists a transition from x to a state y such that $y \lesssim y'$. In the context of description logic, such a relation is called the largest bisimulation-based auto-comparison of the considered interpretation [4, 3]. The relation $\simeq$, defined by $x \simeq x'$ iff $x \lesssim x'$ and $x' \lesssim x$, is an equivalence relation that can be called the directed similarity relation.[1] The characterization of $\lesssim$ is as follows: $x \lesssim x'$ iff x' satisfies all semi-positive formulas satisfied by x. This was proved for modally saturated interpretations in some description logics [4, 3]. The "only if" implication was proved earlier for some modal logics [8].

The above mentioned notions and their characterizations can be formulated appropriately for different kinds of transition systems and different variants or extensions of modal logic. For example, when transitions are labeled, the transitions used in each of the mentioned conditions should have the same label. This corresponds to the case of multimodal logics and description logics. Extensions of the basic description logic $\mathcal{ALC}$ may allow additional concept constructors, which may require more conditions for bisimulation or bisimulation-based comparison [3]. Some of such constructors are qualified number restrictions, which correspond to graded modalities in graded modal logics. In this work, the case with these constructors is called the setting with counting successors, and in this case, we use the symbol $\lesssim_c$ instead of $\lesssim$ (the latter is used for the setting without counting successors).

In this work, we consider bisimulation-based comparison and directed similarity, formulated for (finite) labeled graphs, and the objective is to provide efficient algorithms for computing the largest bisimulation-based auto-comparison (and hence also the directed similarity relation) of a labeled graph for two settings, with or without counting successors.

The related work is as follows. The Paige-Tarjan algorithms for computing bisimilarity [9] are currently the most efficient ones, with complexity $O((m+n)n)$, where m is the number of transitions and n the number of states. They were originally formulated for graphs w.r.t. both the settings with or without counting successors, but can be reformulated or extended for other contexts (like transition systems, Kripke models, or interpretations in description logics). It exploits the idea of Hopcroft's automaton minimization algorithm [7], but makes a generalization to deal with nondeterministic finite automata instead of deterministic ones. The currently most efficient algorithms, with complexity $O((m+n)n)$, for computing the similarity relation were first provided by Bloom and Paige [2] and by Henzinger et al. [6]. The former was formulated for transition systems and the latter was formulated for graphs. They are both devoted to the setting without counting successors. Divroodi [3] provided a simple algorithm for computing the largest bisimulation-based auto-comparison of an interpretation in a number of description logics.[2] It is not efficient, having a high

[1] The term "bisimulation-based comparison" of [4, 3] is a synonym of the term "directed simulation" of [8], and the term "directed similarity" is named in this spirit.

[2] It is like Algorithm 1 given on page 5 for the setting without counting successors.

polynomial time complexity for the case without counting successors, and an exponential time complexity for the case with counting successors (i.e., qualified number restrictions).

In this work, by using the idea of Henzinger et al. [6] for computing the similarity relation, we give an efficient algorithm, with complexity $O((m+n)n)$, for computing the largest bisimulation-based auto-comparison and the directed similarity relation of a labeled graph for the setting without counting successors. Moreover, we provide the first algorithm with a polynomial time complexity, $O((m+n)^2n^2)$, for computing such relations but for the setting with counting successors.

The rest of this paper is structured as follows. Section 2 formally introduces some notions. In Sections 3 and 4, we present our algorithms for the settings with or without counting successors, respectively, justify their correctness and analyze their complexity. Section 5 concludes this work.

2 Bisimulation-Based Comparisons

A *(finite) labeled graph* is a tuple $G = \langle V, E, A, L\rangle$, where V is a finite set of vertices, $E \subseteq V^2$ is a set of edges, A is a finite set of labels, and L is a function that maps each vertex to a subset of A.

From now on, let $G = \langle V, E, A, L\rangle$ be a given labeled graph.

For a binary relation R, we write $R(x, y)$ to denote $\langle x, y\rangle \in R$. For $x \in V$, we denote $post(x) = \{y \in V \mid E(x, y)\}$ and $pre(x) = \{y \in V \mid E(y, x)\}$.

A relation $Z \subseteq V^2$ is called a *bisimulation-based auto-comparison of G without counting successors* if it satisfies the following conditions for every $x, y, x', y' \in V$:

$$Z(x, x') \Rightarrow L(x) \subseteq L(x') \tag{1}$$

$$Z(x, x') \wedge y \in post(x) \Rightarrow \exists y' \in post(x')\ Z(y, y') \tag{2}$$

$$Z(x, x') \wedge y' \in post(x') \Rightarrow \exists y \in post(x)\ Z(y, y'). \tag{3}$$

A relation $Z \subseteq V^2$ is called a *bisimulation-based auto-comparison of G with counting successors* if it satisfies (1) and, for every $x, x' \in V$,

$$\begin{array}{l}\text{if } Z(x, x') \text{ holds, then there exists a bijection}\\ h : post(x) \to post(x') \text{ such that } h \subseteq Z.\end{array} \tag{4}$$

We write $x \lesssim x'$ (resp. $x \lesssim_c x'$) to denote that there exists a bisimulation-based auto-comparison Z of G without (resp. with) counting successors such that $Z(x, x')$ holds. Observe that both $\lesssim$ and $\lesssim_c$ are pre-orders, and $\lesssim_c$ is stronger than $\lesssim$ (i.e., the former is a subset of the latter). It can be shown that the relation $\lesssim$ (resp. $\lesssim_c$) is the largest (w.r.t. $\subseteq$) bisimulation-based auto-comparison of G without (resp. with) counting successors.

We write $x \simeq x'$ to denote that $x \lesssim x'$ and $x' \lesssim x$. Similarly, we write $x \simeq_c x'$ to denote that $x \lesssim_c x'$ and $x' \lesssim_c x$. We call $\simeq$ (resp. $\simeq_c$) the *directed similarity relation of G without (resp. with) counting successors*. Both $\simeq$ and $\simeq_c$ are equivalence relations.

3 The Case without Counting Successors

In this section, we present three algorithms for computing the largest bisimulation-based auto-comparison of a given labeled graph $G = \langle V, E, A, L\rangle$ in the setting without counting successors. The first two algorithms, SchematicComparison and RefinedComparison, are used to help to understand the last one, EfficientComparison. The idea of the EfficientComparison algorithm and the explanation via two simpler algorithms follow from the ones by Henzinger et al. for computing simulations [6].

All the three mentioned algorithms compute the sets $leq(v)$ and $geq(v)$ for $v \in V$, where $leq(v) = \{u \in V \mid u \lesssim v\}$ and $geq(v) = \{u \in V \mid v \lesssim u\}$ with $\lesssim$ being the relation defined in Section 2. For the explanations given in the current section, however, by $\lesssim$ we denote the following relation, which depends on and changes together with leq:

$$\{\langle u, v\rangle \in V^2 \mid u \in leq(v)\}. \tag{5}$$

The SchematicComparison algorithm (on page 5) initializes the sets $leq(v)$ and $geq(v)$, for $v \in V$, to satisfy the condition (1) with Z replaced by $\lesssim$. Then, while the condition (2) (resp. (3)) with Z replaced by $\lesssim$ and x, y, x' (resp. x', y', x) replaced by u, v, w is not satisfied, it updates the mappings leq and geq appropriately in order to delete the pair $\langle u, w\rangle$ (resp. $\langle w, u\rangle$) from $\lesssim$. It is easy to see that at the end the relation $\lesssim$ specified by (5) is the largest bisimulation-based auto-comparison of G without counting successors. That is, the SchematicComparison algorithm is correct.

The RefinedComparison algorithm (on page 5) refines the SchematicComparison algorithm by using two additional mappings $prevGeq$ and $prevLeq$, which approximate the mappings geq and leq, respectively. The mappings geq and leq are refined by reducing the sets $geq(v)$ and $leq(v)$, for $v \in V$, in each iteration of the "repeat" loop. The set $prevGeq(v)$ (resp. $prevLeq(v)$) stores the value of $geq(v)$ (resp. $leq(v)$) at some earlier iteration and is a superset of $geq(v)$ (resp. $leq(v)$). The refinement is done by making updates only for $w \in pre(prevGeq(v)) \setminus pre(geq(v))$ (resp. $w \in pre(prevLeq(v)) \setminus pre(leq(v))$) instead of for $w \in pre(geq(v))$ (resp. $pre(leq(v))$) as in the SchematicComparison algorithm. It is straightforward to check that the conditions I1, I2, I3 listed in the RefinedComparison algorithm are invariants of the "repeat" loop. When the algorithm terminates, we have $geq(v) = prevGeq(v)$ and $leq(v) = prevLeq(v)$ for all $v \in V$, and hence the relation $\lesssim$ specified by (5) satisfies the conditions (1)–(3) and is a bisimulation-based auto-comparison of G without counting successors. It is the largest one because the deletions of elements from the sets $geq(v)$ and $leq(v)$, for $v \in V$, are appropriate and done only when needed. Therefore, the RefinedComparison algorithm is correct.

The EfficientComparison algorithm (on page 6) modifies the RefinedComparison algorithm in that instead of maintaining the sets $prevGeq(v)$ and $prevLeq(v)$, for $v \in V$, it maintains the sets

$$removeGeq(v) = pre(prevGeq(v)) \setminus pre(geq(v)), \tag{6}$$

Algorithm 1: SchematicComparison

input : a labeled graph $G = \langle V, E, A, L \rangle$.
output : for each $v \in V$, the sets $geq(v)$ and $leq(v)$.

foreach $v \in V$ **do**
  $geq(v) := \{u \in V \mid L(v) \subseteq L(u)\}$;
  $leq(v) := \{u \in V \mid L(u) \subseteq L(v)\}$;
repeat
  // **assert:** I0: $\forall u, w \in V\ w \in geq(u) \leftrightarrow u \in leq(w)$
  if *there are* $u, v, w \in V$ *such that* $v \in post(u)$, $w \in geq(u)$ *and* $post(w) \cap geq(v) = \emptyset$ **then**
    $geq(u) := geq(u) \setminus \{w\}$, $leq(w) := leq(w) \setminus \{u\}$;
  if *there are* $u, v, w \in V$ *such that* $v \in post(u)$, $w \in leq(u)$ *and* $post(w) \cap leq(v) = \emptyset$ **then**
    $leq(u) := leq(u) \setminus \{w\}$, $geq(w) := geq(w) \setminus \{u\}$;
until *no change occurred during the last iteration*;

Algorithm 2: RefinedComparison

foreach $v \in V$ **do**
  $prevGeq(v) := V$, $prevLeq(v) := V$;
  if $post(v) = \emptyset$ **then**
    $geq(v) := \{u \in V \mid L(v) \subseteq L(u)$ and $post(u) = \emptyset\}$;
    $leq(v) := \{u \in V \mid L(u) \subseteq L(v)$ and $post(u) = \emptyset\}$;
  else
    $geq(v) := \{u \in V \mid L(v) \subseteq L(u)$ and $post(u) \neq \emptyset\}$;
    $leq(v) := \{u \in V \mid L(u) \subseteq L(v)$ and $post(u) \neq \emptyset\}$;
repeat
  // **assert:**
  // I1: $\forall v \in V\ geq(v) \subseteq prevGeq(v) \wedge leq(v) \subseteq prevLeq(v)$
  // I2: $\forall u, v, w \in V\ v \in post(u) \wedge w \in geq(u) \rightarrow post(w) \cap prevGeq(v) \neq \emptyset$
  // I3: $\forall u, v, w \in V\ v \in post(u) \wedge w \in leq(u) \rightarrow post(w) \cap prevLeq(v) \neq \emptyset$
  if *there exists* $v \in V$ *such that* $geq(v) \neq prevGeq(v)$ **then**
    foreach $u \in pre(v)$ *and* $w \in pre(prevGeq(v)) \setminus pre(geq(v))$ **do**
      if $w \in geq(u)$ **then**
        $geq(u) := geq(u) \setminus \{w\}$, $leq(w) := leq(w) \setminus \{u\}$;
    $prevGeq(v) := geq(v)$;
  if *there exists* $v \in V$ *such that* $leq(v) \neq prevLeq(v)$ **then**
    foreach $u \in pre(v)$ *and* $w \in pre(prevLeq(v)) \setminus pre(leq(v))$ **do**
      if $w \in leq(u)$ **then**
        $leq(u) := leq(u) \setminus \{w\}$, $geq(w) := geq(w) \setminus \{u\}$;
    $prevLeq(v) := leq(v)$;
until *no change occurred during the last iteration*;

Algorithm 3: EfficientComparison

```
1  foreach v ∈ V do
       // prevGeq(v) := V ,   prevLeq(v) := V
2      if post(v) = ∅ then
3          geq(v) := {u ∈ V | L(v) ⊆ L(u) and post(u) = ∅};
4          leq(v) := {u ∈ V | L(u) ⊆ L(v) and post(u) = ∅};
5      else
6          geq(v) := {u ∈ V | L(v) ⊆ L(u) and post(u) ≠ ∅};
7          leq(v) := {u ∈ V | L(u) ⊆ L(v) and post(u) ≠ ∅};
8      removeGeq(v) := pre(V) \ pre(geq(v));
9      removeLeq(v) := pre(V) \ pre(leq(v));
10 repeat
       // assert:
       // I4: ∀v ∈ V removeGeq(v) = pre(prevGeq(v)) \ pre(geq(v))
       // I5: ∀v ∈ V removeLeq(v) = pre(prevLeq(v)) \ pre(leq(v))
11     if there exists v ∈ V such that removeGeq(v) ≠ ∅ then
12         foreach u ∈ pre(v) and w ∈ removeGeq(v) do
13             if w ∈ geq(u) then
14                 geq(u) := geq(u) \ {w},  leq(w) := leq(w) \ {u};
15                 foreach w′ ∈ pre(w) do
16                     if post(w′) ∩ geq(u) = ∅ then
17                         removeGeq(u) := removeGeq(u) ∪ {w′};
18                 foreach u′ ∈ pre(u) do
19                     if post(u′) ∩ leq(w) = ∅ then
20                         removeLeq(w) := removeLeq(w) ∪ {u′};
           // prevGeq(v) := geq(v)
21         removeGeq(v) := ∅;
22     if there exists v ∈ V such that removeLeq(v) ≠ ∅ then
23         foreach u ∈ pre(v) and w ∈ removeLeq(v) do
24             if w ∈ leq(u) then
25                 leq(u) := leq(u) \ {w},  geq(w) := geq(w) \ {u};
26                 foreach w′ ∈ pre(w) do
27                     if post(w′) ∩ leq(u) = ∅ then
28                         removeLeq(u) := removeLeq(u) ∪ {w′};
29                 foreach u′ ∈ pre(u) do
30                     if post(u′) ∩ geq(w) = ∅ then
31                         removeGeq(w) := removeGeq(w) ∪ {u′};
           // prevLeq(v) := leq(v)
32         removeLeq(v) := ∅;
33 until no change occurred during the last iteration;
```

$$removeLeq(v) = pre(prevLeq(v)) \setminus pre(leq(v)). \tag{7}$$

It can be checked that the equivalences (6) and (7) are invariants of the "repeat" loop if the sets $prevGeq(v)$ and $prevLeq(v)$ are computed as in the RefinedComparison algorithm by uncommenting the corresponding lines. Thus, the EfficientComparison algorithm reflects the RefinedComparison algorithm and is correct.

