FI MU

Faculty of Informatics
Masaryk University

INFINITY 2002

Pre-Proceedings of 4th International Workshop on Verification of Infinite-State Systems

by

Antonín Kučera
Richard Mayr
(Eds.)

FI MU Report Series FIMU-RS-2002-04

August 2002

INFINITY 2002

Antonín Kučera
Richard Mayr
(Editors)

INFINITY 2002
4th International Workshop on Verification of Infinite-State Systems
A Satellite Workshop of CONCUR'02
Brno, Czech Republic, August 24, 2002
Pre-Proceedings

Preface

INFINITY 2002, the 4th International Workshop on Verification of Infinite-State Systems was held as a satellite workshop of CONCUR 2002 (the 13th International Conference on Concurrency Theory) in Brno, Czech Republic, on August 24, 2002.

The aim of the workshop is to provide a forum for researchers interested in the development of mathematical techniques for the analysis and verification of systems with infinitely many states.

The topics of INFINITY 2002 included the following: techniques for modeling and analysis of infinite-state systems, equivalence-checking and model-checking with infinite-state systems, parameterized systems, calculi for mobility and security, finite-state abstractions of infinite-state systems.

The volume consists of eight contributed papers selected by the INFINITY 2002 programme committee, and short abstracts of four presentations of work-in-progress which accompanied the regular contributed talks. The programme of INFINITY 2002 was further enriched by an invited talk given by Colin Stirling.

We would like to thank the programme committee members for their support in composing the INFINITY 2002 programme, and the CONCUR'02 Organizing Committee for arranging all local affairs.

Final proceedings will appear as volume 68(6) in the ENTCS series published by Elsevier. We thank Michael Mislove, the managing editor of ENTCS, for providing this opportunity.

Brno and Freiburg, August 2002

Antonín Kučera
Richard Mayr

Programme Committee

Parosh Abdulla	Uppsala (SE)
Julian Bradfield	Edinburgh (UK)
Didier Caucal	Irisa (F)
Hubert Comon	LSV Cachan (F)
Giorgio Delzanno	Genova (I)
Yoram Hirshfeld	Tel-Aviv (Israel)
Antonín Kučera (co-chair)	Brno (CZ)
Denis Lugiez	Marseille (F)
Richard Mayr (co-chair)	Freiburg (D)
Bernhard Steffen	Dortmund (D)
P.S. Thiagarajan	Chennai (India)
Moshe Vardi	Rice (USA)

Contents

I. Regular Papers

II. Presentations

Deciding Framed Bisimilarity

Hans Hüttel*
BRICS, Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, 9220 Aalborg Øst, Denmark

July 8, 2002

Abstract

The spi-calculus, proposed by Abadi and Gordon, is a process calculus based on the π-calculus and is intended for reasoning about the behaviour of cryptographic protocols. We consider the finite-control fragment of the spi-calculus, showing it to be Turing-powerful (a result which is joint work with Josva Kleist, Uwe Nestmann, and Björn Victor.) Next, we restrict our attention to finite (non-recursive) spi-calculus. Here, we show that framed bisimilarity, an equivalence relation proposed by Abadi and Gordon, showing that it is decidable for this fragment.

1 Introduction

The spi-calculus, originally proposed by Abadi and Gordon [AG97a], is a process calculus based on the π-calculus [MPW92] and is intended for describing and reasoning about the behaviour of cryptographic protocols.

An important insight of the spi-calculus is that correctness properties can be expressed as statements of behavioural equivalence. For instance, a protocol $P(M)$ transmitting the message x satisfies the secrecy property w.r.t. M if we cannot distinguish between two instances of P which transmit different messages. Expressed using behavioural equivalence, this reduces to stating that

$$\forall M_1, M_2. P(M_1) \sim P(M_2)$$

Deciding correctness properties of cryptographic protocols now amounts to deciding the behavioural equivalence $\sim$.

*hans@cs.auc.dk

Various notions of behavioural equivalence have been put forward. Abadi and Gordon [AG97a] choose *may-testing equivalence* (originally proposed by De Nicola and Hennessy [NH83]). While may-testing is ideal from a philosophical point of view – processes are equivalent iff they behave in the same way under all attacks / observations – this equivalence is defined via universal quantification over observer processes and is therefore less ideal from the perspective of actually determining the equivalence of processes.

Consequently, in [AG98b] Abadi and Gordon define a bisimulation-style equivalence, *framed bisimilarity*, and show it to be as a sound approximation of may-testing equivalence. A main motivation behind their work was to define a notion of behavioural equivalence which has a useful proof technique and is decidable.

The main focus of this paper is to examine to which extent the latter is the case.

As the full spi-calculus is Turing-powerful one can only hope for a positive decidability result within a proper subcalculus. A natural candidate would be the *finite-control* spi-calculus, the spi-calculus counterpart of regular CCS; finite-control processes have a bounded number of parallel components and, because of the presence of recursion, are able to describe multiple protocol runs.

However, even the finite-control spi-calculus is Turing-powerful [HKNV97]. In this paper we first demonstrate this by presenting an encoding of Minsky's two-counter machines into the finite-control calculus, a result which is joint work with Josva Kleist, Uwe Nestmann, and Björn Victor.

Next, we restrict our attention to *finite* spi-calculus processes and show that framed bisimilarity is decidable in this fragment. The finite spi-calculus processes are the recursion-free processes of the spi-calculus, corresponding to single runs of a cryptographic protocol.

In [AL00] Amadio and Lugiez consider a finite spi-calculus similar to ours and show that its associated reachability problem is decidable (albeit NP-hard). As further work they mention finding an algorithm for deciding bisimilarity.

A main problem in obtaining our result stems from matching input transitions, since two processes must be equivalent under *all* value instantiations; we overcome this problem by showing that only finitely many values need be considered.

2 The spi-calculus

The spi-calculus extends the π-calculus [MPW92, Mil99] with primitives for encryption and decryption. As in the π-calculus, communication takes place over channels that can either be public or restricted. Messages may be decrypted; the perfect encryption hypothesis is adopted in the spi-calculus – an attacker cannot guess the key of an encrypted message.

2.1 Syntax

In this section we present the two fragments of the spi-calculus that we shall study in the rest of the paper. Our syntax largely follows that of [AG97a]. We only consider shared key cryptography since the definitions related to framed bisimilarity in [AG98b] only use shared key cryptography. However, an extension of the results in the present paper should be straightforward.

2.1.1 Terms

Common to our two fragments is the set of terms that can be communicated by processes. Unlike the π-calculus, the spi-calculus allows us to communicate composite terms. The set of terms, $\mathcal{T}$, has its syntax defined by the following grammar.

$$L, M, N ::= x \mid n \mid \{M\}_N \mid (M, N)$$

In the above, x ranges over the set of variables, n ranges over the set of *names*, $\{M\}_N$ denotes the term M encrypted using key N and (M, N) denotes the pair whose components are the terms M and N.

2.1.2 The finite-control spi-calculus

The finite-control spi-calculus is a straightforward extension of the finite-control π-calculus introduced by Lin [Lin91].

As the definition below shows, a finite-control process consists of a fixed number of sequential processes running in parallel.

Definition 1 The set of finite-control spi-calculus processes is given by the grammar

$$\begin{array}{rcl} R & ::= & M(x).R \mid \overline{M}\langle N\rangle.R \mid (\nu n)R \\ & \mid & D(M) \mid 0 \mid [M = N]\,R \mid R_1 + R_2 \\ & \mid & \text{let } (x, y) = M \text{ in } R \mid \text{rec } D(M).R \\ & \mid & \text{case } L \text{ of } \{x\}_N \text{ in } R \\ P & ::= & R \mid (\nu n)P \mid P|P \end{array}$$

The spi-calculus distinguishes between variables $x, y, z, \ldots \in \mathcal{V}$ and names $c, m, n, k \ldots \mathcal{N}$. Names refer to a key or a channel, whereas variables are instantiated to messages. When concerning channels, a name c is used for input and its co-name $\overline{c}$ used for output.

The spi-calculus has two communication primitives. $\overline{M}\langle N\rangle.P$ is output; N is emitted on the channel M. $M(x).P$ is input; the variable x is received on the channel M, and x is bound in P.