We use the same idea and technique of [6] for analyzing the complexity of the EfficientComparison algorithm. Let $n = |V|$, $m = |E|$ and assume that $|A|$ is a constant. We also assume that the algorithm is modified by using and maintaining two arrays $countPostGeq[1..n, 1..n]$ and $countPostLeq[1..n, 1..n]$ of natural numbers such that $countPostGeq[w', u] = |post(w') \cap geq(u)|$ and $countPostLeq[w', u] = |post(w') \cap leq(u)|$ for all vertices w' and u from V. These arrays are initialized in time $O((m + n)n)$. Whenever a vertex w is removed from $geq(u)$ (resp. $leq(u)$), the counters $countPostGeq[w', u]$ (resp. $countPostLeq[w', u]$) are decremented for all predecessors w' of w. The costs of these decrements is absorbed in the cost of the "if" statements at the lines 16, 19, 27, 30 of the algorithm. Using the arrays $countPostGeq$ and $countPostLeq$, the test $post(w') \cap geq(u) = \emptyset$ at the line 16 and the similar ones at the lines 19, 27, 30 of those "if" statements can be executed in constant time (e.g., by checking if $countPostGeq[w', u] = 0$ for the case of the line 16).

The initialization of $geq(v)$ and $leq(v)$ for all $v \in V$ requires time $O(n^2)$. The initialization of $removeGeq(v)$ and $removeLeq(v)$ for all $v \in V$ requires time $O((m + n)n)$. Given two vertices v and w, if the test $w \in removeGeq(v)$ at the line 12 is positive in iteration i of the "repeat" loop, then that test is negative in all the iterations $j > i$. This is due to the invariant I4 and that

- in all the iterations, $w \in removeGeq(v)$ implies that $w \notin pre(geq(v))$,
- the value of $prevGeq(v)$ in all the iterations $j > i$ is a subset of the value of $geq(v)$ in the iteration i.

It follows that the test $w \in geq(u)$ at the line 13 is executed $\Sigma_v \Sigma_w |pre(v)| = O((m+n)n)$ times. This test is positive at most once for every w and u, because after a positive test w is removed from $geq(u)$ and never put back. Thus, the body of the "if" statement at the line 13 contributes time $\Sigma_w \Sigma_u (1 + |pre(w)| + |pre(u)|) = O((m + n)n)$. This implies that the "if" statement at the line 11 contributes time $O((m + n)n)$. Similarly, the "if" statement at the line 22 contributes time $O((m + n)n)$. Summing up, the (modified) EfficientComparison algorithm runs in time $O((m + n)n)$. We arrive at:

Theorem 3.1. *Given a labeled graph G with n vertices and m edges, the largest bisimulation-based auto-comparison of G and the directed similarity relation of G in the setting without counting successors can be computed in time $O((m+n)n)$.*

4 The Case with Counting Successors

In this section, we present the ComparisonWithCountingSuccessors algorithm (on page 8) for computing the largest bisimulation-based auto-comparison of

a given labeled graph $G = \langle V, E, A, L\rangle$ in the setting with counting successors. It uses the following data structures:

- $leq \subseteq V^2$,
- $remove \subseteq V^2$,
- $f : V^3 \to V$ is a partial mapping,
- $g : V^3 \to V$ is a partial mapping.

Algorithm 4: ComparisonWithCountingSuccessors

```
   input  : a labeled graph G = ⟨V, E, A, L⟩.
   output : the relation leq.
 1 set leq, remove to empty sets and f, g to empty mappings;
 2 foreach u, u' ∈ V do
 3     if L(u) ⊆ L(u') and |post(u)| = |post(u')| then
 4         leq := leq ∪ {⟨u, u'⟩},  W := post(u');
 5         foreach v ∈ post(u) do
 6             extract v' from W,  f(u, v, u') := v',  g(u', v', u) := v;
 7     else remove := remove ∪ {⟨u, u'⟩};
 8 while remove ≠ ∅ do
 9     extract (v, v') from remove;
10     foreach u ∈ pre(v) and u' ∈ pre(v') such that f(u, v, u') = v' do
11         undefine f(u, v, u') and g(u', v', u);
12         set g' to the empty mapping;
13         S := {v},  T' := ∅;
14         repeat
15             S' := {w' ∈ post(u') | ∃w ∈ S ⟨w, w'⟩ ∈ leq} \ T';
16             foreach w' ∈ S' do
17                 set g'(w') to an element w of S such that ⟨w, w'⟩ ∈ leq;
18             S := {g(u', w', u) | w' ∈ S' and w' ≠ v'},  T' := T' ∪ S';
19         until S' = ∅ or v' ∈ S';
20         if S' = ∅ then
21             delete the pair ⟨u, u'⟩ from leq;
22             undefine f(u, w, u') and g(u', w', u) for any w, w';
23             remove := remove ∪ {⟨u, u'⟩};
24             break;
25         w' := v';
26         repeat
27             w := g'(w'),  w'' := f(u, w, u');
28             f(u, w, u') := w',  g(u', w', u) := w,  w' := w'';
29         until w = v;
```

The variable *leq* will keep the relation to be computed $\lesssim_c$. As an invariant, *leq* is a superset of $\lesssim_c$. A pair $\langle u, u'\rangle$ is deleted from *leq* and inserted into the set

remove only when we already know that $u \not\lesssim_c u'$. At the beginning (in the steps 1–7), the relation *leq* is initialized to $\{\langle u, u' \rangle \in V^2 \mid L(u) \subseteq L(u') \wedge |post(u)| = |post(u')|\}$ and the relation *remove* is initialized to $V^2 \setminus leq$. Then, during the (main) "while" loop, we refine the relation *leq* by extracting and processing each pair $\langle v, v' \rangle$ from the relation *remove*. As invariants of that main loop, for all $u, w, u', w' \in V$,

$$leq \cap remove = \emptyset, \tag{8}$$

$$f(u, w, u') = w' \Rightarrow \langle u, w \rangle \in E \wedge \langle u, u' \rangle \in leq \wedge \langle u', w' \rangle \in E \wedge \langle w, w' \rangle \in leq \cup remove, \tag{9}$$

$$g(u', w', u) = w \Leftrightarrow f(u, w, u') = w', \tag{10}$$

$$\text{if } leq(u, u') \text{ holds, then the function } \lambda w \in post(u).f(u, w, u') \text{ is well-defined and is a bijection between } post(u) \text{ and } post(u') \text{ and } \{\langle w, w' \rangle \mid w \in post(u), w' = f(u, w, u')\} \subseteq leq \cup remove. \tag{11}$$

Processing a pair $\langle v, v' \rangle$ extracted from the set *remove* is done as follows. For each $u \in pre(v)$ and $u' \in pre(v')$ such that $f(u, v, u') = v'$, we want to repair the values of the data structures so that the above invariants still hold. We first undefine $f(u, v, u')$ and $g(u', v', u)$ (at the line 11). Figure 1 illustrates the "repeat" loop at the lines 14–19 and its initialization at the line 13: after the initialization, $S = \{v\}$ and $T' = \emptyset$; after the first iteration, $S' = S'_1$, $S = S_1$ and $T' = S'_1$; after the second iteration, $S' = S'_2$, $S = S_2$ and $T' = S'_1 \cup S'_2$; observe that $|S_1| = |S'_1|$ and $|S_2| = |S'_2|$. Since S' is disjoint with the value of T' in the previous iteration and T' is monotonically extended by S', the loop will terminate with $S' = \emptyset$ or $v' \in S'$.

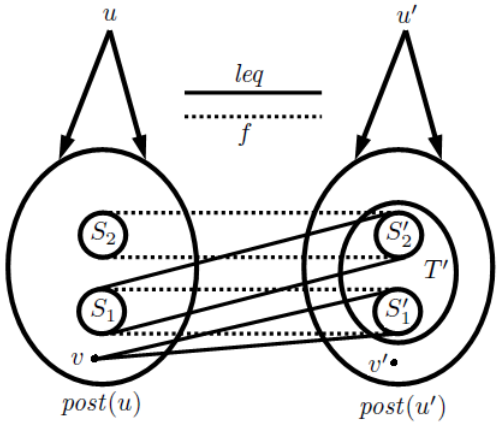


Fig. 1. An illustration for the steps 14–19 of Algorithm 4.

In the case $S' = \emptyset$, for $T = \{v\} \cup \{w \in post(u) \mid \exists w' \in T'\ f(u, w, u') = w\}$, we have that $|T| = |T'| + 1$ and $T' = \{w' \in post(u') \mid \exists w \in T\ \langle w, w' \rangle \in leq\}$ (to see this, one can use Figure 1 to help the imagination), and as a consequence, $u \not\lesssim_c u'$, because otherwise the condition (4) with Z replaced by $\lesssim_c$ cannot hold (recall that *leq* is a superset of $\lesssim_c$). So, in the case $S' = \emptyset$, we delete the pair $\langle u, u' \rangle$ from the relation *leq*, add it to the relation *remove*, and modify the mappings f and g appropriately.

In the case $v' \in S'$, the mappings f and g are repaired at the steps 25–29, using the mapping g' initialized at the step 12 and updated at the step 17. This is illustrated in Figure 2, where the dotted arrows from $post(u)$ to $post(u')$ represent some pairs $\langle w, f(u, w, u') \rangle$ such that $f(u, w, u') \in T'$ w.r.t. the old (i.e., previous) value of f, and the normal arrows from $post(u')$ to $post(u)$ represent some pairs $\langle w', g'(w') \rangle$ such that $w' \in T'$. The repair relies on updating f so that those dotted arrows are replaced by the inverse of those normal arrows, and updating g to satisfy the invariant (10).

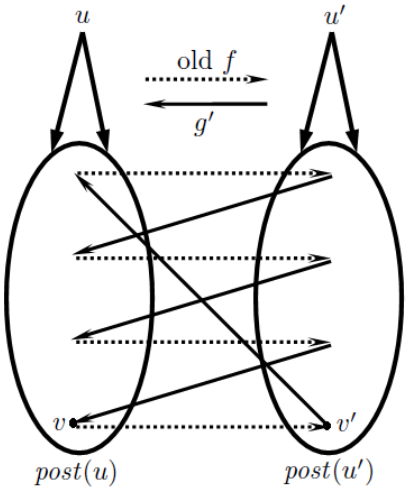


Fig. 2. An illustration for the steps 25–29 of Algorithm 4.

Technically, it can be checked that the conditions (8)–(11) are invariants of the (main) "while" loop. When the loop terminates, we have that $remove = \emptyset$, and the invariants (9) and (11) guarantee that *leq* satisfies the condition (4) with Z replaced by *leq*. As the condition (1) with Z replaced by *leq* holds due to the initialization of *leq*, it follows that the resulting relation *leq* is a bisimulation-based auto-comparison of the given labeled graph G in the setting with counting successors. As each pair $\langle u, u' \rangle$ deleted from *leq* during the computation satisfies $u \not\lesssim_c u'$, we conclude that the resulting relation *leq* is the largest bisimulation-based auto-comparison of G with counting successors.

Let $n = |V|$, $m = |E|$ and assume that $|A|$ is a constant. Our complexity analysis for the ComparisonWithCountingSuccessors algorithm is straightforward. The steps 1–7 for initialization run in time $O((m + n)n)$. The body of the "foreach" loop at the line 10 runs no more than $(m + n)^2$ times totally. The "repeat" loop at the lines 14–19 runs in time $O(n^2)$. The steps 25–29 run in time $O(n)$. Summing up, the algorithm runs in time $O((m + n)^2n^2)$. We arrive at:

Theorem 4.1. *Given a labeled graph G with n vertices and m edges, the largest bisimulation-based auto-comparison of G and the directed similarity relation of G in the setting with counting successors can be computed in time $O((m+n)^2n^2)$.*

5 Conclusions

By using the idea of Henzinger et al. [6] for computing the similarity relation, we have given an efficient algorithm, with complexity $O((m + n)n)$, for computing the largest bisimulation-based auto-comparison and the directed similarity relation of a labeled graph for the setting without counting successors. Moreover, we have provided the first algorithm with a polynomial time complexity, $O((m + n)^2n^2)$, for computing such relations but for the setting with counting successors. One can adapt this latter algorithm to obtain the first algorithm with a polynomial time complexity for computing the similarity relation in the setting with counting successors. Our algorithms can also be reformulated and extended for interpretations in various modal and description logics (instead of labeled graphs).

Acknowledgements

This work was done in cooperation with the project 2011/02/A/HS1/00395, which is supported by the Polish National Science Centre (NCN).

References

1. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.
2. B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comput. Program.*, 24(3):189–220, 1995.
3. A.R. Divroodi. *Bisimulation Equivalence in Description Logics and Its Applications*. PhD thesis, University of Warsaw, 2015. Available at `http://www.mimuw.edu.pl/wiadomosci/aktualnosci/doktoraty/pliki/ali_rezaei_divroodi/ad-dok.pdf`.
4. A.R. Divroodi and L.A. Nguyen. Bisimulation-based comparisons for interpretations in description logics. In *Proceedings of Description Logics'2013*, volume 1014 of *CEUR Workshop Proceedings*, pages 652–669. CEUR-WS.org, 2013.
5. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

6. M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of FOCS'1995*, pages 453–462. IEEE Computer Society, 1995.
7. J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. Available at `ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/71/190/CS-TR-71-190.pdf`, 1971.
8. N. Kurtonina and M. de Rijke. Simulating without negation. *J. Log. Comput.*, 7(4):501–522, 1997.
9. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.
10. J. van Benthem. *Modal Correspondence Theory*. PhD thesis, Mathematisch Instituut & Instituut voor Grondslagenonderzoek, University of Amsterdam, 1976.