While encryption is handled at the level of message terms, decryption is a process construct. $\text{case } L \ \text{ of } \{x\}_N \text{ in } P$ is used to decrypt terms; x is bound in P. The other term destructor is $\text{let } (x, y) = M \text{ in } P$ which allows us to split a pair; the variables x and y are bound in P.

The remaining process constructs are also found in the π-calculus: $(\nu n)P$ is the restriction construct. The new name n is bound in P. $P \mid Q$ denotes parallel composition and 0 is the empty process. Finally, the match construct $[M = N]\, P$ can proceed as P iff M is equal to N.

In the finite-control calculus we allow two additional constructs, namely nondeterministic choice, $R_1 + R_2$ and recursively defined processes, $\mathsf{rec}\ D(M).R$. $D(M)$ ranges over recursion constants which may be parameterised by a term.

We identify processes up to renaming of bound names and variables. A process without any free variables is *closed*; we let $\mathcal{P}$ denote the set of closed processes. Furthermore we let $\text{fn}[\![P]\!]$ denote the set of free names in P, and $\text{fv}[\![P]\!]$ the free variables in P. For any set of terms S, we let $n(S)$ denote the set of names occurring in S, free as well as bound. $P[M/x]$ denotes the substitution of the term M for all free occurrences of x in the process P and is defined as expected.

The original presentation of the spi-calculus in [AG97a] introduces natural numbers into the syntax. This, however, is unimportant as we can encode the naturals using encryption and decryption. Let a, b be fresh names. We then let

$$\begin{aligned} [\![0]\!] &= a \\ [\![n+1]\!] &= [\![\{[\![n]\!]\}_b]\!] \end{aligned}$$

The test-for-zero process construct now becomes

$$[\![\text{case } \nu \ \text{ of } 0 : P \ \ \text{suc}(x) : Q]\!] = \text{case } \nu \ \text{ of } \{x\}_b \text{ in } P + [\nu = a]Q$$

In our undecidability proof in section 3 we use natural numbers freely by implicit appeal to this encoding.

2.1.3 Finite processes

The syntax of processes in the finite spi-calculus omits nondeterministic choice and recursion from the finite-control fragment.

$$\begin{array}{rcl} P, Q, R & ::= & (\nu n)P \mid \overline{M}\langle N\rangle.P \mid M(x).P \mid P \mid Q \\ & \mid & [M = N]\, P \mid 0 \mid \text{let } (x, y) = M \text{ in } P \mid \text{case } L \text{ of } \{x\}_N \text{ in } P \end{array}$$

2.1.4 Agents

An agent can be a process, an abstraction or a concretion. The syntax of agents is defined by the following grammar:

$$\begin{array}{rcl} A, B & ::= & P \mid C \mid F \\ F, G & ::= & (x)P \\ C, D & ::= & (\nu\vec{m})\langle M\rangle P \end{array}$$

$(x)P$ is an *abstraction*, which needs to bind a term to x before proceeding. $(\nu\vec{m})\langle M\rangle P$ is a *concretion*, which is immediately able to output the term M. $\mathcal{A}$ will denote the set of closed agents.

2.2 Semantics

Our labelled commitment semantics of the spi-calculus is that of [AG98b].

2.2.1 Reduction and structural congruence

The reduction relation describes how processes unfold and make preparations for a reaction. In particular, the rules describe how the term deconstructors behave (Table 1) and, for finite-control processes, how a recursive process proceeds by unfolding the recursive definition (Table 2) . In the case of a decryption we only proceed if the key is a name. See Table 1.

Structural congruence, $\equiv$, is defined in Table 3. It captures the identities that should intuitively hold.

2.2.2 The commitment relation

The *commitment transition system* $(\mathcal{P}, \{\xrightarrow{\alpha} \mid \alpha \in \mathcal{N} \cup \{\tau\}\}, \mathcal{A})$ has its transition relation defined inductively by the rules in Definition 4.

$$
\begin{array}{rcl}
[M = M] & > & P \\
\mathrm{let}\ (x, y) = (M, N)\ \mathrm{in}\ P & > & P[M/x][N/y] \\
\mathrm{case}\,\{M\}_n\ \mathrm{of}\ \{x\}_n\ \mathrm{in}\ P & > & P[M/x]
\end{array}
$$

Table 1: The reduction rules for term destructors

$$
\mathsf{rec}\ D(x).P \quad > \quad P[\mathsf{rec}\ D(M_i).P/D(M_i)]
$$

Table 2: The reduction rule for recursion

$$
\begin{array}{rrr}
P \mid 0 \equiv P & P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\[2em]
P + (Q + R) \equiv (P + Q) + R & P + Q \equiv Q + R & P + 0 \equiv P \\[2em]
(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P & (\nu n)0 \equiv 0 & P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \quad \text{if } n \notin \mathrm{fn}[\![P]\!] \\[2em]
\dfrac{P > Q}{P \equiv Q} & \dfrac{P \equiv Q \quad Q \equiv R}{P \equiv R} & \dfrac{}{P \equiv P} \\[2em]
\dfrac{P \equiv Q}{Q \equiv P} & \dfrac{P \equiv Q}{P \mid R \equiv Q \mid R} & \dfrac{P \equiv Q}{(\nu n)P \equiv (\nu n)Q}
\end{array}
$$

Table 3: Rules defining structural congruence

In Definition 4 we use the *interaction* operator $\bullet$ defined by

$$C \bullet F \triangleq (\nu\vec{n})(Q \mid P[N/x]) \qquad\qquad F \bullet C \triangleq (\nu\vec{n})(P[N/x] \mid Q),$$

when $\{\vec{n}\} \cap \mathrm{fn}[\![P]\!] = \emptyset$. Here, we extend restriction and composition as follows:

$$\begin{aligned}
(\nu n)(x)P &\triangleq (x)(\nu n)P\\
Q \mid (x)P &\triangleq (x)(Q \mid P)\\
(\nu n)(\nu\vec{m})\langle M\rangle P &\triangleq \begin{cases}(\nu n,\vec{m})\langle M\rangle P & \text{if } n \in \mathrm{fn}[\![M]\!]\\ (\nu\vec{m})\langle M\rangle(\nu n)P & \text{otherwise}\end{cases}\\
Q \mid (\nu\vec{m})\langle M\rangle P &\triangleq (\nu\vec{m})\langle M\rangle(Q \mid P)
\end{aligned}$$

where we assume $x \notin \mathrm{fv}[\![Q]\!]$, $n \notin \{\vec{m}\}$ and $\{\vec{m}\} \cap \mathrm{fn}[\![Q]\!] = \emptyset$. The dual composition $A \mid Q$ is defined symmetrically.

$$(\text{Input}) \quad m(x).P \xrightarrow{m} (x)P \qquad\qquad (\text{Output}) \quad \overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle P$$

$$(\text{Com-1}) \quad \frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F \bullet C} \qquad\qquad (\text{Com-2}) \quad \frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C \bullet F}$$

$$(\text{Par-1}) \quad \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \qquad\qquad (\text{Par-2}) \quad \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}$$

$$(\text{Sum-1}) \quad \frac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A} \qquad\qquad (\text{Sum-2}) \quad \frac{Q \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A}$$

$$(\text{Res}) \quad \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A} \qquad\qquad (\text{Red}) \quad \frac{P > Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

Table 4: The commitment semantics of the spi-calculus

3 The finite-control fragment is Turing-powerful

As the finite-control spi-calculus calculus is the spi-calculus analogue of the finite-control fragment of the π-calculus, introduced by [Lin91], one might expect the situation to be same as in the π-calculus. Here, Dam [Dam97] has shown that late and early bisimilarity [MPW92] as well as open bisimilarity [San96] are all decidable. Dam's result depends on the fact that it is always suffices to consider a finite set of names due to the bounded parallelism of a finite-control process.

However, the finite-control spi-calculus is in fact Turing-powerful, destroying all hope of obtaining positive decidability results for any non-trivial notion of behavioural equivalence. The encoding presented here is joint work with Josva Kleist, Uwe Nestmann, and Björn Victor.

3.1 Encoding two-counter machines in the finite-control fragment

For our proof of this fact, we consider another universal model of computation, namely the two-counter machines of [Min67]. A two-counter machine is a simple imperative program consisting of a sequence of labelled instructions that can modify the values of two nonnegative integer counters, c_0 and c_1. Two instructions are singled out, namely L_{start} and L_{stop}. The program starts with the line L_{start} and halts if L_{stop} is reached. The instruction set consists of two different types of instructions (in the indices of the counter variables we always assume addition and subtraction modulo 2):

1. $L \,:\, c_k := c_k + 1;$ `goto` L_n
2. $L \,:\,$ `if` $c_k = 0$ `then goto` L_n^1 `else` $c_k := c_k - 1;$ `goto` L_n^2

We can always assume that a type 1 instruction has $L \neq L_n$ (if $L = L_n$ the machine would loop forever) and that a type 2 instruction has $L \neq L_n^1$ (here, too, if $L = L_n^1$ the machine would loop forever) and $L \neq L_n^2$ (we can simply duplicate the instruction in question.)

Theorem 2 Any two-counter machine can be simulated in the finite-control spi-calculus.

PROOF: We define an encoding $[\![]\!]$ from two-counter machine instructions into the finite-control spi-calculus. The idea is simply that the two counters are represented by processes and the each instruction corresponds to a process that communicates with the counters.

We assume the following set of names, which we denote by **n**:

- For every instruction label L_n we introduce the name l_n, used to signal a `goto`, and the constant D_{l_n}.
- For counter c_k we introduce the names

  d_k indicating that the counter is decremented

c_k indicating that the counter is incremented

r_k indicating that the value of the counter is being read

A counter c_k is represented as the process

$$C_k = \mathsf{rec}\ D_k(x).(\overline{r_k}\langle x\rangle.D_k(x) + d_k.D_k(x-1) + i_k.D_k(x+1))$$

Instructions are encoded as

$$\begin{aligned}
[\![L : c_k := c_k + 1; \mathtt{goto}\ L_n]\!] &= \mathsf{rec}\ D_l.l.i_k.\overline{l_n}.D_l \\
[\![L :\ \mathtt{if}\ c_k = 0\ \mathtt{then}& \\
\mathtt{goto}\ L_n^1\ \mathtt{else}& \\
c_k := c_k - 1;\ \mathtt{goto}\ L_n^2]\!] &= \mathsf{rec}\ D_l.l.r_k(y).([y=0]\,\overline{l_n^1}.D_n + [y \neq 0]\,\overline{d_k}.\overline{l_n^2}.D_n)
\end{aligned}$$

Suppose that a two-counter machine M is composed of a sequence of instructions $S_1, \ldots, S_m$. Then the encoding of the machine is given by

$$[\![M]\!] = (\nu\mathbf{n})\prod_{i=1}^{m}[\![S_i]\!] \mid C_0 \mid C_1$$

It is now easy see that the two-counter machine can reach a state where $c_0 = \nu_0$ and $c_1 = \nu_1$ if and only if $[\![M]\!] \xrightarrow{\tau}{}^{*} P'$ where the term P' has counter constants whose values are $D_k(\nu_0)$ and $D_k(\nu_1)$, respectively. □

Corollary 3 Any nontrivial notion of behavioural equivalence is undecidable in the finite-control spi-calculus.

4 Framed bisimilarity

Framed bisimilarity was introduced by Abadi and Gordon in [AG98b].

4.1 Frames and theories

Processes are related with respect to a *frame-theory pair* which represents the knowledge of the environment.

Definition 4 A *frame fr* is a finite set of names. A *theory th* is a finite set of pairs of terms (M, N). We let e range over the set of frame-theory pairs.

$$\text{(Ind Var)} \quad \frac{}{e \vdash x \leftrightarrow x}$$

$$\text{(Ind Frame)} \quad \frac{n \in \mathit{fr}}{e \vdash n \leftrightarrow n} \qquad \text{(Ind Theory)} \quad \frac{(M, N) \in \mathit{th}}{e \vdash M \leftrightarrow N}$$

$$\text{(Ind Pair)} \quad \frac{e \vdash M \leftrightarrow M' \quad e \vdash N \leftrightarrow N'}{e \vdash (M, N) \leftrightarrow (M', N')} \qquad \text{(Ind Enc)} \quad \frac{e \vdash M \leftrightarrow M' \quad e \vdash N \leftrightarrow N'}{e \vdash \{M\}_N \leftrightarrow \{M'\}_{N'}}$$

Table 5: Rules defining the indistinguishability relation

Intuitively, when comparing processes P and Q, the elements of the frame are the names from P and Q that the attacker knows. If $(M, N) \in \mathit{th}$ the attacker cannot distinguish the term M coming from P and the term N coming from Q.

In what follows, when given an environment e we refer to its frame part as fr_e and its environment part as th_e.

Definition 5 Let $e = (\mathit{fr}, \mathit{th})$ be an environment. Terms M and N are indistinguishable under e, written $e \vdash M \leftrightarrow N$, if it can be derived by the rules in Table 5.

An environment must be consistent. This is captured by

Definition 6 Environment e is ok, written $e \vdash \text{ok}$, if:

1. $\forall (M, N) \in \mathit{th}$ it must hold that M is closed, $\exists M_1, M_2 : M = \{M_1\}_{M_2}$ and $\nexists N_2 : e \vdash M_2 \leftrightarrow N_2$. The converse must also hold for N.
2. whenever $(M, N) \in \mathit{th}$ and $(M', N') \in \mathit{th}$, $M = M'$ iff $N = N'$.

Definition 7 Let e and e' be environments. e' extends e, written $e \leq e'$, iff $\forall M, N : e \vdash M \leftrightarrow N \Rightarrow e' \vdash M \leftrightarrow N$.

A *framed process pair* is a quadruple $(\mathit{fr}, \mathit{th}, P, Q)$, where $P, Q \in \mathcal{P}$. If $\mathcal{R}$ is a set of framed process pairs, we write $e \vdash P\mathcal{R}Q$ when $(\mathit{fr}, \mathit{th}, P, Q) \in \mathcal{R}$. A *framed relation* is a set $\mathcal{R}$ of framed process pairs, such that $e \vdash \text{ok}$ whenever $e \vdash P\mathcal{R}Q$.

4.2 Framed simulations and bisimulations

Framed simulation is a *late* simulation [MPW92]; the choice of a matching transition for an input transition does not depend on the value that will eventually be received.

Definition 8 A *framed simulation* is a framed relation $\mathcal{S}$ such that, whenever $e \vdash P\mathcal{S}Q$, the following three conditions hold

1. If $P \xrightarrow{\tau} P'$ then there exists a process Q' such that $Q \xrightarrow{\tau} Q'$ and $e \vdash P'\mathcal{S}Q'$.
2. If $P \xrightarrow{c} (x)P'$ and $c \in \mathit{fr}$ then there exists an abstraction $(x)Q'$ with $Q \xrightarrow{c} (x)Q'$ and, for all sets $\{\vec{n}\}$ disjoint from $\mathrm{fn}[\![P]\!] \cup \mathrm{fn}[\![Q]\!] \cup f \cup \mathrm{fn}(\mathit{th})$ and all closed terms M and N, if $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash M \leftrightarrow N$ then $(f \cup \{\vec{n}\}, \mathit{th}) \vdash P'[M/x]\mathcal{S}Q'[N/x]$.
3. If $P \xrightarrow{\overline{c}} (\nu\vec{m})\langle M\rangle P'$, $c \in \mathit{fr}$ and $\{\vec{m}\} \cap (\mathrm{fn}[\![P]\!] \cup \mathrm{fn}(\pi_1(\mathit{th})) \cup \mathit{fr}) = \emptyset$ then there exists a concretion $(\nu\vec{n})\langle N\rangle Q'$ with $Q \xrightarrow{\overline{c}} (\nu\vec{n})\langle N\rangle Q'$ and $\{\vec{n}\} \cap (\mathrm{fn}[\![Q]\!] \cup \mathrm{fn}(\pi_2(\mathit{th})) \cup f) = \emptyset$. Furthermore $\exists e' : e \leq e'$, $e' \vdash M \leftrightarrow N$, and $e' \vdash P'\mathcal{S}Q'$.