Towards model checking argumentative dialogues with emotional reasoning

(Extended abstract)

Magdalena Kacprzak[1], Anna Sawicka[2], and Andrzej Zbrzezny[3]

[1] Bialystok University of Technology, Bialystok, Poland
[2] Polish-Japanese Academy of Information Technology, Warsaw, Poland
[3] Jan Długosz University, Czestochowa, Poland
m.kacprzak@pb.edu.pl, asawicka@pja.edu.pl, a.zbrzezny@ajd.czest.pl

Abstract. In the paper, we show a formal model for a dialogue game in which players can perform actions representing locutions like *claim*, *question*, *concede* as well as locutions which have a greater emotional charge like *scold* or *nod*. We define a protocol for dialogues in which participants have emotional skills and then give an interpreted system for them. Finally, we propose an extension of CTL logic with commitment, emotion and goal modalities. All of this is a formal basis, which we use to perform semantic verification of properties of dialogue systems with emotional reasoning.

1 Introduction

As members of society we have the need of collective work and protocols are an important part of our social skills. They help to facilitate structured conversations and are commonly used in everyday situations (sometimes unconsciously). Protocols should support dialogue, to help achieve the goal of the conversation. The first group of our interest are children who are known to have different cognitive abilities in comparison to adults. Usually, they are hard to convince and such a discourse is quite specific. For many people, it is hard to tell at first sight, from which argument we would benefit the most, and which one we should definitely avoid.

The aim of such an argumentation is a change in the emotional state of the interlocutor. The emotional state of our understanding consists of many factors, e.g. a sense of security, self-agency, self-satisfaction, self-confidence and so on. Some argument at the same time can increase one's sense of self-agency, but decrease one's sense of security ("if you find a job, you could move out, but you would have to rent your own flat"). We aim in designing an application which would be a support for people who have to convince somebody, but the more important factor is the emotional well-being of the interlocutor. We see such an application as a trainer of good practices in argumentation. We could consider possible reactions of potential interlocutor (e.g. a rebellious teenager, an expatriate) to specific arguments and monitor changes in the simplified representation of the emotional state. That is the reason we work on argumentative dialogue protocol, which is supposed to take into account change an emotional state of interlocutor in order to obtain the desired result (e.g. some kind of decision).

Usually, the aim of argumentation is figuring the agreement, the conviction of someone for their own reasons or even reaching a compromise [14, 32]. Persuasion dialogues are dialogues aimed at resolving conflicts of opinion between at least two participants. There are many types of such dialogues, e.g. *conflict resolution* dialogue begins with a conflict of opinion and ends when one of the participants convinces the other one. By the contrast, the argumentation under our consideration does not necessarily have to convince a child to do something, but it should help him become aware of his feelings. Certainly, we do not want to claim that there is an obvious argumentation that will convince everybody, but there are some argumentation strategies and mechanisms, which are quite known and considered as convincing ones.

Formal dialogue systems, which are growing field in the research on the process of communication, can be used as a schema for such dialogues, both between artificial agents or between the man and the machine. In our case, we need an argumentative dialogue model designed for human-computer communication, which applies the mentioned specific types of dialogues. There are other approaches which are focused rather on the agent to agent communication [4].

In this paper, we present a continuation of our work on the mathematical model of dialogue inspired by dialogue games [16]. We would like to use this model as a semantic structure in verification of properties of dialogue protocols and enable automated analysis of dialogues. The model we base our current research on is founded on the tradition of argumentative dialogue games by Prakken and others [25].

There are many approaches that assume very strict rules of communications. Our model, focusing on machine-man communication, is also based on such strict rules. On one hand, it makes it a little trivial, but on the other hand we can extract and focus on most important features of the dialogues. We can perceive *dialogue games* [7, 14, 26, 31, 33] as examples of such strict dialogues. In dialogue games, a dialogue is treated as some kind of a game played between two parties. Rules of such a game define policies for the communication between parties in order to meet some assumptions, for example in Hamblin system [12, 17] we have rules preventing argumentative mistakes, in Lorenzen system [18, 20] we have rules enabling validation of formulas [15, 34].

Each dialogue game should have three basic categories of rules. *Locution rules* define a set of actions (speech acts, locutions) the player is allowed to perform during the game. These actions express communication intentions of players. For example, rules of the dialogue game can assume that player can claim something, argue, justify, ask for justification, concede something etc. The second category of rules is responsible for the definition of possible answers for specific moves. For example, after one interlocutor claims something, the other one can concede it by performing *concede* or he can ask for justification by performing *why*. These rules are called *structural rules*. The third group of rules defines effects of actions. Due to performing some action (e.g. confirming or rejecting) a set of public declarations (commitments) of the interlocutor is changed. The result of an action is a change in the commitments set of the player, i.e. addition of some new statement to this set. These rules are called *effect rules*. We are specifying above rules which determine available moves for each player at every moment of the dialogue.

Even though every protocol must meet some general requirements, each one can be quite unique and we are interested in verifying characteristic properties of the dialogue defined by the specific protocol. In order to do that, we would like to use model checking method applied in verification of multi-agent systems (MAS). Main solutions in this matter combine bounded model checking (BMC) with symbolic verification using translations to either ordered binary decision diagrams (BDDs) [13] or propositional logic (SAT) [24]. Verified properties are expressed in logics which are combinations of the epistemic logic with branching [27] or linear time temporal logic [30]. Such logics can be interpreted either over interleaved interpreted systems (IIS) [19] or interpreted systems themselves [11]. To interpret the properties of dialogue games we chose IIS, in which only one action at a time is performed in a global transition.

The work presents a sketch of a formal system, which is a base for designing model checking techniques for verification dialogue games. We are concerned with argumentative dialogues, in which players can perform actions affecting commitments as well as their emotions. As a result, they change the emotional state, mood and attitude of the players. The proposed model will be used to show what mechanisms occur in human argumentative dialogues. In particular, we focus on argumentations where rational arguments are less effective (or not as effective) as the arguments referring to the emotions. On this basis, we will build a tool for learning managing emotions. Since emotions play a major role in persuading children, this tool can be used for personal development training for teachers or parents, which are often confused about children's feelings.

The study of emotions is part of various disciplines like Psychology, Economics, Cognitive Neuroscience, and, in recent years, also Artificial Intelligence and Computer Science. These studies aim to establish systems for emotional interaction. Nowadays, more and more artificial agents integrate emotional skills to achieve expressiveness, adaptability, and credibility. Such multiagent systems find application in the improvement of human-machine interaction, testing, refining and developing an emotional hypothesis or even the improvement of artificial intelligence techniques, once it optimizes decision-making mechanisms [28, 23].

2 Interpreted system

We start out by defining a mathematical model for argumentation dialogue games. This model uses the concept of interpreted systems and Kripke structures. In this model formulas of a modal logic adequate to express properties that allow prediction of players' behavior are interpreted. The obtained Kripke structure will be used to perform automatic verification of dialogue protocols via model checking techniques.

First, we assume that the set of players of a dialogue game consists of two players: White (W) and Black (B), $Pl = \{W, B\}$. To each player $p \in Pl$, we assign a set of actions Act_p and a set of possible local states L_p.

Every action from Act_p can influence participant's commitments. We assume that the set Act_p contains also the special empty (null) action ε. Every action (except null action) is synonymous with locution expressed by the specific player. Results of locutions are determined by *evolution function* and are specified afterwards.

Player's local state $l_p \in L_p$ consists of the player's *commitments*, *emotions*, and *goals*, $l_p = (C_p, E_p, GO_p)$. Player's commitments and goals are elements of a fixed topic language, which allows expressing the content of locutions. Thus, C_p and GO_p are sets of such expressions. These sets may be subject to change after a player's action. More specifically, the player can add or delete the selected expression. Emotions which we consider are fear, disgust, joy, sadness, and anger. Their strength (intensity) is represented by natural numbers from the set $\{1,2,\ldots,10\}$. Thus, E_p is a 5-tuple consisting of five values, which may also change after a certain action. It is worth highlighting here that a change in the intensity of the emotions is dependent on the type of locution and, perhaps even more, on its content.

Next, Act denotes the Cartesian product of the players' actions, i.e. $Act = Act_W \times Act_B$. The global action $a \in Act$ is a pair of actions $a = (a_W, a_B)$, where $a_W \in Act_W$, $a_B \in Act_B$ and at least one of these actions is the empty action. This means that players cannot speak at the same time. Moreover, a player cannot reply to his own moves. Thus, the empty action is performed alternately by players W and B.

Also, we need to order performed global actions and indicate which actions correspond with which ones and therefore we define *double-numbered global actions* set $Num_2Act = \mathbb{N} \times \mathbb{N} \times Act$. During the dialogue, we assign to each performed global action two numbers: the first one (ascending) indicates order (starting from the value 1). The second one points out to which earlier action this action is referring (0 at the beginning of the dialogue means that we are not referring to any move).

Furthermore, we define *numbered global actions* set $Num_1Act = \mathbb{N} \times Act$. Each element of this set is a pair (n,a) consisting of an action $a \in Act$ and the identifier of the action it refers to, $n \in \mathbb{N}$. If we want to find out whether we can use some global action one more time, we should check if the possible move containing the same global action refers to the different earlier move. We define function $Denum : Num_2Act \rightarrow Num_1Act$, which maps double-numbered global action to the numbered global action. We understand dialogue d as a sequence of moves and in particular, we denote $d_{1..n} = d_1, ..., d_n$, where $d_i \in Num_2Act$, $d_i = (i, j, a)$, $j \in \mathbb{N}$, $j < i$, $a \in Act$.

A global state g is a triple consisting of dialogue history and players' local states corresponding to a snapshot of the system at a given time point $g = (d(g), l_W(g), l_B(g))$, $g \in G$ where G is the set of global states. Given a global state g, we denote by $d(g)$ a sequence of moves executed on a way to state g and by $l_p(g)$ - the local state of player p in g.

An *interpreted system* for a dialogue game is a tuple $IS = (I, \{L_p, Act_p\}_{p \in Pl})$ where $I \subseteq G$ is the set of initial global states.

Let $\alpha, \beta, \varphi, \psi_1, .., \psi_n, \gamma_1, .., \gamma_n \in Form(PV)$, i.e., be formulas defined over the set PV, which is a set of atomic propositions under which a content of speech acts is specified. Locutions used in players' actions are the same for both players: $Act_W = Act_B = \{\varepsilon,$ *claim* φ, *concede* φ, *why* φ, *scold* φ, *nod* φ, φ *since* $\{\psi_1, \ldots, \psi_n\}$, *retract* φ, *question* φ $\}$.

In argumentation dialogues, a player can *claim* some facts, *concede* with the opponent or change his mind performing action *retract*. To challenge the opponent's statement, he may ask *why*, or ask whether the opponent commits to something, i.e., perform action *question*. For defense he can use the action *since*. It is the kind of reasoning and

argumentation. Actions *scold* and *nod* express reprimand and approval, respectively. Note that all of these locutions refer to commitments, i.e., public announcements. We are not talking here about beliefs or knowledge, which may differ from the commitments.

Now we define *legal answer function* $F_{LA} : Num_2Act \to 2^{Num_1Act}$, which maps a double-numbered action to the set of possible numbered actions. This function is symmetrical for both players and determines for every action a set of legal actions which can be performed next.

- $F_{LA}(i,j,(\varepsilon,\varepsilon)) = \emptyset$,
- $F_{LA}(i,j,(claim\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,why\ \varphi), (\varepsilon,concede\ \varphi), (\varepsilon,claim\ \neg\varphi), (\varepsilon,node\ \psi), (\varepsilon,scold\ \psi)\ \}$, for some $\psi \in Form(PV)$,
- $F_{LA}(i,j,(why\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,\varphi\ since\ \{\psi_1,\ldots,\psi_n\}),(\varepsilon,retract\ \varphi)\}$,
- $F_{LA}(i,j,(\varphi\ since\ \{\psi_1,\ldots,\psi_n\},\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,\ why\ \alpha), (\varepsilon,concede\ \beta), (\varepsilon,\neg\varphi\ since\ \{\gamma_1,\ldots,\gamma_n\}), (\varepsilon,node\ \psi), (\varepsilon,scold\ \psi)\ \}$, where $\alpha \in \{\psi_1,\ldots,\psi_n\}$, $\beta \in \{\varphi,\psi_1,\ldots,\psi_n\}$, and $\psi \in Form(PV)$,
- $F_{LA}(i,j,(concede\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,\varepsilon), (\varepsilon,claim\ \alpha), (\varepsilon,node\ \alpha), (\varepsilon,scold\ \alpha), (\varepsilon,\alpha\ since\ \{\psi_1,\ldots,\psi_n\})\ \}$, for some $\alpha,\psi_1,\ldots,\psi_n \in Form(PV)$,
- $F_{LA}(i,j,(retract\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,\varepsilon), (\varepsilon,claim\ \alpha), (\varepsilon,node\ \alpha), (\varepsilon,scold\ \alpha), (\varepsilon,\alpha\ since\ \{\psi_1,\ldots,\psi_n\})\}$, for some $\alpha,\psi_1,\ldots,\psi_n \in Form(PV)$,
- $F_{LA}(i,j,(question\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,retract\ \varphi), (\varepsilon,claim\ \varphi), (\varepsilon,claim\ \neg\varphi)\}$,
- $F_{LA}(i,j,(scold\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,why\ \varphi), (\varepsilon,concede\ \varphi), (\varepsilon,claim\ \neg\varphi), (\varepsilon,node\ \psi), (\varepsilon,scold\ \psi)\ \}$, for some $\psi \in Form(PV)$,
- $F_{LA}(i,j,(nod\ \varphi,\varepsilon)) = \{(i,act) : act \in \{(\varepsilon,\varepsilon), (\varepsilon,claim\ \alpha), (\varepsilon,node\ \alpha), (\varepsilon,scold\ \alpha), (\varepsilon,\alpha\ since\ \{\psi_1,\ldots,\psi_n\})\ \}$, for some $\alpha,\psi_1,\ldots,\psi_n \in Form(PV)$.

The actions executed by players are selected according to a *protocol function* $Pr : G \to 2^{Num_2Act}$, which maps a global state g to the set of possible double-numbered global actions. The function Pr satisfies the following rules.

(R1) For $\iota \in I$ $Pr(\iota) =$
$\{(1,0,(claim\ \varphi,\ \varepsilon)), (1,0,(question\ \varphi,\varepsilon)), (1,0,(\varphi\ since\ \{\psi_1,\ldots,\psi_n\},\varepsilon))\}$.

(R2) $Pr((d_{1..k-1},(k,l,(\varepsilon,\varepsilon)),l_W(g),l_B(g))) = \{(k+1,numact) : numact \in F_{LA}(k,l,(\varepsilon,\varepsilon))$.

(R3) $Pr((d_{1..k-1},(k,l,(a,\varepsilon)),l_W(g),l_B(g))) = \{(k+1,numact) : numact \in F_{LA}(k,l,(a,\varepsilon))\}$, for $a \in \{\varepsilon,\ claim\ \varphi, scold\ \varphi, why\ \varphi,\ \varphi\ since\ \{\psi_1,\ldots,\psi_n\}\}$.