Definition 9 A *framed bismulation* is a framed simulation $\mathcal{S}$ such that $\mathcal{S}^{-1} = \{e' \vdash Q\mathcal{S}P \mid e \vdash P\mathcal{S}Q \wedge e' = (\mathit{fr}, \{(M,N) \mid (N,M) \in \mathit{th}\})\}$ is also a framed simulation.

Definition 10 Framed bisimilarity is the greatest framed bisimulation, written $\sim_f$.

5 A decidability result

Definitions 8 and 9 do not provide us with a straightforward means of checking bisimilarity. The goal of the rest of our paper is to address this issue. More precisely, we shall show that in the case of finite processes

- we only need to consider finitely many terms when matching input transitions.
- we only need to consider finitely many possible frame extensions when matching input transitions
- we only need to consider finitely many frame-theory extensions when matching output transitions

Taken together, these observations will allow us to obtain a simple decision procedure for framed bisimilarity.

5.1 Matching input transitions

Assume that we are trying to determine whether $(\mathit{fr}, \mathit{th}) \vdash P \sim_f Q$. We have an input commitment $P \xrightarrow{c} (x)P'$, have a candidate for a matching commitment, $Q \xrightarrow{c} (x)Q'$, and now need to determine whether $P' \sim_f Q'$.

Assume that the maximal number of successive term destructors in P and Q is m, and that the maximal number of term constructors of any term in *th* is d. Then we need only consider the finitely many terms of depth $\leq m + d$ constructed from $(\mathit{fr}, \mathit{th})$ and a bounded number of new names in order to determine if $(\mathit{fr}, \mathit{th}) \vdash P' \sim_f Q'$. This must hold as the process can only inspect any input term up to m levels of encryption / pairing and because the environment may ask us to regards terms whose depth is up to d as indistinguishable.

5.1.1 The depth of terms and processes

The notion of the maximal constructor depth of a term is as expected. It counts the *level of encryption* and the *level of pairing*. The level of decryption takes precedence over the level of pairing and only the level of decryption within the contents of a ciphertext matters, as terms appearing in key position must be names. Otherwise, they will cause the process not to evolve any further.

Definition 11 The *maximal constructor depth* $d(M)$ of a term M is defined inductively by the clauses

$$\begin{aligned} d(n) &= 0 \\ d(x) &= 0 \\ d(\{M\}_N) &= d(M) + 1 \\ d((M, N)) &= \max(d(M), d(N)) \end{aligned}$$

The above definition easily extends to frame-theory pairs.

Definition 12 Let $(\mathit{fr}, \mathit{th})$ be a frame-theory pair where $\mathit{fr} = \{(M_1, N_1), \ldots, (M_k, N_k)\}$. The *maximal constructor depth* of $(\mathit{fr}, \mathit{th})$ is defined b

$$d((\mathit{fr}, \mathit{th})) = \max\{\max(d(M_i), d(N_i)) \mid 1 \leq i \leq k\}$$

The maximal destructor depth of a process P is the maximal number of encryptions and pairing operators that can ever be removed along the process P. Decryption and pair splitting operations each contribute by 1, whereas a parallel composition $P \mid Q$ may contribute with decryptions from both P and Q.

Definition 13 Let P be a finite process. The *maximal destructor depth* of P is denoted by $\mathrm{mdd}(P)$ and defined inductively by the clauses

$$\begin{array}{rcl}
\mathrm{mdd}(0) & = & 0 \\
\mathrm{mdd}((\nu n)P) & = & \mathrm{mdd}(P) \\
\mathrm{mdd}(\overline{M}\langle N\rangle.P) & = & \mathrm{mdd}(P) \\
\mathrm{mdd}(M(x).P) & = & \mathrm{mdd}(P) \\
\mathrm{mdd}(P \mid Q) & = & \mathrm{mdd}(P) + \mathrm{mdd}(Q) \\
\mathrm{mdd}([M = N]\,P) & = & \mathrm{mdd}(P) \\
\mathrm{mdd}(\mathrm{let}\ (x, y) = M\,\mathrm{in}\,P) & = & \mathrm{mdd}(P) + 1 \\
\mathrm{mdd}(\mathrm{case}\,L\ \ \mathrm{of}\,\{x\}_N\,\mathrm{in}\,P) & = & \mathrm{mdd}(P) + 1
\end{array}$$

5.1.2 d-framed bisimilarity

d-framed bisimilarity is a variant of framed bisimilarity that only requires input transitions to be matched for transmitted message terms up to a certain depth.

Definition 14 Let k be a nonnegative integer and let e be a frame-theory pair such that $e \vdash \mathrm{ok}$. We write $e \vdash M \leftrightarrow^k N$ if $e \vdash M \leftrightarrow N$ and $\max(d(M), d(N)) = k$. Whenever $e \vdash M \leftrightarrow^k N$ we say that M and N are k*-indistinguishable in* e*.*

Since we only consider terms up to a certain depth, we need only consider finitely many extensions of the frame. This is expressed in the following lemma.

Lemma 15 Let $(\mathit{fr}, \mathit{th})$ be a frame-theory pair and assume that $\max(d(M), d(N)) = k$. If there is a $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th})$ such that $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash M \leftrightarrow^k N$, then we may choose a $\{\vec{n}\}$ where $|\vec{n}| \leq 2k$ satisfying $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash M \leftrightarrow^k N$.

PROOF: If M and N are not indistinguishable under $(\mathit{fr}, \mathit{th})$, this must be amended by applying the constructor rules, the rule (Ind Theory) and the rule (Ind Frame) to new names. Every application of a constructor rule can introduce at most two new names, so at most $2k$ new names can be introduced. □

Lemma 15 leads to the following definition of d-framed simulation.

Definition 16 For any nonnegative integer d, a d*-framed simulation* is a framed relation $\mathcal{S}$ such that, whenever $(\mathit{fr}, \mathit{th}) \vdash P\mathcal{S}Q$, the following three conditions hold

1. If $P \xrightarrow{\tau} P'$ then there exists a process Q' such that $Q \xrightarrow{\tau} Q'$ and $e \vdash P'\mathcal{S}Q'$.

2. If $P \xrightarrow{c} (x)P'$ and $c \in \mathit{fr}$ then there exists an abstraction $(x)Q'$ with $Q \xrightarrow{c} (x)Q'$ and, for all sets $\{\vec{n}\}$ disjoint from $\mathrm{fn}[\![P]\!] \cup \mathrm{fn}[\![Q]\!] \cup \mathit{fr} \cup \mathrm{fn}(\mathit{th})$ such that $|\vec{n}| \leq 2d$ and all closed terms M and N, if $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash M \leftrightarrow^i N$ and $0 \leq i \leq d$ then $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash P'[M/x]\mathcal{S}Q'[N/x]$.

3. If $P \xrightarrow{\overline{c}} A \equiv (\nu\vec{m})\langle M\rangle P'$, $c \in \mathit{fr}$ and $\{\vec{m}\} \cap (\mathrm{fn}[\![Q]\!] \cup \mathrm{fn}(\pi_1(\mathit{th})) \cup \mathit{fr}) = \emptyset$ then there is a concretion $B \equiv (\nu\vec{n})\langle N\rangle Q'$ such that $Q \xrightarrow{\overline{c}} B$, the set $\{\vec{n}\}$ is disjoint from $\mathrm{fn}[\![Q]\!] \cup \mathrm{fn}(\pi_2(\mathit{th})) \cup \mathit{fr}$ and $e' \vdash P'\mathcal{S}Q'$ for some $e' \geq (\mathit{fr}, \mathit{th})$ where $e' \vdash M \leftrightarrow N$.

Definition 17 A d-*framed bisimulation* is a d-framed simulation $\mathcal{S}$ such that $\mathcal{S}^{-1} = \{e' \vdash Q\mathcal{S}P \mid e \vdash P\mathcal{S}Q \wedge e' = (\mathit{fr}_e, \{(M, N) \mid (N, M) \in \mathit{th}_e\})\}$ is also a d-framed simulation.

Definition 18 d-framed bisimilarity is the greatest d-framed bisimulation, written $\sim_f^d$.

Our goal is to show that for finite processes P and Q we have that P and Q are framed bisimilar iff they are d-bisimilar where d is the *critical depth*.

The critical depth of (e, P, Q) is the maximal depth of terms that must be considered as inputs when determining whether P and Q are framed bisimilar under e.

Definition 19 Let (e, P, Q) be a framed process pair. The *critical depth* of (e, P, Q) is defined by

$$\mathrm{cd}(e, P, Q) = \mathrm{d}(e) + \max(\mathrm{mdd}(P), \mathrm{mdd}(Q))$$

We let

$$\mathrm{cd}(e, P) = \mathrm{cd}(e, P, P)$$

When considering the result of an input commitment, we only need to consider instantiations with terms whose depths do not exceed the critical depth. Intuitively, this suffices as all subterms occurring below the critical depth are inaccessible by the destructors of a process.

If two terms are indistinguishable, their subterms appearing at depth d can be replaced by fresh names for any d such that the resulting terms will still be indistinguishable. This is the idea behing d-pruning.

Example 20 Let $M = \{\{a\}_b\}_c$ and $N = \{\{d\}_e\}_f$ and assume that we have $(M, N) \in \mathit{th}$ for some theory th. Let $\mathit{fr} = \{h\}$. Then we have $(\mathit{fr}, \mathit{th}) \vdash \{M\}_h \leftrightarrow \{N\}_h$. We also have $(\mathit{fr} \cup \{g\}, \mathit{th}) \vdash \{\{g\}_g\}_h \leftrightarrow \{\{g\}_g\}_h$ where g is a fresh name not found in fr. $((\mathit{fr} \cup \{g\}, \mathit{th}), \{g\}_h \leftrightarrow \{g\}_h)$ is the 1-pruning of (e, M, N).

The pruning of a pair of terms (M, N) at depth d generates a pair of pruned terms (M', N'). M' and N' are constructed by replacing subterms appearing at levels greater than d by encryptions of arbitary fresh names by the same fresh names. The fresh names are then added to the frame.

Definition 21 Let M and N be closed terms and let $e \vdash \text{ok}$. Further assume that $e \vdash M \leftrightarrow N$, that all subterms appearing in key position in M and N are names and that d is a nonnegative integer. The d-*pruning* of (e, M, N), denoted by $\text{pr}_d((e, M, N))$, is defined inductively by the clauses

$$\begin{array}{lcll}
\text{pr}_0(((fr, th), n, n)) & = & ((fr, th), n, n) & \\
\text{pr}_0(((fr, th), M, N)) & = & ((fr, th), M, N) & \text{if } (M, N) \in th \\
\text{pr}_0(((fr, th), M, N)) & = & ((fr \cup \{a\}, th), \{a\}_a, \{a\}_a) & \begin{array}{l}\text{if } (M, N) \notin th \\ \text{and } a \text{ is fresh}\end{array} \\
\text{pr}_{d+1}((fr, th), \{M_1\}_k, \{N_1\}_k) & = & (e', \{M'\}_k, \{N'\}_k) & \begin{array}{l}\text{where } (e', M', N') = \\ \text{pr}_d(((fr, th), M_1, N_1))\end{array}
\end{array}$$

If M is an open term, we define $\text{pr}_d((e, M)) = (e, M)$.

The pruning operator extends to single terms by defining $\text{pr}_d((e)(M)) = \text{pr}_d((e)(M, M))$.

Note that, because of the usage of unspecified fresh names, the pruning operator as defined here does not generate a unique pair of terms. This can be dealt with by means of introducing suitable bookkeeping.

Note also how the definition exploits the fact that only names are allowed in key position.

Lemma 22 If $e \vdash M \leftrightarrow N$, $d = \max(d(M), d(N))$ and $\text{pr}_d((e, M, N)) = (e', M', N')$ then $e' \vdash M' \leftrightarrow^d N'$.

PROOF: A straightforward induction in d, appealing to Definition 21. □

We can extend the pruning operation to pairs of term vectors. This is done inductively; we prune the components of the vectors successively, extending the frame as we proceed.

Definition 23 Let $|\vec{M}| = |\vec{N}| = k$. Then $\text{pr}_d((\vec{M}, \vec{N}))$ is defined inductively by

$$\text{pr}_d((e, (M_1, \ldots, M_k), (N_1, \ldots, N_k))) = (e', (M_1', \ldots, M_k'), (N_1', \ldots, N_k'))$$

where

$$(e'', M_1', N_1') = \mathrm{pr}_d((e, M_1, N_1))$$

and

$$(e', (M_2', \dots, M_k'), (N_2', \dots, N_k')) = \mathrm{pr}_d((e'', (M_2, \dots, M_k), (N_2, \dots, N_k)))$$

Lemma 24 Let P be a process such that $P = A[\vec{M}/\vec{x}]$ and let $d = \mathrm{cd}(e, P)$. $P > A$ iff $P_1 > A_1$ where $P_1 = A[\vec{N}/\vec{x}]$ where $\mathrm{pr}_d((e, \vec{M})) = (e', \vec{N})$ and $A_1 = A[\vec{N}/\vec{x}]$.

PROOF: Both implications are seen to hold by an inspection of the clauses in the definition of the reduction relation. The interesting case is the decryption clause:

$$\mathrm{case}\,\{M\}_k \text{ of } \{y\}_k \,\mathrm{in}\, P' > P'[M/y]$$

If $P = \mathrm{case}\,\{M\}_k$ of $\{y\}_k \,\mathrm{in}\, P'$, then the definition of the pruning operator tells us that $P_1 = \mathrm{case}\,\{N\}_k$ of $\{y\}_k \,\mathrm{in}\, P_1'$ where $P' = A_1'[\vec{M}/\vec{x}]$ and $P_1' = A'[\vec{N}/\vec{x}]$ for some A_1'. We now see that

$$\mathrm{case}\,\{N\}_k \text{ of } \{y\}_k \,\mathrm{in}\, P_1' > P_1'[N/y]$$

□

Lemma 25 Let $P = A[\vec{M}/\vec{x}]$ and let $d = \mathrm{cd}(e, P)$. $P \xrightarrow{\alpha} A'$ iff $P_1 \xrightarrow{\alpha} A_1'$ where $P_1 = A[\vec{N}/\vec{x}]$ where $\mathrm{pr}_d((e, \vec{M})) = (e', \vec{N})$ and $A_1' = B[\vec{N}/\vec{x}]$ and $A' = B[\vec{M}/\vec{x}]$ for some B.

PROOF: In the case of both implications, the proof proceeds by transition induction. The induction hypothesis in the case concerning the rule (Red) uses Lemma 24. The only other interesting cases are the prefix axioms. □

Theorem 26 Let P and Q be finite spi processes and let $d = \mathrm{cd}(e, P, Q)$ where $e \vdash \mathrm{ok}$. We have that $e \vdash P \sim_f Q$ iff $e \vdash P \sim_f^d Q$.

PROOF: By definition, any framed bisimulation is also a d-framed bisimulation. It therefore suffices to establish that $e \vdash P \sim_f Q$ whenever $e \vdash P \sim_f^d Q$. We show that

$$\mathcal{R} = \left\{ (e, P, Q) \;\middle|\; \exists e', A, B, \vec{M}, \vec{N}. \quad \begin{array}{l} P = A[\vec{M}/\vec{x}], Q = B[\vec{N}/\vec{y}] \\ e' \vdash A[\vec{M}'/\vec{x}] \sim_f^n B[\vec{N}'/\vec{y}] \\ (e', (\vec{M}', \vec{N}')) = \mathrm{pr}_d((e, \vec{M}, \vec{N})) \\ d = \mathrm{cd}(e, P, Q) \end{array} \right\}$$

is a framed bisimulation. This follows from Lemma 25. □

5.2 Matching output transitions

Next, we have to deal with matching output transitions. Fortunately, there are only finitely many candidates for an environment extension in the case of the output clause.

Unfortunately, as was shown in [BN02], the characterization of framed bisimilarity presented in [EHHN99] is sound but not complete. We are therefore unable to fall back on the algorithm for computing environment extensions presented in [EHHN99]. Instead we use

Lemma 27 Let $e \vdash \mathrm{ok}$ and let $M, N \in \mathcal{T}$. It is decidable whether there is an $e \leq e'$ such that $e \vdash M \leftrightarrow N$.