(R4) $Pr((d_{1..k-1},(k,l,(a,\varepsilon)),l_W(g),l_B(g))) = \{(k+1,numact) : numact \in ((\bigcup_{i<=k} F_{LA}(d_i) \cap \{(n,(\varepsilon,\alpha)) : n < k, \alpha \in Act_B\}) \setminus \{Denum(d_i) : i = 1,..,k\})\}$, for $a \in \{concede\ \varphi,\ nod\ \varphi,\ question\ \varphi\}$.
After opponent's locutions *concede*, *nod* or *question* the player can use one from possible answers for all previous opponent's moves, excluding these ones which he has already used.

(R5) $Pr((d_{1..k-1},(k,l,(retract\ \varphi,\varepsilon)),l_W(g),l_B(g))) = \{(k+1,numact) : numact \in ((\bigcup_{i<=k} F_{LA}(d_i) \cap \{(n,(\varepsilon,\alpha)) : n < k, \alpha \in Act_B\}) \setminus \{Denum(d_i) : i = 1,..,k\})\} \cup \{(k+1,x,(\varepsilon,why\ \beta)) : \exists_{x<k}\ d_x = (x,y,(\beta\ \text{since}\ \varphi,\varepsilon))\}$ for some $\varphi,\beta \in Form(PV)$.
After opponent's locution *retract* φ the player can use one from possible answers for all previous opponent's moves, excluding these ones which he has already used but also he can ask for the reason for β if φ was previously used to justify β.

These rules for player B are analogous.

The protocol is a crucial element of the model since it gives strict rules which determine the behaviour of players. In other words, it formally describes who, when and which action can perform. Rules (R1) and (R2) refer to the beginning and end of the dialogue, respectively. Rule (R3) states that after locutions *claim*, *scold*, *why*, and *since*, only actions determined by the legal answer function can be used. According to rules (R4) and (R5), actions *concede*, *nod* and *retract* end one of the threads of dialogue. Therefore, the next action can start a new thread or return to one of the unfinished. Actions *nod* and *scold* act similarly to actions *concede* and *claim*, but what distinguishes these actions is their emotional charge.

To show how locutions and their contents affect players' emotions and goals we define two functions. The first one determines the change of intensity of emotions: $EMOT_p : Act_w \times Emotion_p \to Emotion_p$ where $p \in Pl$ and $Emotion_p$ is a set of all possible 5-tuples for emotions, i.e., $Emotion_p = \{(n_1,\ldots,n_5) : n_i \in \{1,\ldots,10\} \wedge i \in \{1,\ldots,5\}\}$. The second one determines the change of goals: $GOAL_p : Act_w \times Goal_p \to Goal_p$ where $p \in Pl$ and $Goal_p$ is a set of possible goals represented by expressions from the topic language, i.e. $Goal_p \subset Form(PV)$.

Finally, we define *global* (partial) *evolution function* $t : G \times Num_2Act \to G$, which determines results of actions. This function is symmetrical for both players. Let $d(g) = d(g)_{1,\ldots,m}$, then:

- $t(g,(m+1,j,(claim\ \varphi,\varepsilon))) = g'$ iff $\varphi \notin C_W(g) \wedge C_W(g') = C_W(g) \cup \{\varphi\}$ $\wedge\, E_W(g') = EMOT_W(claim\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(claim\ \varphi, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(claim\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(concede\ \varphi,\varepsilon))) = g'$ iff $\varphi \in C_B(g) \wedge C_W(g') = C_W(g) \cup \{\varphi\} \wedge E_W(g')$ $= EMOT_W(concede\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(concede\ \varphi, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(concede\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(why\ \varphi,\varepsilon))) = g'$ iff $C_W(g') = C_W(g)$ $\wedge\, E_W(g') = EMOT_W(why\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(why\ \varphi, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(why\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(\varphi\ since\ \{\psi_1,\ldots,\psi_n\},\varepsilon))) = g'$ iff $C_W(g') = C_W(g) \cup \{\varphi,\psi_1,..,\psi_n\}$ $\wedge\, E_W(g') = EMOT_W(\varphi\ since\ \{\psi_1,\ldots,\psi_n\}, E_W(g)) \wedge GO_W(g') =$ $GOAL_W(\varphi\ since\ \{\psi_1,\ldots,\psi_n\}, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(\varphi\ since\ \{\psi_1,\ldots,\psi_n\},\varepsilon)))$,
- $t(g,(m+1,j,(retract\ \varphi,\varepsilon))) = g'$ iff $C_W(g') = C_W(g) \setminus \{\varphi\}$ $\wedge\, E_W(g') = EMOT_W(retract\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(retract\ \varphi, GO_W(g))$ $\wedge\, d(g') = (d(g)_{1,\ldots,m},(m+1,j,(retract\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(question\ \varphi,\varepsilon))) = g'$ iff $C_W(g') = C_W(g)$ $\wedge\, E_W(g') = EMOT_W(question\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(question\ \varphi, GO_W(g))$ $\wedge\, d(g') = (d(g)_{1,\ldots,m},(m+1,j,(question\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(scold\ \varphi,\varepsilon))) = g'$ iff $\varphi \notin C_W(g) \wedge C_W(g') = C_W(g) \cup \{\varphi\}$ $\wedge\, E_W(g') = EMOT_W(scold\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(scold\ \varphi, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(claim\ \varphi,\varepsilon)))$,
- $t(g,(m+1,j,(nod\ \varphi,\varepsilon))) = g'$ iff $\varphi \in C_B(g) \wedge C_W(g') = C_W(g) \cup \{\varphi\}$ $\wedge\, E_W(g') = EMOT_W(nod\ \varphi, E_W(g)) \wedge GO_W(g') = GOAL_W(nod\ \varphi, GO_W(g)) \wedge$ $d(g') = (d(g)_{1,\ldots,m},(m+1,j,(concede\ \varphi,\varepsilon)))$,

Global evolution function defines results of actions. In particular, actions *claim*, *concede*, *scold*, *nod* and *since* add an expression to the commitments set while action *retract* deletes it. Actions *why* and *question* do not modify this set.

3 Kripke model and model checking

The mathematical model for argumentative dialogue games provides a basis for applying the methods of model checking to verify the correctness of dialogue protocols relative to the properties that the protocols should satisfy. Model checking [2, 8, 9, 22] is an automatic verifying technique for concurrent systems such as digital systems, distributed systems, real-time systems, multi-agent systems, communication protocols, cryptographic protocols, concurrent programs, dialogue systems, and many others.

The prerequisite inputs to model checking are a *model* of the system under consideration and a formal characterisation of the *property* to be checked. Therefore, we associate with the given interpreted system a *Kripke structure*, that is the basis for the application of model checking. A Kripke structure is defined as a tuple $M = (G, Act, T, I)$ consisting of a set of global states G, a set of actions Act (in our approach Num_2Act), a set of initial states $I \subseteq G$, a transition relation $T \subseteq G \times Act \times G$ such that T is left-total. Relation T is defined as follows $(g, a, g') \in T$ *iff* $g' \in t(g, a)$. By T^* we will denote the reflexive and transitive closure of T.

To formulate properties of dialogue protocols suitable propositional temporal logics are applied. The most commonly used, in general, are linear temporal logic (LTL), computation temporal logic (CTL), a full branching time logic (CTL*), the universal and existential fragments of these logics, and other logics which are their modifications and extensions. One of the most important practical problems in the model checking is the exponential growth of the number of states of the Kripke structure. That is why in future work we intend to focus on symbolic model checking of dialogue protocols. Symbolic model checking avoids building a state graph; instead, sets and relations are represented by Boolean formulae. One of the possible methods of symbolic model checking is bounded model checking (BMC) [5, 6, 1, 3, 29]. It uses a reduction of the problem of truth of a temporal formula in a Kripke structure to the problem of satisfiability. In SAT-based BMC the aforementioned reduction is achieved by a translation of the transition relation and a translation of a given property to formulae of classical propositional calculus, whereas in SMT-based BMC to quantifier-free first order formulae.

The standard BMC algorithm, starting with $k = 0$, creates for a given Kripke structure M and a given formula φ, a formula $[M, \varphi]_k$. Then the formula $[M, \varphi]_k$ is forwarded to either a SAT-solver or a SMT-solver. Note, that in the case of SAT-base BMC the propositional formula is converted to a satisfiability equivalent propositional formula in conjunctive normal form before forwarding it to a SAT-solver. If the tested formula is unsatisfiable, then k is increased (usually by 1) and the process is repeated. The BMC algorithm terminates if either the formula $[M, \varphi]_k$ turns out to be satisfiable for some k, or k becomes greater than a certain, M-dependent, threshold (e.g. the number of states of M). Exceeding this threshold means that the formula φ is not true in the Kripke structure M. On the other hand, satisfiability of $[M, \varphi]_k$, for some k means that the formula φ is true in M.

4 Computation Tree Logic of Commitment and Action with Past

Interpreted systems are traditionally used to give a semantics to an epistemic language enriched with temporal connectives based on linear time [11]. Here we use CTL by Emerson and Clarke [10] as our basic temporal language and add commitment, emotion, goal, dynamic and past components to it. We call the resulting logic *Computation Tree Logic of Commitment and Action with Past*.

Definition 1 (Syntax). *Let $Pl = \{W, B\}$ be a set of players. The set of formulas is defined inductively as follows:*

- *true is a formula,*
- *if $\varphi \in Form(PV)$ and $p \in Pl$ then $COM_p(\varphi)$ and $G_p(\varphi)$ are formulas,*
- *$E_p(e)$ is a formula for $p \in Pl$ and $e \in \{$fear, disgust, joy, sadness, anger$\}$,*
- *if α and β are formulas, then so are $\neg\alpha$, $\alpha \wedge \beta$ and $\alpha \vee \beta$,*
- *if $\bar{a} \in Act_W$ and α is a formula, then so are $\mathrm{AX}_{(W,\bar{a})}\alpha$ and $\mathrm{AY}_{(W,\bar{a})}\alpha$,*
- *if $\bar{a} \in Act_B$ and α is a formula, then so are $\mathrm{AX}_{(\bar{a},B)}\alpha$ and $\mathrm{AY}_{(\bar{a},B)}\alpha$,*
- *if α and β are formulas, then so are $\mathrm{AX}\alpha$, $\mathrm{AG}\alpha$ and $\mathrm{A}(\alpha \mathrm{U} \beta)$,*
- *if α is a formula, then so are $\mathrm{AY}\alpha$ and $\mathrm{AH}\alpha$.*

The remaining basic modalities are defined by derivation: $\mathrm{EF}\alpha \stackrel{def}{=} \neg\mathrm{AG}\neg\alpha$, $\mathrm{EP}\alpha \stackrel{def}{=} \neg\mathrm{AH}\neg\alpha$, $\mathrm{EZ}\alpha \stackrel{def}{=} \neg\mathrm{AZ}\neg\alpha$, $\mathrm{EZ}_{(W,\bar{a})}\alpha \stackrel{def}{=} \neg\mathrm{AZ}_{(W,\bar{a})}\neg\alpha$, $\mathrm{EZ}_{(\bar{a},B)}\alpha \stackrel{def}{=} \neg\mathrm{AZ}_{(\bar{a},B)}\neg\alpha$, for $\mathrm{Z} \in \{\mathrm{X}, \mathrm{Y}\}$, Moreover, $\alpha \Rightarrow \beta \stackrel{def}{=} \neg\alpha \vee \beta$, $\alpha \Leftrightarrow \beta \stackrel{def}{=} (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$, and $false \stackrel{def}{=} \neg\, true$.

The formula *true* is used for technical reasons and helps to express that some action is possible to execute, i.e., an action can lead to a state in which *true* holds. Of course, *true* is satisfied in every state.

Formula $COM_p(\varphi)$ describes the actual set of commitments of player p, more precisely, it expresses that φ is in this set. We should emphasize that φ is not a formula of the language defined herein, but a part of a separate structure in which it is possible to express the spoken sentences. In dialogue system, all actions are aimed at influencing the players' commitments. Therefore, the modality *COM* is very important and often used in the protocol specification. Modalities E_p and G_p allow for expressing properties concerning emotions and goals of player p.

The temporal modalities X, G stand for "at the next step", and "forever in the future", respectively. Y, H are their past counterparts "at the previous step", and "forever in the past". The modality A is the universal quantifier - "for all". Thus, AX means "for all next states" while AG means "for all states on all paths".

We also introduce modality $\mathrm{AX}_{(W,\bar{a})}$. It encodes an additional fact calling the action that led to the next state. Since we are talking about the implementation of a specific action, we must also indicate its executor. Hence, the subscript $(W, \bar{a})$, expressing that the performer is a player White, is added. A similar modality is defined for Black: $\mathrm{AX}_{(\bar{a},B)}$. As a result, the formula $\mathrm{AX}_{(\bar{a},B)}\alpha$ intuitively expresses that "at all next states reached after execution of action $\bar{a}$ by Black, α is true".

The operator U stands for *Until*; the formula $\alpha \mathrm{U} \beta$, expresses the fact that β eventually occurs and that α holds continuously until then.

As customary, the negation $\neg$A can be replaced by the existential quantifier E using the de Morgan's laws. So, $\neg \mathrm{AX}\alpha$ is equivalent with $\mathrm{EX}\neg\alpha$ - there exists a next state at which α holds. The interpretation of the other existential formulas is similar.

First, in order to give the semantics for the above formulas, we need to give a formal definition of a computation. A *computation* in a Kripke structure $M = (G, Act, T, I)$ is a possibly infinite sequence of states $\pi = (g_0, g_1, \ldots)$ such that there exists an action a_m for which $(g_m, a_m, g_{m+1}) \in T$ for each $m \in \mathbb{N}$, i.e., g_{m+1} is the result of applying the transition relation T to the global state g_m, and the action a_m.

Below we abstract from the transition relation, the actions, and the protocols, and simply use T, but it should be clear that this is uniquely determined by the interpreted system under consideration. In interpreted systems terminology, a computation is a part of a run. A k-computation is a computation of length k. For a computation $\pi = (g_0, g_1, \ldots)$, let $\pi(k) = g_k$, and $\pi_k = (g_0, \ldots, g_k)$, for each $k \in \mathbb{N}$. By $\Pi(g)$ we denote the set of all the infinite computations starting at g in M, whereas by $\Pi_k(g)$ the set of all the k-computations starting at g.

Definition 2 (Semantics – Interpretation). *Let M be a model (Kripke structure), $g \in G$ be a state, π be a computation, and α, β be formulas. $M, g \models \alpha$ denotes that α is true at the state g in the model M. M is omitted, if it is implicitly understood. The relation $\models$ is defined inductively as follows:*

$g \models true$ *for all* $g \in G$,
$g \models COM_p(\varphi)$ *iff* $\varphi \in C_p(g)$,
$g \models E_p(e)$ *iff* $n_i > 5$ *in* $E_p(g) = (n_1, .., n_5)$, *where e is fear, disgust, joy, sadness, anger and* $i = 1, 2, 3, 4, 5$, *respectively,*
$g \models G_p(\varphi)$ *iff* $\varphi \in GO_p(g)$,
$g \models \neg\alpha$ *iff* $g \not\models \alpha$,
$g \models \alpha \wedge \beta$ *iff* $g \models \alpha$ *and* $g \models \beta$,
$g \models \mathrm{AX}_{(W,\bar{a})}\alpha$ *iff* $\forall a = (i, j, (\bar{a}, \varepsilon)) \in Num_2Act$ *and* $\forall g' \in G$ (*if* $(g, \bar{a}, g') \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AX}_{(\bar{a},B)}\alpha$ *iff* $\forall a = (i, j, (\varepsilon, \bar{a})) \in Num_2Act$ *and* $\forall g' \in G$ (*if* $(g, \bar{a}, g') \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AX}\alpha$ *iff* $\forall g' \in G\, \forall a \in Num_2Act$ (*if* $(g, a, g') \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AG}\alpha$ *iff* $\forall \pi \in \Pi(g)$ $(\forall_{m \geq 0}\, \pi(m) \models \alpha)$,
$g \models \mathrm{A}(\alpha \mathrm{U} \beta)$ *iff* $\forall \pi \in \Pi(g)$ $(\exists_{m \geq 0}\, [\pi(m) \models \beta$ *and* $\forall_{j<m}\, \pi(j) \models \alpha])$,
$g \models \mathrm{AY}_{(W,\bar{a})}\alpha$ *iff* $\forall a = (i, j, (\bar{a}, \varepsilon)) \in Num_2Act$ *and* $\forall g' \in G$ (*if* $(g', a, g) \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AY}_{(\bar{a},B)}\alpha$ *iff* $\forall a = (i, j, (\varepsilon, \bar{a})) \in Num_2Act$ *and* $\forall g' \in G$ (*if* $(g', a, g) \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AY}\alpha$ *iff* $\forall g' \in G\, \forall a \in Num_2Act$ (*if* $(g', a, g) \in T$, *then* $g' \models \alpha$),
$g \models \mathrm{AH}\alpha$ *iff* $\forall g' \in G$ (*if* $(g', g) \in T^*$, $g' \models \alpha$).

The description of the semantics is finished by giving the definition of the validity in the model.

Definition 3. (Validity) *A formula φ is valid in M (denoted $M \models \varphi$) iff $M, \imath \models \varphi$, i.e., φ is true at the initial state of the model M.*

5 Properties of dialogue protocols

The formal language introduced in the previous section is used for giving the specification for dialogue protocols as well as for describing properties of these protocols. The properties can be divided into several classes [21]. Some of them are studied below.

Safety. Safety property usually expresses that something bad does not happen. However, it can also express that something good is always true. The best illustration here is the specification of locutions used in dialogues:

$$\mathrm{AG}(\mathrm{AX}_{(W,claim\ \alpha)}\ COM_W\,\alpha).$$

This formula states that after locution *claim* α, the formula α is in the set of commitments of the performer.

The next formula expresses a similar property, i.e., before the execution of the locution *retract* α, the formula α must be in the commitments set of the player:

$$\mathrm{AG}(\mathrm{AY}_{(W,retract\ \alpha)}\ COM_W\,\alpha).$$

Nontermination. One of the most important safety properties is nontermination. It expresses that every legal dialogue, i.e., dialogue in accordance with rules of a dialogue game does not have a termination state:

$$\mathrm{AG}(\mathrm{EX}true).$$

This formula states that in every state of every computation there is an action which can be performed and after execution of this action a formula *true* is satisfied. As a consequence, every dialogue is infinite.

Guarantee. One of the guarantee properties, i.e., properties that ensure that some event eventually happens, is *termination*. In dialogue systems, we often assume that the end of a dialogue means the fulfillment of a certain condition. This condition may express that one of the players, e.g. W, is happy:

$$\mathrm{E}(true\ \mathrm{U}\ E_W(joy)).$$

If any dialogue should end with the *termination* condition and this condition means that White does not feel fear, then we can express this fact as follows:

$$\mathrm{A}(true\ \mathrm{U}\ \neg E_W(fear)).$$

The formula claims that every computation contains a state at which the required condition holds.

Response. The response property expresses the fact that a property β is a guaranteed response to a condition α. An example of this is the formula

$$\mathrm{AG}(COM_p(\alpha) \Rightarrow \mathrm{E}(true\ \mathrm{U}\ \neg\ COM_p(\alpha)))$$

which states that if a player is committed to α, then during the dialogue he can change it. This property is very important since it states that it is possible to reject some commitment and at the same time it means the ability to change some opinion, what is crucial for argumentative dialogues. It makes no sense to provide and analyze arguments if the change of players' commitments is not possible at all.

6 Conclusion

The aim of our research is to design and implement a framework to provide a communication between a user and a machine which allows to better understand emotions that appear during human dialogues. We plan to create a tool that will support the personal development in this matter, i.e., the acquisition of skills of identifying and naming emotions. This is particularly important for training teachers, educators, psychologists, and parents. This process can take place between a human, which plays a role of a student, and a software agent, which plays a role of a teacher. The challenge is to design a suitable interface for such communication. However, the implementation should be preceded by constructing a mathematical model and proposing a new dialogue protocol. In our work, we also propose formal language for protocol specification and expressing its properties. On this basis, we plan to design and implement a multimedia tool for educational purposes. Psychological aspects of the project are consulted with a group of psychologists. Our research does not deal with linguistic analysis, but we want to explore dialogues with the fixed base so that the user can learn to recognize these places and elements of dialogue which relate to emotions.

Acknowledgment. The research by Kacprzak have been carried out within the framework of the work S/W/1/2014 and funded by Ministry of Science and Higher Education.

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *STTT*, 11(1):69–83, 2009.
2. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
3. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pp. 825–885, 2009.
4. J. Bentahar, J.-J. C. Meyer, and W. Wan. *Specification and Verification of Multi-agent Systems*, chapter Model Checking Agent Communication, pages 67–102. Springer US, 2010.
5. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of DAC'99*, pages 317–320, 1999.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.
7. K. Budzynska, M. Kacprzak, A. Sawicka, and O. Yaskorska. *Dialogue Dynamics: Formal Approach*. IFS PAS, 2015.
8. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In J. R. Wright, L. Landweber, A. J. Demers, and T. Teitelbaum, editors, *Conf. Rec. of the Tenth Annual ACM Symposium on Principles of Programming Languages,* pages 117–126, ACM Press, 1983.
9. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.
10. E. A. Emerson and E. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
11. R. Fagin, J.Halpern, Y. Moses, and M.Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
12. C. Hamblin. *Fallacies*. Methuen, London, 1970.

13. A. V. Jones and A. Lomuscio. Distributed BDD-based BMC for the verification of multi-agent systems. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *Proc. of AAMAS, Volume 1-3*, pages 675–682, 2010.
14. M. Kacprzak, M. Dziubinski, and K. Budzynska. Strategies in dialogues: A game-theoretic approach. In S. Parsons, N. Oren, C. Reed, and F. Cerutti, editors, *Proc. of COMMA*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 333–344. IOS Press, 2014.
15. M. Kacprzak and A. Sawicka. Identification of formal fallacies in a natural dialogue. *Fundam. Inform.*, 135(4):403–417, 2014.
16. M. Kacprzak, A. Sawicka, and A. Zbrzezny. Dialogue systems: Modeling and prediction of their dynamics. In A. Abraham, K. Wegrzyn-Wolska, E. A. Hassanien, V. Snasel, and M. A. Alimi, editors, *Proc. of AECIA*, pages 421–431. Springer International Publishing, 2016.
17. M. Kacprzak and O. Yaskorska. Dialogue protocols for formal fallacies. *Argumentation*, 28(3):349–369, 2014.
18. L. Keiff. Dialogical logic. In E. N. Zalta, editor, *The Stanford Enc. of Philosophy*. 2011.
19. A. Lomuscio, W. Penczek, and H. Qu. Partial order reductions for model checking temporal epistemic logics over interleaved multi-agent systems. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *IFAAMAS, Volume 1-3*, pages 659–666, 2010.
20. K. Lorenz and P. Lorenzen. Dialogische logik. *WBG. Darmstadt*, 1978.
21. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.
22. A. Meski, W. Penczek, M. Szreter, B. Wozna-Szczesniak, and A. Zbrzezny. BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: algorithms and their performance. *AAMAS*, 28(4):558–604, 2014.
23. F. S. Nawwab, P. E. Dunne, and T. Bench-Capon. Exploring the role of emotions in rational decision making. In *Proc. of COMMA*, 2010.
24. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundam. Inform.*, 55(2):167–185, 2003.
25. H. Prakken. Models of persuasion dialogue. In *Argumentation in AI*, pages 281–300. Springer, 2009.
26. S. Rahman and T. Tulenheimo. From games to dialogues and back: towards a general frame for validity. In O. Majer, A. Pietarinen, and T. Tulenheimo, editors, *Games: Unifying Logic, Language, and Philosophy*, volume 15 of *LEUS*, pages 153–208. Dordrecht: Springer, 2006.
27. F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic*, 5(2):235 – 251, 2007. Logic-Based Agent Verification.
28. R. Silveira, G. K. da Silva Bitencourt, T. . Gelaim, J. Marchi, and F. de la Prieta. Towards a model of open and reliable cognitive multiagent systems: Dealing with trust and emotions. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 4(3), 2016.
29. C. Tinelli. SMT-based model checking. In *NASA Formal Methods - 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings*, page 1, 2012.
30. W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bosnacki and S. Leue, editors, *Proc. of SPIN*, volume 2318 of *LNCS*, pages 95–111, 2002.
31. J. Visser, F. Bex, C. Reed, and B. Garssen. Correspondence between the pragma-dialectical discussion model and the argument interchange format. *Studies in Logic, Grammar and Rhetoric*, 23(36):189–224, 2011.
32. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of N.Y. Press, 1995.
33. S. Wells and C. A. Reed. A domain specific language for describing diverse systems of dialogue. *J. Applied Logic*, 10(4):309–329, 2012.
34. O. Yaskorska, K. Budzynska, and M. Kacprzak. Proving propositional tautologies in a natural dialogue. *Fundam. Inform.*, 128(1-2):239–253, 2013.

Shapes of concurrency

Piotr Chrząstowski-Wachtel
Institute of Informatics
Warsaw University
pch@mimuw.edu.pl

The question, how true concurrency differs from interleaving, has been studied intensively, beginning with the works of Mazurkiewicz [Maz], Milner [Mil], Pratt [Pra] and many others. The Mazurkiewicz trace theory requests for specifying the dependency relation, which makes such difference explicit. If in some state the sequences *ab* and *ba* are executable, then if *a* and *b* are independent, then both the sequences describe the same behavior. Moreover, we can say about a concurrent step allowing the execution of *a* and *b* in parallel. However, if *a* and *b* are dependent, then such two sequences describe two different sequential processes.

Pratt in [Pra] makes an interesting geometrical interpretation of concurrency, letting it be considered as the creation of a new dimension. If we imagine the progress of executing *a* and *b* as segments: *a* on OX axis and *b* on OY axis, both originating in O, then executing *ab* in a sequential way means that we first go along the segment *a*, next come to the corner of a rectangle, from which we traverse the segment *b*. On the other hand, when *a* and b are concurrent, we can imagine the space of possible states as the whole interior of the rectangle spanned on *a* and *b* as sides.

In many situations, when we consider the reachability graph of a concurrent system, whenever two actions can be executed in parallel, we can see a diamond-like shape.

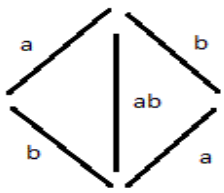


Fig 1. A diamond-like concurrency

The diagonal *ab* reflects the parallelism. Actions *a* and *b* can occur in any order (left-hand and right-hand paths) or they can occur in parallel.

The situation gets more complex if we allow multiple actions to be performed in parallel. The diamond-like pattern can no longer be sufficient. The question of interleaving becomes inappropriate. Let's illustrate this phenomenon on one physical example. From now on we assume that we have ideally resilient identical balls of negligible radius. We will consider collisions without energy loss, satisfying the laws of physics, so exchanging the momentum and energy (perfectly elastic collisions).

Fig 2. Three colliding balls

Let's start with a simple example of three balls on a straight line. The left-hand ball is moving with constant velocity v to the right, the right-hand ball is moving with velocity v to the left, while the middle ball does not move. Let's encode the state of the system by three letters: R for right, S for stand and L for left. So the initial state of the system is RSL.

There are three actions possible in the initial state:

1. The first ball hits the middle one, stopping immediately and transmitting all the momentum to the second ball leading to the state SRL.
2. The third ball hits the middle one, leading to the state RLS
3. The first and the third ball hit the middle one, bouncing backwards, without moving it, so leading to the final state LSR.

Two first of them lead to consecutive events that must happen next. The complete picture of the states is given on Fig 3. The labels on edges name the balls, which take part in the collision. Mind that they reflect different actions (for instance 12 means just that the ball 1 collides with ball 2, without specifying what kind of collision it is).

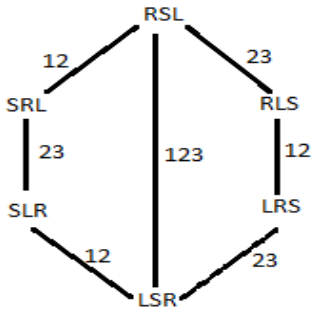


Fig 3. State graph of the tree colliding balls

The diagonal labeled 123 reflects the event in which the first and the last ball hit the middle one in the same moment. In fact it can be interpreted as 12 || 23, but this time no interleaving is possible. The event 123 is not a combination of 12 and 23.

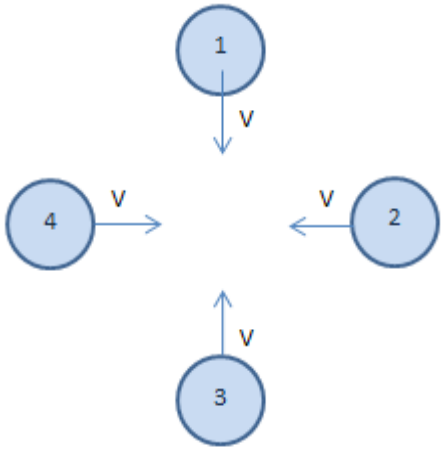


Fig 4. Four colliding balls

Such hexagonal shape can be typical for 3 objects, which interact in more complex situations, like one billiard ball hitting the two other ones, which stick together; a situation of great instability.