PROOF: To construct an e' such that $e' \vdash M \leftrightarrow N$, we only need to add pairs of the form (M_1, N_1) where $\max(d(M_1), d(N_1)) \leq \max(d(M), d(N))$ and such that $n[\![M_1]\!] \cup n[\![N_1]\!] \subseteq n[\![M]\!] \cup n[\![N]\!]$. Only finitely many such candidate pairs exist. □

6 Deciding framed bisimilarity

We can now state the main results of our paper.

Theorem 28 Let $e \vdash \mathrm{ok}$ and let P and Q be finite spi-calculus processes. For any $d \geq 0$ it is decidable whether $e \vdash P \sim_f^d Q$.

PROOF: Table 6 presents a nondeterministic recursive algorithm $\mathcal{B}((e,(P,Q))$ for determining if $e \vdash P \sim_f^d Q$.

As the algorithm encodes the 'bisimulation game' of Definition 16, $e \vdash P \sim_f^d Q$ iff there exists a successful evaluation of $\mathcal{B}((e,(P,Q)))$. The algorithm always terminates, as Lemma 15 and Lemma 27 guarantee that the checks performed in the conditional statements of the algorithm are effective and as all transition sequences examined along recursive calls are finite due to the absence of recursion. □

Corollary 29 Let $e \vdash \mathrm{ok}$ and let P and Q be finite spi-calculus processes. It is decidable whether $e \vdash P \sim_f Q$.

7 Conclusions and further work

In this paper we have shown that framed bisimilarity is decidable for finite processes. The ideas used in this paper are closely related to those employed in giving *symbolic semantics* to process calculi. The precise relationship is a topic for further work.

```
$\mathcal{B}(((\mathit{fr}, \mathit{th}), (0, 0))) \quad = \mathtt{tt}$
$\mathcal{B}(((\mathit{fr}, \mathit{th}), (P_1, P_2))) =$
            let $(\mathit{fr}, \mathit{th}) = e$ in
                for each $P_i \xrightarrow{a} (x)P'_i$ where $a \in \mathit{fr}$
                        select a $P_{i+1} \xrightarrow{a} (y)P'_{i+1}$
                        if no such $P'_{i+1}$ exists
                        then fail
                        else
                        for each $\vec{n}$ where $|\vec{n}| \leq d, \vec{n} \cap \mathrm{fn}[\![P_i]\!] \cup \mathrm{fn}[\![P_{i+1}]\!] \cup \mathrm{fn}(\mathit{th}) = \emptyset$
                            for each $(\mathit{fr} \cup \{\vec{n}\}, \mathit{th}) \vdash M \leftrightarrow^{d} N$
                            $\mathcal{B}(((\mathit{fr} \cup \{\vec{n}\}, \mathit{th}), (P'_i[M/x], P'_{i+1}[N/y])))$
                for each $P_i \xrightarrow{\overline{a}} (\nu\vec{c})\langle M \rangle P'_i$ where $a \in \mathit{fr}$
                        select a $P_{i+1} \xrightarrow{\overline{a}} (\nu\vec{d})\langle N \rangle P'_{i+1}$
                        if no such $P'_{i+1}$ exists
                        then fail
                        else
                        select $e \leq (\mathit{fr}', \mathit{th}')$ such that $(\mathit{fr}', \mathit{th}') \vdash M \leftrightarrow N$
                            $\mathcal{B}(((\mathit{fr}', \mathit{th}'), P'_i, P'_{i+1}))$
                for each $P_i \xrightarrow{\tau} P'_i$
                        select a $P_{i+1} \xrightarrow{\tau} P'_{i+1}$
                        if no such $P'_{i+1}$ exists
                        then fail
                        else
                            $\mathcal{B}(((\mathit{fr}, \mathit{th}), P'_i, P'_{i+1}))$
```

Table 6: A nondeterministic algorithm for checking bisimilarity

Recent, currently unpublished results [FN01, BN02] establish that the *environment sensitive bisimilarity* of Boreale et al. [BDP99] corresponds to *hedged bisimilarity*, the variant of framed bisimilarity that omits the frame-component. We therefore conjecture that our results and techniques carry over to environment sensitive bisimilarity.

A topic for further work is how to develop an efficient version of the bisimulation checking algorithm. However, framed bisimulation subsumes the late bisimulation equivalence of the π-calculus and the decision problem for this latter equivalence is known to be PSPACE-complete for a number of recursion-free process calculi with value-passing [BT00].

As we have omitted recursion, we can only study attacks that involve a given number of runs of a protocol. Another topic for further work is therefore to study the class of attacks that can be detected within the finite spi-calculus.

Acknowledgements I would like to thank Josva Kleist for his careful reading of an earlier version of this paper.

References

[AG97a] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi-calculus. In *Fourth ACM Conference on Computer and Communications Security*. ACM Press, 1997.

[AG98b] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4), pp. 267-303, Winter 1998.

[AL00] R.M. Amadio, D. Lugiez. On the reachability problem in cryptographic protocols *Proceedings of CONCUR 00*, LNCS 1877, Springer-Verlag.

[BDP99] Boreale, Michele & De Nicola, Rocco & Pugliese, Rosario. Proof Techniques for Cryptographic Processes (Extended version). *Proceedings of LICS 99*, pp. 157–166, 1999.

[BT00] M. Boreale and L. Trevisan A Complexity Analysis of Bisimilarity for Value-passing Processes *Theoretical Computer Science*, Vol. 238, Number 1-2, pp. 313-345, May 2000.

[BN02] J. Borgström and U. Nestmann On Bisimulations for the Spi Calculus Submitted for publication.

[Dam97] Mads Dam On the Decidability of Process Equivalences for the π-Calculus *Theoretical Computer Science*, vol. 183, 1997, pp. 215–228.

[EHHN99] A.S. Elkjær, H. Hüttel, M. Höhle and K. O. Nielsen. Towards automatic bisimilarity checking in the spi calculus. Proceedings of DMTCS'99 and CATS'99. *Australian Computer Science Communications*, 21(3), Springer, 1999.

[FN01] U. Frendrup and J. Nyholm Jensen. Bisimilarity in the Spi-Calculus Masters' Thesis, Department of Computer Science, Aalborg University, June 2001.

[HKNV97] Hans Hüttel, Josva Kleist, Uwe Nestmann and Björn Victor. A symbolic semantics for the spi-calculus. Unpublished manuscript.

[Lin91] Huimin Lin Complete Proof Systems for Observation Congruences in Finite-Control Pi-calculus. In Kim G. Larsen and Mogens Nielsen (editors), *Automata, Languages and Programming, 25th Colloquium*. Volume 1443 of *Lecture Notes in Computer Science*, pages 443-454, Aalborg, Denmark, July 1998, Springer-Verlag.

[MPW92] Robin Milner, Joachim Parrow and David Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, vol. 100(1), 1992, pp. 1–77.

[Mil99] Robin Milner. Communicating and mobile systems: the π-calculus. Cambridge University Press, 1999.

[Min67] Marvin Minsky. Computation: Finite and Infinite Machines. Prentice-Hall 1967.

[NH83] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalence for processes. In Josep Díaz (editor) *Automata, Languages and Programming, 10th Colloquium*. Volume 154 of *Lecture Notes in Computer Science*, pages 548–560, Barcelona, Spain, 18–22 July 1983. Springer-Verlag.

[San96] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, vol. 33, 1996, pp. 69–97.

Modifications of Expansion Trees for Weak Bisimulation in BPA

Jitka Stříbrná Ivana Černá [*]

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic

1 Introduction

The purpose of this work is to examine the decidability problem of weak bisimilarity for BPA-processes. It has been known that strong bisimilarity, which may be considered a special case of weak bisimilarity, where the internal (silent) action τ is treated equally to observable actions, is decidable for BPA-processes ([1, 2, 4]). For strong bisimilarity, these processes are finitely branching and so for two non-bisimilar processes there exists a level n that distinguishes the two processes. Additionally, from the decidability of whether two processes are equivalent at a given level n, semidecidability of strong non-bisimilarity directly follows. There are two closely related approaches to semidecidability of strong equivalence: construction of a (finite) bisimulation or expansion tree and construction of a finite Caucal base. We have attempted to find out if any of the above mentioned approaches could be generalized to (semi)decide weak bisimilarity.

For weak bisimulation we need to consider separately semidecidability of bisimilarity and semidecidability of non-bisimilarity. To be more precise, in case of strong bisimulation the latter is guaranteed by the finite-image property which for weak equivalence fails to hold. Therefore, in the following we will only consider semidecidability of weak bisimulation equivalence.

The technique of bisimulation trees was proposed by Hirshfeld in [6]. In the most general concept, bisimulation trees contain all possible derivative pairs of some initial pair. Hence the trees are complete and correctness is obviously maintained, however it may not be feasible to search such trees. In order to reach algorithmic feasibility it appears necessary to

[*] Supported by GA ČR grant no. 201/00/0400

introduce some modification into the construction of bisimulation trees. There are two kinds of rules summarized by Jančar and Moller in [8]: omission and replacement. We can omit a pair from a reached node if it is in some sense implied by already visited pairs. We can replace one pair by a set of pairs in a newly constructed sibling node if we do not introduce "false" bisimulation witnesses in this process. It now has to be proved that completeness and correctness are maintained which is done by introducing an inductive invariant. The method has been further modified for weak bisimulation and totally normed BPA in[7] where the criteria for omission and replacement have been modified to comply with properties of weak bisimulation. In this work we formulate additional rules to cope with weak bisimilarity of (general) BPA and prove their correctness. We discuss the question whether these modifications are strong enough, i.e. whether they always guarantee the existence of a finite witness of bisimilarity.

The Caucal base (i.e. a set of pairs that would generate the maximal bisimulation by congruence closure - for more details consult [3, 4]) is used to semidecide strong bisimulation by enumeration of finite sets for which the Caucal condition is tested. In this way, in the positive case a finite bisimulation (Caucal) base is eventually constructed. The notion of Caucal base can be modified into weak Caucal base which serves as generation base for the maximal weak bisimulation equivalence. However, we can construct a pair of two weakly bisimilar processes for which there does not exist a finite weak (Caucal) bisimulation base, which indicates that it cannot always be used efficiently for weak bisimilarity.

The paper is structured as follows. In Section 2 we give basic definitions of BPA, bisimulation equivalences and approximation of bisimulations. Section 3 describes decompositions and bisimulations up to which are the core notions for expansion trees. The rules for creating an expansion tree are described and their correctness is proved in Section 4. Section 5 is devoted to a discussion concerning applicability of those rules.

2 Background

In order to define Basic Process Algebras we presuppose a finite set of *actions* Act that contains a special action τ, and a finite set of process variables or atoms Σ. A *Basic Process Algebra (BPA)* is then a pair (Σ^*, Δ), where Σ^* is the free monoid generated by Σ, and $\Delta = \{X \xrightarrow{a} \alpha \mid X \in \Sigma, \alpha \in \Sigma^*, a \in Act\}$ is a finite set of transitions. BPA-processes are identified with words from Σ^*. We will use capital letters X, Y to range over process variables, $\alpha, \beta, \gamma, \delta$ to range over BPA-processes and a, b, c to range over process actions. The empty word ϵ denotes the *empty* process that cannot perform any action.

The transition rules of Δ determine a *transition relation* $\longrightarrow$ on general BPA-processes:

$$X\beta \xrightarrow{a} \alpha\beta \text{ iff there is a rule } X \xrightarrow{a} \alpha \text{ in } \Delta.$$

A *weak transition relation* $\Longrightarrow$ is defined as

$$\overset{a}{\Longrightarrow} \overset{\text{def}}{=} \begin{cases} (\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau \end{cases}$$

If α is a process then the *norm* of α, denoted by $|\alpha|$, is the minimum of lengths of derivation sequences leading from α to the empty process ϵ. We say that a process is *normed* if it has a finite norm, otherwise it is *unnormed*. We also call this notion *strong norm* to distinguish it from *weak norm* which does not count the τ-moves on the way to ϵ, and is denoted by $\|\alpha\|$. When weak norm is considered, a process is called *normed* if it has a finite norm, *totally normed* if the norm is finite and positive, and *unnormed* otherwise.

Each process α of a BPA (Σ^*, Δ) generates a labeled transition system (LTS) with α labeling the root, processes derivable from α labeling the nodes and the action leading from α to α' labeling the arc that leads from α to α'. If two processes give rise to labeled transition systems that are isomorphic up to different names at the nodes then the processes are considered identical. Usually we want to identify a broader class of processes, namely processes which exhibit the same observable behavior. We will investigate two of the major equivalences: strong and weak bisimulations ([9, 10]).

Definition 1 *Let (Σ^*, Δ) be a BPA. A binary relation $\mathcal{R}$ over Σ^* is a* weak bisimulation *if for every pair (α, β) from $\mathcal{R}$ and and every action a from Act the following holds:*
— for every $\alpha \overset{a}{\Longrightarrow} \alpha'$ there exists $\beta \overset{a}{\Longrightarrow} \beta'$ so that $(\alpha', \beta') \in \mathcal{R}$;
— for every $\beta \overset{a}{\Longrightarrow} \beta'$ there exists $\alpha \overset{a}{\Longrightarrow} \alpha'$ so that $(\alpha', \beta') \in \mathcal{R}$.

If we assume that τ does not appear in BPA Δ then the relations $\Longrightarrow$ and $\longrightarrow$ coincide and we call the corresponding version of bisimulation *strong bisimulation* and denote it by $\sim$. Processes α and β are strongly bisimilar, written $\alpha \sim \beta$, if they are related by some strong bisimulation. It was shown in [9] that the union of all strong bisimulations is also a strong bisimulation. It is the largest strong bisimulation, denoted by $\sim$, and it is an equivalence relation. We will also call it *strong bisimulation equivalence.* Moreover, strong bisimulation is a congruence on every BPA, i.e. if $\alpha \sim \beta$ and $\gamma \sim \delta$ then $\alpha\gamma \sim \beta\delta$.

Processes α and β are weakly bisimilar, written $\alpha \approx \beta$, if they are related by some weak bisimulation. The union of all weak bisimulations gives rise to the maximal weak bisimulation which is denoted by $\approx$. An equivalent definition of weak bisimulation is phrased in terms of simple transition in the premise followed by weak transition. Both definitions yield identical maximal weak bisimulations ([9]).

As opposed to strong bisimulation, a maximal weak bisimulation relation is not always necessarily a congruence — see [12] for counterexample. In order to ensure that this desirable property holds it is enough to require for a BPA (Σ^*, Δ) that for all variables $X \in \Sigma$, if $X \approx \epsilon$ then $X \overset{\tau}{\Longrightarrow} \epsilon$ (see [12]). Another trivial assumption is formulated in [7]. Here, in order to obtain congruence and simplify proofs, we will make a slightly stronger assumption throughout the rest of the paper that $P \approx \epsilon$ implies $P \equiv \epsilon$, i.e. the *only* process with no observable behavior is the empty process.

The strong, resp. weak, maximal bisimulations were obtained as the union of smaller strong, resp. weak, bisimulation relations. There exists an alternative approach (see [9]) where the maximal equivalences are obtained as the limits of respective non-increasing chains of *bisimulation approximants*. Weak bisimulation approximants $\approx_\kappa$ for a fixed BPA (Σ^*, Δ) are defined inductively on the class of ordinal numbers $\mathcal{O}n$:

— $\alpha \approx_0 \beta$ for all α and β from Σ^*;

— $\alpha \approx_{\kappa+1} \beta$ if for all actions a,

whenever $\alpha \overset{a}{\Longrightarrow} \alpha'$ then there exists $\beta \overset{a}{\Longrightarrow} \beta'$ so that $\alpha' \approx_\kappa \beta'$;

whenever $\beta \overset{a}{\Longrightarrow} \beta'$ then there exists $\alpha \overset{a}{\Longrightarrow} \alpha'$ so that $\alpha' \approx_\kappa \beta'$;

— $\alpha \approx_\lambda \beta$ if $\alpha \approx_\kappa \beta$ for every $\kappa < \lambda$, for a limit ordinal λ.

Strong bisimulation approximants $\sim_\kappa$ are defined analogously, with weak transition $\overset{a}{\Longrightarrow}$ being replaced by single transition $\xrightarrow{a}$, in both premise and conclusion.