We can presume that the multi-event parallelism can create much more complex patterns. Let's consider one more example. Imagine 4 balls, like on Fig 4, targeting to one common centre: ball 1 going South, ball 2 West, ball 3 North and ball 4 East. We do not assume that the balls are equidistant from the center. It can happen that, for instance, ball 1 hits ball 2 before any other event happens. According to the laws of physics, in this case ball 1 and ball 2 will exchange the momentum, hence ball 1 will start going West, and ball 2 will go South. Observe, that if any collision happens, it will take place in the common center and the resulting directions will be like the original ones, so S-W-N-E. Eventually we expect that the four balls, after possible bounces, will pass the center and leave the initial area going to infinity.

Let us encode the states by four letters assigned to each of the balls, so the initial state will be SWNE.

On Fig. 5 we can see a part of the big state graph, again hiding some information, like where the ball actually is with respect to the center. Some of the possible runs are not shown, like the one, in which none or only one collision takes place.

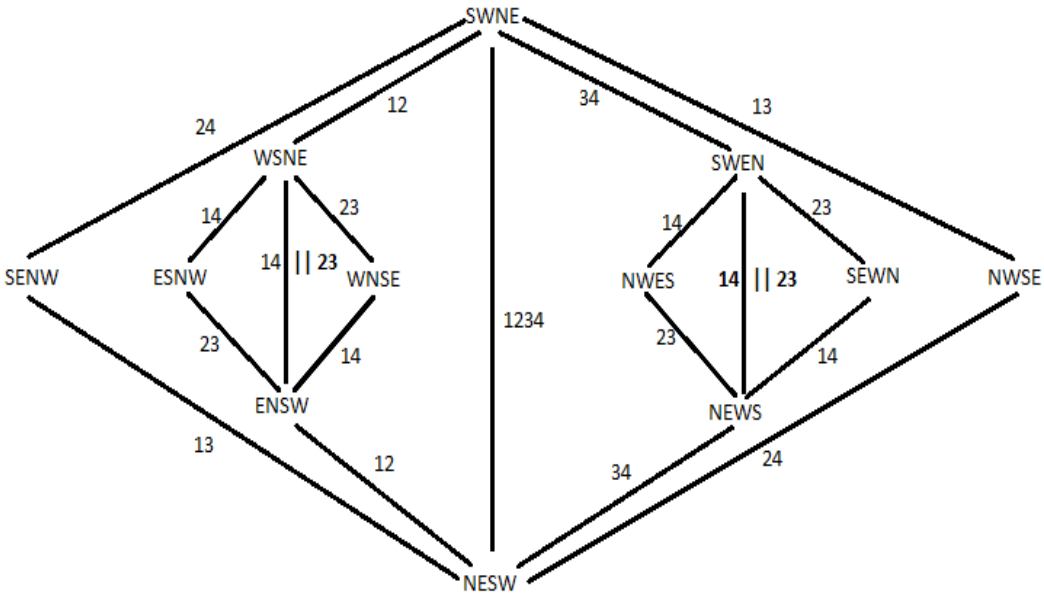


Fig 5. Part of the state graph of four possibly colliding balls

In the last case we can see an octagonal shape (among many) SWNE-WSNE-WNSE-ENSW-NESW-NEWS-NWES-SWEN, which makes sequentially the same result as the main diagonal SWNE-NESW. The diagonal run reflects the situation, in which all the balls are equidistant and reach the center at the same time. Each of them will bounce symmetrically the two neighboring ones and go immediately, without intermediate steps into the final direction. Mind that inside the diagram we can see two typical diamond-like occurrences of concurrent events.

So again, here we cannot replace the concurrent behavior by the composition of the sequential ones. Concurrency, like in the previous example, creates new run, which has little to do with the partly sequential ones, leading to the same result.

On Fig 5 we also see an interesting case of a diamond non-concurrency: the outer edges, however they form a diamond shape, cannot be replaced by a diagonal 13 || 24. Bouncing ball 1 against ball 3 can never happen in the same moment as bouncing ball 2 against ball 4.

With the increase of the number of parallel events, one can expect other shapes of concurrency, not necessarily being planar. Usually they reflect unstable physical systems -- a small change in the initial state can cause a dramatic change in the behavior. Such instability is well recognized by the physicists. Let us look deeper into some other part of the state graph of the behavior of the system from Fig. 4. For the

moment let us concentrate on three balls 1,4,2, assuming that the ball 3 is far away from the common center. So the initial state of these balls is SWNE, but the ball 3 will not change its status for a while.

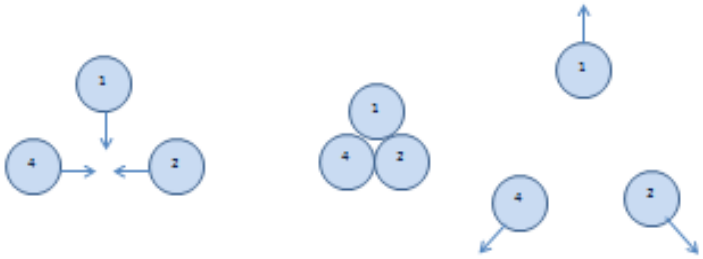


Fig.6 Unstable collision of 3 balls

If all the three considered balls collide in the center simultaneously, then the ball 1 will go North, but the two balls 4 and 2 will exchange their momentums and absorb the momentum of ball 1. As a result they will bounce in two directions a bit South, as depicted in Fig.6. Let us call this state N(SE)N(SW) No other collision would happen, so this would be a final state of the system.

This will differ a lot from the case, in which the ball 2 would find itself a little bit towards East in the initial position. In such a case two collisions between the balls 1,4,2 would happen. First the ball 1 would collide with the ball 4 exchanging their momentums and transforming the system state to EWNS, next the ball 1 and 2 would collide reaching the state WENS, quite different from the final state of Fig 6. So a small change in the initial position would result in a major qualitative difference of the final state. This is an illustration of an instability phenomenon, in which case the multi-collision of balls cannot be factorized as a result of interleaving the bi-collisions.

Anyway, the world of concurrency seems to hide some mysteries. Here we demonstated a set of examples showing why interleaving models can be not apropppriate for modeling true concurency.

Bibliography

[Maz] A.Mazurkiewicz, Introduction to Trace Theory, in The Book of Traces; V.Diekert, G.Rozenberg (Eds.) World Scientific, 1995

[Mil] R. Milner, Communication and concurrency, Prentice Hall International Series in Computer Science, 1989

[Pra] V.R.Pratt, Modeling Concurrency with Geometry, in 18th Annual ACM Symposium onx Principles of Programming Languages, Orlando, Florida, USA, 1991

A Protocol of Mutual Exclusion for DSM Based on Vectors of Global Timestamps

Ludwik Czaja
[1]Vistula University, Warsaw
[2] Institute of Informatics, The University of Warsaw
lczaja@mimuw.edu.pl

Abstract

A new protocol using vectors of global timestamps for mutual exclusion in systems with Distributed Shared Memory (DSM) is described and some of its properties proved.

1. Introduction

Exclusive access to resources in concurrent programming has found various solutions originating in aforetime works by Dekker (unpublished but presented in [Dij 1968]), Dijkstra [Dij 1968], [Dij 2002], Lamport [La 1978], [La 1979], Ricart &Agrawala [R-A 1981], Saxena&Rai [S-R 2003] and a number of others. With coming of real multicomputer distributed systems without central memory and clock, where cooperation or competition of computers takes place only by message passing, the problem became essentially more complicated than in case of time-sharing systems or multiprocessors with shared physical memory. This concerns especially systems with no aid of central server: the computers „negotiate" by exchanging messages through the network and only one at a time is entitled to access a resource. A protocol based on vectors of global timestamps is presented in this paper and some of its properties are proved. The protocol is intended for systems with distributed shared memory (DSM), where the local memory in every computer is uniformly accessible for all computers: DSM is treated as a union of local memories. It is known that applications using DSM with some models of memory consistency [Cz 2016], require mutual exclusion. In the described solution, every computer keeps a vector of global timestamps of current requests for critical section, which are being issued by the connected computers. We do not discuss in details problems of timestamp advancement and mechanism of vector clocks (cf, for instance [S-R 2003], [K-M-C-H 2016]). It is assumed that such mechanisms provide current values of timestamps for the protocol described here. Likewise an issue of deadlock and fairness has been omitted, because of permitted space limitation of this paper.

A schematic structure of multicomputer system with Distributed Shared Memory is in Fig.1.1.

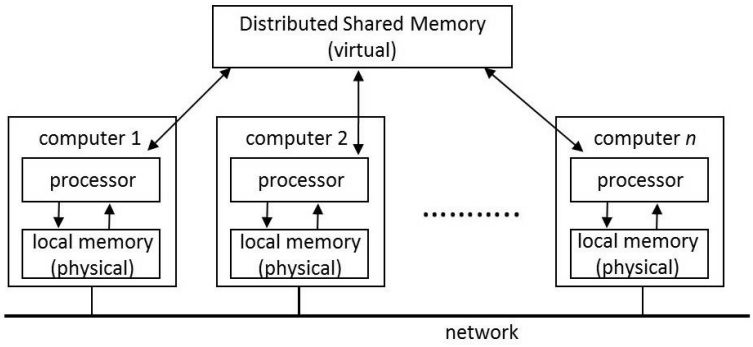


Fig.1.1.

2. Global timestamps revisited

A set Z is partially ordered iff its elements are related by relation $\sqsubseteq \subseteq Z \times Z$ satisfying: for every $x \in Z,\ y \in Z,\ z \in Z$:

1. $x \sqsubseteq x$ (reflexivity)
2. if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$ (antisymmetry)
3. if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$ (transitivity)

Moreover, if apart from (1), (2), (3):

4. $x \sqsubseteq y$ or $y \sqsubseteq x$ (connectivity)

then Z is linearly (totally) ordered. As usually, $x \sqsubset y$ iff $x \sqsubseteq y$ and $x \neq y$

Let $E(S)$ denote a set of events that may occur during activity of a distributed system S and let us define a partial order relation combining two kinds of precedence of events: occurring in the same process and of message sending and reception. For $x, y \in E(S)$ two auxiliary binary relations $\xrightarrow{process}$ and $\xrightarrow{message}$ are admitted as primary notions with the meaning:

- if x precedes y in the same process or if $x = y$ then $x \xrightarrow{process} y$
- if x is a sending message by a certain process and y is a reception of this message by another process then $x \xrightarrow{message} y$

A (weak) precedence $\xrightarrow{precedes} \subseteq E(S) \times E(S)$ is the least relation satisfying:

- if $x \xrightarrow{process} y$ or $x \xrightarrow{message} y$ then $x \xrightarrow{precedes} y$
- if $x \xrightarrow{precedes} y$ and $y \xrightarrow{precedes} z$ then $x \xrightarrow{precedes} z$

Events x, y are independent (concurrent) iff neither $x \xrightarrow{precedes} y$ nor $y \xrightarrow{precedes} x$, written $x||y$

Relation $\xrightarrow{precedes}$ is a modified precedence introduced by Lamport [La 1978] but due to the reflexivity of $\xrightarrow{process}$, the relation $\xrightarrow{precedes}$ is a partial order – contrarily to the Lamport's version. A common (global) clock and common memory are absent in asynchronous distributed systems, and partially ordered events are watched from outside of the system as occuring in real (external) time, thus, their precedence relation should imply similar order between real time instants of their occurrences. So, an injection mapping $C : E(S) \rightarrow R$ (R - set of real numbers), called a *logical clock* should be defined, satisfying implication $x \xrightarrow{precedes} y \Rightarrow C(x) \leq C(y)$, where values $C(x), C(y)$ are logical (not real) time instants of events x, y. To avoid absurd relationship of events to their time instants visible from outside of the system (when message reception precedes its dispatch), a compensation of processors' local clocks is necessary. So, if a sender sends a message together with its local time of dispatch and a receiver gets it earlier with respect to its local time, then the receiver must put forward its clock (a time register) to a time a little later than the time received from the sender. This procedure ensures the implication $x \xrightarrow{precedes} y \Rightarrow C(x) \leq C(y)$ if the values $C(x), C(y)$ are assumed to be compensated time instants of events x, y, called their *timestamps*. Obviously the reverse implication does not hold for some x, y if $x||y$ (see Fig.2.1). The logical clock C measures the compensated time. But it may happen that $C(x) = C(y)$ and $x \neq y$ for some concurrent x, y, so the mapping C is not one-to-one function, thus C does not establish unique representation of events by their timestamps. However if the processes are linearly ordered, e.g. numbered and the notion of event's timestamp is supplemented with a number of the process in which the event appears, then events can be uniquely represented by the richer timestamps, called *global*. So, let $nr(p_x)$ be a unique number of process p_x in which the event x occurs (a given event may occur in exactly one process, thus it identifies this process). A pair $\langle C(x), nr(p_x) \rangle$ is called a *global timestamp* of event x and let $\preccurlyeq$ denote a relation between global timestamps defined as $\langle C(x), nr(p_x) \rangle \preccurlyeq \langle C(y), nr(p_y) \rangle$ iff $C(x) < C(y)$ or if $C(x) = C(y)$ then $nr(p_x) \leq nr(p_y)$. Obviously $\preccurlyeq$ is linear, the so-called lexicographic order and the one-to-one injection mapping $\Gamma : E(S) \rightarrow R \times N$ (N - set of natural numbers) has been established by $\Gamma(x) = \langle C(x), nr(p_x) \rangle$, because $\Gamma(x) = \Gamma(y) \Rightarrow x = y$ for all x, y. Therefore, Γ establishes a unique representation of events by their global timestamps. Again, the implication $x \xrightarrow{precedes} y \Rightarrow \Gamma(x) \preccurlyeq \Gamma(y)$ holds but not the reverse one (see Fig.2.1).

Fig.2.1 exemplifies some relationships between events and their global timestamps during a system activity fragment. Black and grey circles are events of send and receive message respectively. Processes are numbered as follows: $nr(p1) < nr(p2) < nr(p3)$.

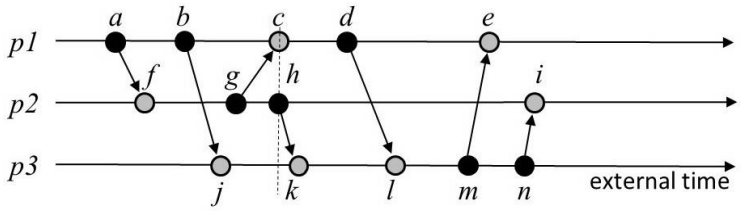


Fig.2.1. $a \xrightarrow{precedes} k \Rightarrow \langle C(a), nr(p_a)\rangle \prec \langle C(k), nr(p_k)\rangle$, $\langle C(c), nr(p_c)\rangle \prec C(h), nr(ph)\rangle$ but $c||h$ where $p_a = p_b = p_c = p_d = p_e = p1$. $p_f = p_g = p_h = p_i = p2$. $p_j = p_k = p_l = p_m = p_n = p3$.

3. Distributed mutual exclusion, a protocol and its properties

The global timestamps are used in a number of implementations of mechanisms in distributed systems. Consider a new protocol implementing distributed mutual exclusion with the following assumptions:

1. computers work in paralel asynchronously and are numbered $1, 2, ..., n$;
2. writing and reading to/from DSM memory is governed by the memory manager of each computer; computers communicate by message passing only and message propagation delay is finite but unpredictable.
3. there is one critical section (if there were more of them, the main concept of the protocol would be retained);
4. each request to the protocol for the critical section makes deliver of a current global timestamp of this event to the requesting computer;
5. computer of number i keeps vector $\overrightarrow{\mathbf{r}_i} = [r_{i1}, r_{i2}, ..., r_{in}]$ of variables r_{ik} $i, k = 1, 2, ..., n$ allocated in its physical memory; it stores the current timestamp of request in the component r_{ii} , then fetches values of components r_{kk} $(k \neq i)$ from remaining computers and stores them in variables r_{ik} of its vector $\overrightarrow{\mathbf{r}_i}$. Fig.3.1 depicts location of vectors of timestamps in the local memories;
6. initially all variables r_{ij} contain ∞ with $\infty > x$ for any number x;
7. by $min(\overrightarrow{\mathbf{r}_i})$ is denoted the least value of the components in $\overrightarrow{\mathbf{r}_i}$;

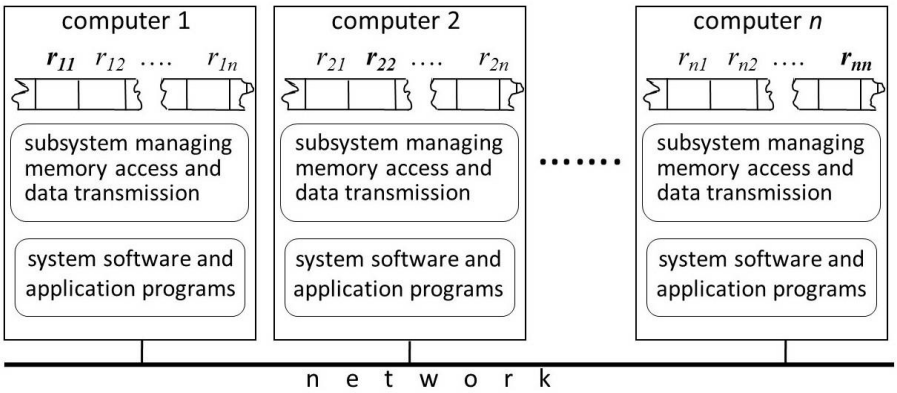


Fig.3.1. Structure of distributed system of n computers with vectors $\overrightarrow{\mathbf{r}_i} = [r_{i1}, r_{i2}, ..., r_{in}]$ $(i = 1, 2, ..., n)$ of timestamps allocated in local memories

A computer of number i when using the protocol depicted as a transition graph in Fig.3.2 for exclusive access to a protected resource, passes throughout the following states (subscript i in the names of states is omitted in order not to overload notation):

- W – execution of local (not critical) section
- B – import of current timestamps stored in variables r_{kk} of remaining computers; execution of $n-1$ assignments $r_{ik} := r_{kk}$ $(k \neq i)$; test of condition $r_{ii} > min(\overrightarrow{\mathbf{r}_i})$. State B is *stable* when the computer has completed fetch of all values of r_{kk} (note the various transmission duration of these values - see Theorem 3.2). In what follows, the adjective "stable" will be ommited when the noun "state" is used.
- Y – refusal to perform critical section (waiting state)
- R – execution of critical section
- G – release of critical section. This state is stable when the computer has completed broadcast of ∞ to all remaining computers.

Their set: $\mathbf{\Omega} = \{W, B, Y, R, G\}$

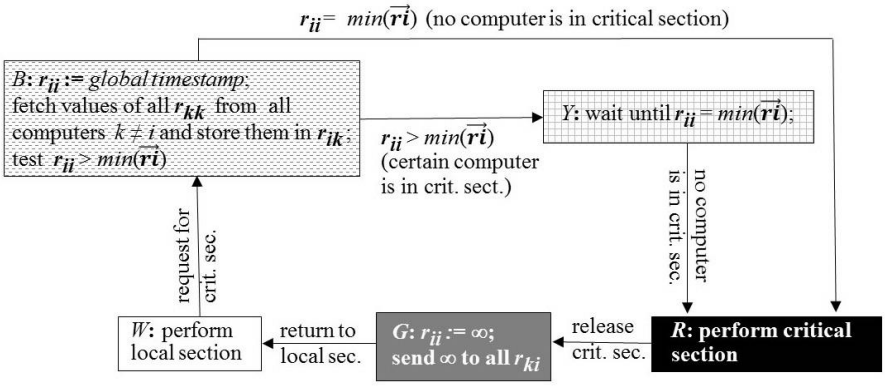


Fig.3.2. The distributed mutual exclusion protocol performed by computer of number i in the cycle from request for critical section till release.

Let us admit the following denotations:

- $Q_i \to Q'_i$ - computer of number i passes from a state $Q_i \in \mathbf{\Omega}$ to the next state $Q'_i \in \mathbf{\Omega}$ in the transition graph depicted in Fig.3.2. Note that transitions $B \to R$ and $Y \to R$ are possible when $r_{ii} = min(\vec{\mathbf{r}_i})$, thus due to steady growth of global timestamp as a strictly increasing function of time, at most one computer may perform critical section at a time. A formal proof is given further.

- Set of global states $\mathbf{\Omega}^n = \underbrace{\mathbf{\Omega} \times \mathbf{\Omega} \times ... \times \mathbf{\Omega}}_{n \text{ times}}$ (the ith component correspods to computer number i) satisfying: if $\vec{Q} = [Q_1, Q_2, \ldots, Q_n] \in \mathbf{\Omega}^n$ then $\neg\exists i, j : (i \neq j \wedge Q_i = R \wedge Q_j = R)$.

- Initial state: $\overrightarrow{Q_{init}} = [W, W, \ldots, W] \in \mathbf{\Omega}^n$ with $r_{ij} = \infty$ for every computer $i = 1, 2, \ldots, n$.

- For $\vec{Q} = [Q_1, Q_2, \ldots, Q_n] \in \mathbf{\Omega}^n$ and $\vec{Q'} = [Q'_1, Q'_2, \ldots, Q'_n] \in \mathbf{\Omega}^n$ let $\vec{Q} \rightrightarrows \vec{Q'}$ mean: there exists a computer of number i such that $Q_i \to Q'_i$ and if $\neg Q_j \to Q'_j$ then $Q_j = Q'_j$. $\vec{Q'}$ is the next global state following $\vec{Q}$. As usually, by $\vec{Q} \overset{*}{\rightrightarrows} \vec{Q'}$ is denoted reachability of $\vec{Q'}$ from $\vec{Q}$ i.e existence of global states $\vec{Q^0}, \vec{Q^1}, ..., \overrightarrow{Q^{m-1}}, \vec{Q^m}$ with $\vec{Q^0} = \vec{Q}$, $\vec{Q^m} = \vec{Q'}$, $\vec{Q^j} \rightrightarrows \overrightarrow{Q^{j+1}}$, $j = 0, 1, ..., m-1$.

It follows from the transition graph in Fig.3.2 that for any computer of number $i = 1, 2, ...n$:

1. Storing a timestamp in register r_{ii} proceeds only in the state B of computer of number i; r_{ii} retains this value until the transition $R \to G$ takes place.

2. Storing ∞ in register r_{ii} and sending to r_{ki} of remaining computers proceeds only in the state G. Thus, from point 1 follows that r_{ii} decreases its value only in the state B.

3. Global states are exactly those reachable from the initial state $\overrightarrow{Q_{init}} = [W, W, \dots, W]$.

4. Because computation of $min(\overrightarrow{\mathbf{r}_i})$ in the state B of computer of number i takes place on completion of fetching values of r_{kk} from remaining computers, the order of entering computers into the critical section does not depend of the transmission latency. This is the FCFS order (First Come First Served) due to the steady growth of the global timestamps. Formal proofs of mutual exclusion realized by the protocol in Fig.3.2 as well as independence of the FCFS strategy of the order of message transmissions and their latency when executing actions in the state B are given in Theorems 3.1 and 3.2.

Table 3.1 presents an exemplary piece of run of a four computers system with Distributed Shared Memory. This is the following succession of global states:
$[B, W, B, W] \rightrightarrows [Y, W, R, W] \rightrightarrows [Y, W, R, B] \rightrightarrows [Y, B, R, Y] \rightrightarrows [Y, Y, G, Y] \rightrightarrows [R, Y, W, Y] \rightrightarrows [G, Y, W, Y] \rightrightarrows [W, Y, W, R]$
Global timestamps, i.e. pairs of numbers, are coded by single numbers – for simplicity of notation.

Before demonstration of correctness of the protocol and its FCFS strategy of giving entrance to critical section for computers, let us make some remarks.

- **Consumption of time**. In the state B, computer i, on request for critical section, broadcasts message „send me value of your r_{kk}" to all $n-1$ remaining computers, and waits for delivery. In the worst case the message reaches all destinations one after one and responses arrive one after one. This takes $2(n-1)$ transmissions. In the state G, on release of critical section, the computer i broadcasts ∞ to all r_{ki} of all $n-1$ remaining computers. This takes $n-1$ transmissions in the worst case.

- **Failure**. If a computer k sends incorrect timestamp stored in r_{kk} to requesting computers in the state B, then their behaviour depends on this value. This may cause indefinite wait of requesting computer i in the state Y (if r_{kk} is small enough to make $min(\overrightarrow{\mathbf{r}_i})$ permanently less than r_{ii}) or violation of mutual exclusion (if computer k delivers to computer i value of r_{kk} such that $r_{ii} = min(\overrightarrow{\mathbf{r}_i})$, and after a while it delivers to computer j value of r_{kk} such that $r_{jj} = min(\overrightarrow{\mathbf{r}_j})$; computer j may then enter into the critical section before computer i leaves it). Problems of failure as well as decidability of deadlock and fairness (cf. [Cz 1980]) are left to a separate paper.

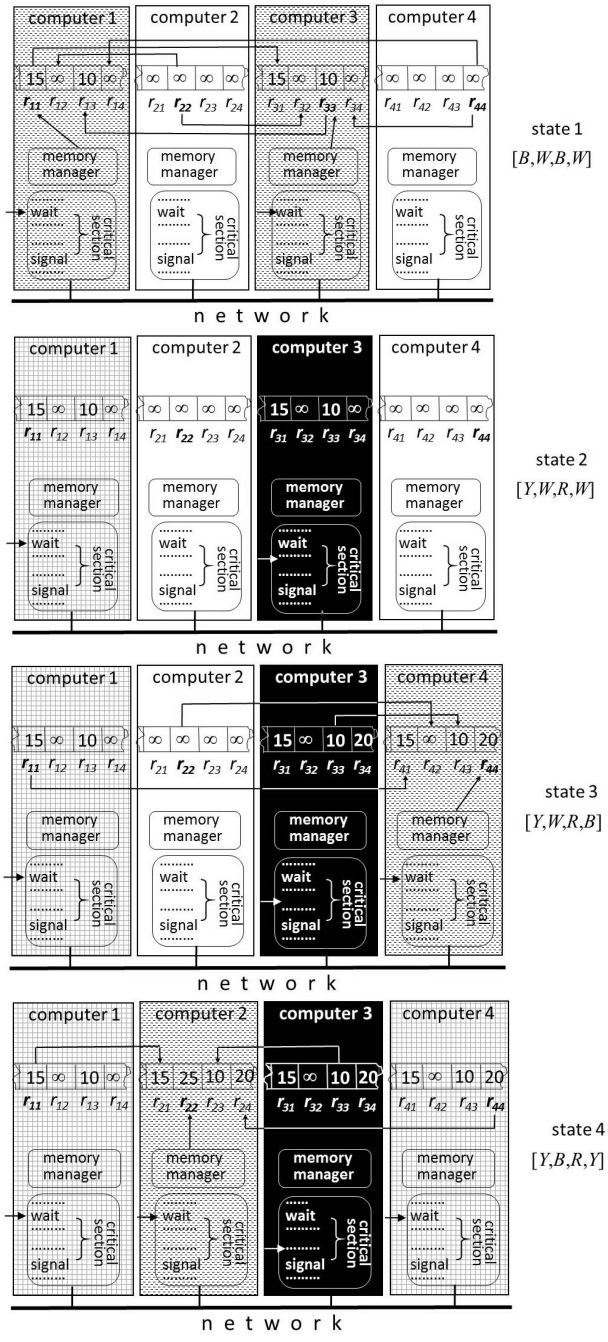
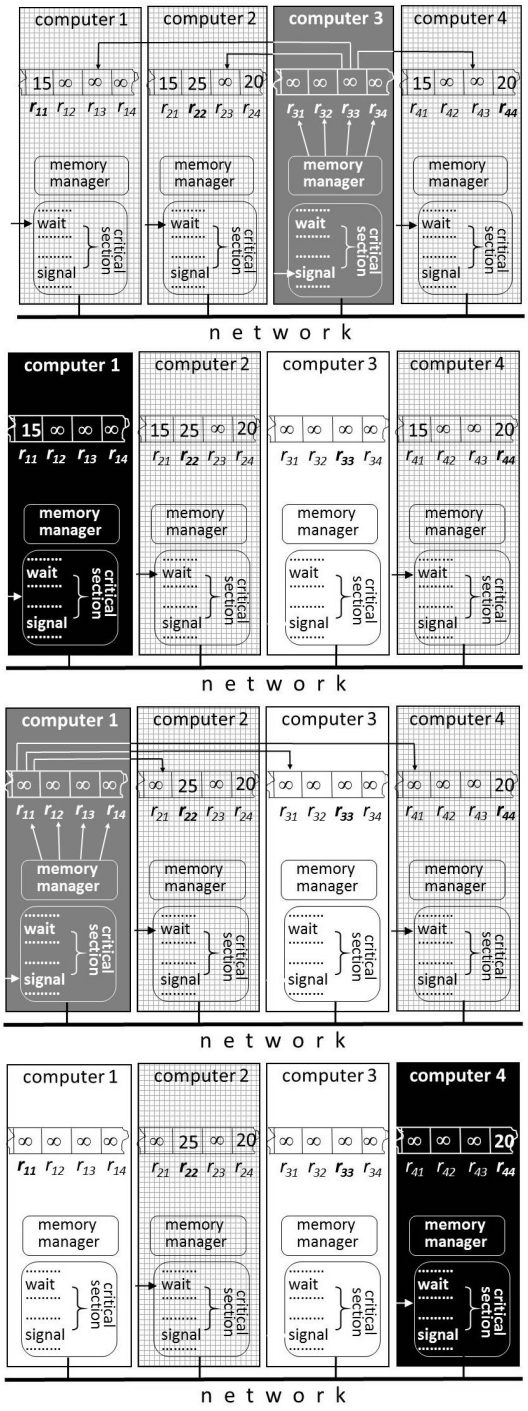


Table 3.1 Exemplary run of a system with four computers using protocol depicted in Fig.3.2. Pattern of the computers' background corresponds to their local states as pictured in the protocol in Fig.3.2



state 5
$[Y,Y,G,Y]$

state 6
$[R,Y,W,Y]$

state 7
$[G,Y,W,Y]$

state 8
$[W,Y,W,R]$

Table 3.1 cont.

Theorem 3.1
In no global state two distinct computers can perform critical section.

Proof. Let on the contrary, in a global state $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n] \in \mathbf{\Omega}^n$ computers of number i and j perform critical section. Then $r_{ii} = min(\overrightarrow{\mathbf{r}_i})$ and $r_{jj} = min(\overrightarrow{\mathbf{r}_j})$ in the local states Q_i and Q_j of these computers. By definition of the global timestamps $r_{ii} \neq r_{jj}$ because events of request for critical section are distinct, so, their global timestamps (i.e. values of r_{ii} and r_{jj}) are also distinct – due to the one-to-one function Γ (Section 3). But because of actions in the stable state B of the protocol in Fig.3.2, $r_{ij} = r_{jj}$ and $r_{ji} = r_{ii}$ hold. Since r_{ii} and r_{jj} are minimal in vectors $\overrightarrow{\mathbf{r}_i}$ and $\overrightarrow{\mathbf{r}_j}$ respectively, so, $r_{ii} \preceq r_{ij}$ and $r_{jj} \preceq r_{ji}$, therefore $r_{ii} \preceq r_{jj}$ and $r_{jj} \preceq r_{ii}$ which implies $r_{ii} = r_{jj}$ (by antisymmetry of $\preceq$ - see Section 2 for definition of the order $\preceq$ between global timestamps) – a contradiction! $\square$

Lemma 3.1
Suppose that in a global state $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n]$, computers of number i and j both are not in the local state W (i.e. $min(\overrightarrow{\mathbf{r}_i}) < \infty$, $min(\overrightarrow{\mathbf{r}_j}) < \infty$) or both are in W (i.e. $min(\overrightarrow{\mathbf{r}_i}) = \infty$, $min(\overrightarrow{\mathbf{r}_j}) = \infty$). Then $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$.

Proof. Let $min(\overrightarrow{\mathbf{r}_i}) < \infty$, $min(\overrightarrow{\mathbf{r}_j}) < \infty$. Then in the global state $\overrightarrow{Q}$, exactly one computer, say of number k, either is performing critical section or is ready to do this, therefore $r_{kk} = min(\overrightarrow{\mathbf{r}_k})$. According to the protocol in Fig.3.2 $r_{ik} = r_{jk} = r_{kk} = min(\overrightarrow{\mathbf{r}_k})$ and r_{ik}, r_{jk} are the least components of vectors $\overrightarrow{\mathbf{r}_i}$, $\overrightarrow{\mathbf{r}_j}$ in the state $\overrightarrow{Q}$. Thus $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_k}) = min(\overrightarrow{\mathbf{r}_j})$. $\square$

Theorem 3.2
Computers enter into the critical section in the order of their requesting for it. This order is independent of the data transmission latency.

Proof. Let $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n]$ be a global state with local states Q_i, Q_j each of them either B or Y, thus $r_{ii} < \infty$, $r_{jj} < \infty$. It is to be proved that if computer i enters into state Q_i before entering of j into state Q_j, i.e. if $r_{ii} < r_{jj}$, then state R would be reached by computer i before computer j. By Lemma 3.1, $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$ in $\overrightarrow{Q}$ and according to the protocol in Fig.3.2, variables r_{ii}, r_{jj} are not changing their values until computers i, j reach state G. Thus, if eventually a global state $\overrightarrow{Q'} = [Q'_1, ..., Q'_i = R, ...Q'_j, ...Q'_n]$ nearest to $\overrightarrow{Q}$ is reached from $\overrightarrow{Q}$ then $min(\overrightarrow{\mathbf{r}_i}) = r_{ii}$ in $\overrightarrow{Q'}$. But if a global state $\overrightarrow{Q"} = [Q"_1, ..., Q"_i, ...Q"_j = R, ...Q"_n]$ had been reached from $\overrightarrow{Q}$ before $\overrightarrow{Q'}$ then $min(\overrightarrow{\mathbf{r}_j}) = r_{jj}$ in $\overrightarrow{Q"}$ would hold. Thus, $min(\overrightarrow{\mathbf{r}_i}) \leq r_{ii} < r_{jj} = min(\overrightarrow{\mathbf{r}_j})$ because values of r_{ii} and of r_{jj} in the state $\overrightarrow{Q"}$ are the same as in $\overrightarrow{Q}$ and $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$ - a contradiction!

Now, note that the value of $min(\overrightarrow{\mathbf{r}_i})$ is established only when values of r_{kk} are completely transmitted from all computers $k \neq i$ to computer i and stored in r_{ik} (see the protocol in Fig.3.2). This value does not depend on duration of these

transmissions nor on their order. Therefore the order of entering computers into critical section is independent of transmission latency but only on the order of their requesting for it. □

Independence of $min(\overrightarrow{\mathbf{r}_i})$ of order as well as latency of transmissions when the run of four computers system shown in Table 3.1, has reached state 1: $[B, W, B, W]$, is illustrated in Fig.3.3(a) and (b). $i\uparrow(r_{kk})_k$ means: "computer k sends value of r_{kk} up to computer i" ; $k\downarrow(r_{ik})_i$ means: "computer i receives a value sent by computer k and stores it in r_{ik}".

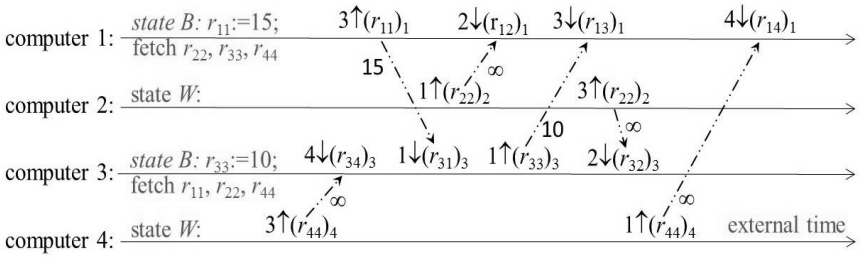


Fig.3.3(a). Diagram of global state $[B, W, B, W]$

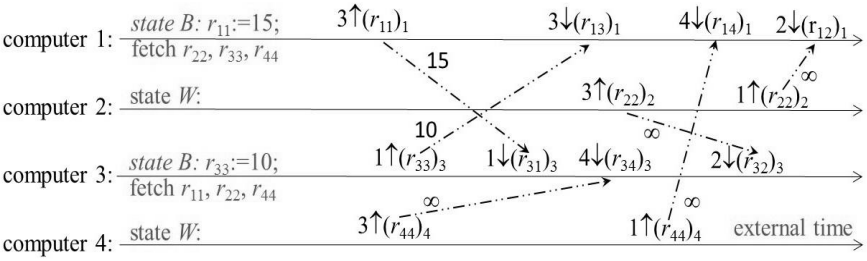


Fig.3.3(b). The same global state and $min(\overrightarrow{\mathbf{r}_i})$ as in (a) but different order and latency of transmissions

Summary

The protocol presented in Fig.3.2 may seem similar to that in [R-A 1981] in that it is fully distributed (without a coordinating server) and because of usage of global timestamps and two-way communication between computer requesting for critical section and remaining computers. However the algorithm proposed here is differently organized: a requesting computer, updates its own vector of timestamps and makes a decision on the basis of its content whether to enter into critical section or wait. Using ∞ as a largest number, not assumed by any timestamp unifies activities when making the decision. Most important properties of the algorithm are formally proved. The algorithm seems suitable for system with Distributed Shared Memory, since problems with data inconsistency do not arise (Theorem 3.2).

References

[Cz 1980] Czaja L., *Deadlock and Fairness in Parallel Schemas: a Set-Theoretic Characterization and Decision Problems*, Information Processing Letters, vol. 10 number 4,5 (1980)

[Cz 2016] Czaja L., *Remarks on Memory Consistency Description*, to appear in Fundamenta Informaticae 2016

[Dij 1968] Dijkstra E.W., *Cooperating sequential processes*, in F. Genuys, ed., Programming Languages: NATO Advanced Study Institute, pp. 43-112, Academic Press, 1968

[Dij 2002] Dijkstra E.W., *The Origin of Concurrent Programming,* (book) Springer Verlag New York, 2002 pp. 65-138

[K-M-C-H 2016] Ihor Kuz, Manuel M. T. Chakravarty & Gernot Heiser, *A course on distributed systems* COMP9243, 2016, 2016

[La 1978] Lamport L., *Time, Clocks and the Ordering of Events in Distributed Systems*, Comm. of the ACM, 1978, 21, pp. 558-565

[La 1979] Lamport L., *How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs*, IEEE Trans. On Computers, 1979 C-28, s. 690-691

[R-A 1981] Ricart G., Agrawala A.K., *An Optimal Algorithm For Mutual Exclusion in Computer Networks*, Comm. of the ACM, 1981, 24, 1, pp. 9-17

[S-R 2003] Saxena P.C., Rai J., A survey of permission-based distributed mutual exclusion algorithms, in Computer Standards & Interfaces, Volume 25, Issue 2, May 2003, Pages 159–181

Author Index

Abdullahi, Ismaila Jihad 9
Akhundov, Jafar 12
Alasgarov, Emin 3

Barylska, Kamila 8

Chrzastowski-Wachtel, Piotr 25
Copik, Marcin 19
Czaja, Ludwik 26

Erofeev, Evgeny 8

Garanina, Natalia 10
Gomolinska, Anna 11
Grochowalski, Piotr 7
Gruska, Damas 14, 20

Haakma, Reinder 3

Jankowski, Andrzej 5

Kacprzak, Magdalena 24
Koutny, Maciej 8
Kozlova, Darya 22

Lanotte, Ruggero 13

Mikulski, Łukasz 8
Müller, Berndt 9

Nguyen, Linh Anh 23
Niewiadomski, Artur 16

Pancerz, Krzysztof 7
Pataky, Mikulas 14
Pelz, Elisabeth 2
Penczek, Wojciech 16
Piatkowski, Marcin 8
Placzek, Stanislaw 21
Polkowski, Lech 6

Rataj, Artur 18, 19
Redziejowski, Roman 1
Rykaczewski, Krzysztof 17

Sawicka, Anna 24
Sidorova, Elena 10
Sidorova, Natalia 3
Skowron, Andrzej 5
Stencel, Krzysztof 17
Szreter, Maciej 15
Szul, Tomasz 7

Tax, Niek 3
Tini, Simone 13
Tröger, Peter 12

Wasilewski, Piotr 5
Werner, Matthias 12
Wisniewski, Piotr 17
Wolski, Marcin 11
Wozna-Szczesniak, Bozena 18, 19

Zakharov, Vladimir 22
Zbrzezny, Agnieszka 4
Zbrzezny, Andrzej 4, 24

Keyword Index

abstract planning 15
adaptive rough set 5
aerospace 12
ambiguity resolution 10
approximation 7
Argumentation 14
argumentative dialogue games 24
Artificial Neural Network (ANN) 21
automated composition of web services 15
axiomatization of semantics 2

bisimilarity 23
bisimulation 23
bisimulation-based comparison 23
boundary 6
bounded model checking 4

circular statistics 3
composition 12
computational complexity 9
concrete planning 16
concurrency 25
coordination 21
coordination principle 21
coordinator structure 21

data analysis 17
decomposition 21
directed similarity 23
Discrete Timed Automata 4
Distributed computer systems 26
Distributed mutual exclusion 26
Distributed shared memory 26
dominance relation 11

Elementary Reference-net System 9
extrapolation 18

finite state automaton 22
formal concept 11
Formal Concept Analysis 10
Formal grammars 1
formal model 24
Future state prognostic 14

GA 16
GEO 16
GPGPU 19
graph database 15

hierarchy 21
Hybrid Automata 13
hybrid automata 12
hybrid systems 12

information flow 20
information retrieval 10
information system 11
instability 25
interleaving 25
Intrusion detection system 14
isomorphic property 9

KDD99 Cup 14

label refinements 3
lexical ambiguity 10
Limited backtracking 1

model checking 22, 24
Monte-Carlo simulation 19
MTL 4
Multi-agent systems 14
Multi-agent temporal logic 14
multiagent systems 10

nets-within-nets 9
non–determinism 18

ontology 7
ontology population 10
opacity 20
optimal policy 18
Over-Approximation 13
OWL 2 7

Parsing Expression Grammar 1
Petri net 8
Petri nets 9
planning tool 16
Prism 19
probabilistic logics 19

probabilistic model checking 18, 19
process algebras 20
process discovery 3
process semantics 2

Reachability 13
reactive system 22
regular language 22
reversibility 8
reversible computation 8
rough mereology 6
rough set 11
rough set based classifier 5
rough set theory 6
rough sets 7

SA 16
SAT 4
Scott information systems 10
security 20
semigroup 22
sFlow 14
simplicial complex 17
simularity 23
simulation 23
SMT 16
sorites paradox 5
state graphs 25
statistical model checking 18, 19
superposition 12
syntactic ambiguity 10

temporal logics 22
Timed Petri Nets 2
Top-down parsing 1
topology 17
transducer 22
trips and travels 16
true concurrency 2
truly rough inclusion 6

unsupervised learning 3

vague concept 5
vagueness 5

web service composition 16