It can be easily verified that binary relations $\approx_\alpha$ are equivalences for every ordinal α. The following proposition sums up the structure of the chain of approximants and the relationship between individual approximants and the maximal bisimulation. A proof can be found in [9, 12].

Proposition 2 *1. for every $\kappa, \mu \in \mathcal{O}n$, $\kappa < \mu \implies \approx_\mu \subseteq \approx_\kappa$;*

2. for every $\kappa \in \mathcal{O}n$, $\approx \subseteq \approx_\kappa$;

3. if there is an κ such that $\approx_\kappa = \approx_{\kappa+1}$ then for all $\mu \geq \kappa$, $\approx_\kappa = \approx_\mu = \approx$;

4. $\approx = \bigcap_{\kappa \in \mathcal{O}n} \approx_\kappa$.

An analogous lemma holds also for strong bisimulation approximants, i.e. the sequence of strong bisimulation approximants converges with the limit being $\sim$. For BPA-processes, owing to their finite-branching structure, the convergence occurs at level ω, that is $\sim = \sim_\omega = \bigcap_{n \in \omega} \sim_n$. Proof of this claim can be found in [5]. Additionally, this finite-branching property guarantees that each approximant $\sim_n$ is decidable. Therefore we obtain a straightforward semidecision procedure for non-bisimilarity by successive enumeration of all natural numbers n and testing equivalence at $\sim_n$. However, this approach cannot be used for weak bisimulation approximants because infinite branching of BPA w.r.t. weak bisimilarity produces algebras where $\approx \subsetneq \approx_\omega$ (consult e.g. [11, 12] for more details).

When we deal with the whole class of ordinals the common induction principle for natural numbers becomes too weak for proving theorems. We need a more powerful proof method than that and, fortunately, the well-ordered structure of ordinal numbers enables us to formulate a statement which is a generalization of the induction principle.

The Principle of Transfinite Induction:
Let $P(\kappa)$ *be a statement for each ordinal* κ. *Assume that*
1. $P(0)$;
2. $P(\kappa) \Rightarrow P(\kappa+1)$ *for every* κ;
3. if λ *is a limit ordinal then* $(\forall\kappa < \lambda.\ P(\kappa)) \Rightarrow P(\lambda)$.
Then for every $\kappa \in On$, $P(\kappa)$.

Now if we want to verify that some property P holds for the class *On* we only have to test three cases: the base case $P(0)$, the successor case $P(\kappa) \Rightarrow P(\kappa+1)$ and the limit case $(\forall\kappa < \lambda.\ P(\kappa)) \Rightarrow P(\lambda)$. If we manage to prove all three cases we can be confident that all ordinals possess the desired property P.

A useful way to understand both strong and weak bisimulation relation is to consider it as a *bisimulation game* between two players Alice and Bob (for detailed description see i.e. [8]). For a given LTS and its two vertices α_0 and β_0, the two players try to achieve opposite goals: Alice wants to show that α_0 and β_0 are different while Bob tries to show their sameness. A *play* of the game is a sequence of pairs $(\alpha_0, \beta_0), (\alpha_1, \beta_1), \ldots$, where each consecutive pair arises in this way: Alice chooses an action a and a transition $\alpha_i \overset{a}{\Longrightarrow} \alpha_{i+1}$, resp. $\beta_i \overset{a}{\Longrightarrow} \beta_{i+1}$. Bob then needs to produce a matching reply $\beta_i \overset{a}{\Longrightarrow} \beta_{i+1}$, resp. $\alpha_i \overset{a}{\Longrightarrow} \alpha_{i+1}$ (in the case of strong bisimulation simple transitions need to be considered). Alice wins the play if Bob cannot respond to a move by Alice, otherwise the winner is Bob. Processes α_0 and β_0 are bisimilar iff Bob is able to win every play of the game regardless of the moves made by Alice.

3 Decompositions

All known algorithms for deciding bisimilarity between two BPA-processes ([4], [7]) are strongly dependent on the notion of decomposability. Decomposition allows to transform the task of deciding bisimilarity between given pairs of processes to a (finite) number of tasks of deciding bisimilarity between smaller pairs (for some suitably formulated criterion of process size).

Let $\alpha_1\alpha_2$ and $\beta_1\beta_2$ be two BPA-processes. Let us consider one particular bisimulation play, i.e. a sequence of pairs starting with $(\alpha_1\alpha_2, \beta_1\beta_2)$. This sequence can be divided into two subsequences: the first one beginning with $(\alpha_1\alpha_2, \beta_1\beta_2)$ and the second one with the (uniquely determined) pair $(\gamma\alpha_2, \delta\beta_2)$, such that in the next step of the play an action is emitted for the first time from α_2 or from β_2. Both subsequences may be empty, finite or infinite.

The strategy for deciding bisimilarity between $\alpha_1\alpha_2$ and $\beta_1\beta_2$ based on this concept is the following. Let A be a suitably chosen set of pairs of processes. Then $\alpha_1\alpha_2 \approx \beta_1\beta_2$ if

1. for every bisimulation play starting from $\alpha_1\alpha_2$ and $\beta_1\beta_2$,

   either Bob has a winning strategy leading to his victory without emitting any action neither from α_2 nor from β_2,

   or the first pair such that in the next step an action is emitted from α_2 or β_2 has the form $(\gamma\alpha_2, \delta\beta_2)$, where $(\gamma, \delta) \in A$, and

2. $\gamma\alpha_2 \approx \delta\beta_2$ for every $(\gamma, \delta) \in A$.

The procedure sketched above can be recursively applied to newly created pairs and is efficient under the assumption that the new pairs are "simpler". In order to implement this procedure we need a generalized notion of bisimulation relation that takes into account the sets of termination pairs that occur within a bisimulation play when the first halves of the original pair are removed. That gives rise to the notion of *bisimulation up to*, originally proposed by Hirshfeld in [7].

3.1 Bisimulation up to

Definition 3 *Given an arbitrary set of pairs A, we say that a binary relation R is a* weak bisimulation up to A *if for every pair from R*
— either $(\alpha, \beta) \in A$,
— or for every action a, if $\alpha \overset{a}{\Longrightarrow} \alpha'$ then there exists $\beta \overset{a}{\Longrightarrow} \beta'$ with $(\alpha', \beta') \in R$, and symmetrically. Furthermore we require that if $\alpha \equiv \epsilon$ and $\beta \equiv \epsilon$, then $(\alpha, \beta) \in A$.

The processes α and β are *weakly bisimilar up to A*, denoted by $\alpha \approx_A \beta$, if there exists a weak bisimulation up to A that contains them. The union of all weak bisimulations up to A is a maximal weak bisimulation up to A, denoted also $\approx_A$. The relationship between "classical" bisimulation and bisimulation up to can be characterized in this way: $\approx = \approx_A$ if and only if, for every pair $(\gamma, \delta) \in A$, $\gamma \approx \epsilon$ and $\delta \approx \epsilon$.

We can follow the alternative approach towards obtaining the maximal weak bisimulation and define weak bisimulation approximants up.

Definition 4 *For a BPA (Σ^*, Δ), and a set up to A,* weak bisimulation approximants up to A *are binary relations denoted by $\approx_{\kappa,A}$, defined by*

— $\alpha \approx_{0,A} \beta$ for all α and $\beta \in \Sigma^*$,

— $\alpha \approx_{\kappa+1,A} \beta$ if $(\alpha, \beta) \in A$ or, for all actions a,

whenever $\alpha \overset{a}{\Longrightarrow} \alpha'$ then there exists $\beta \overset{a}{\Longrightarrow} \beta'$ so that $\alpha' \approx_{\kappa,A} \beta'$, and

whenever $\beta \overset{a}{\Longrightarrow} \beta'$ then there exists $\alpha \overset{a}{\Longrightarrow} \alpha'$ so that $\alpha' \approx_{\kappa,A} \beta'$;

— $\alpha \approx_{\lambda,A} \beta$ if $\alpha \approx_{\kappa,A} \beta$ for every $\kappa < \lambda$, for a limit ordinal λ.

Furthermore we require that if $\alpha \equiv \epsilon$ and $\beta \equiv \epsilon$ then $(\alpha, \beta) \in A$.

For any set up to A, the respective approximants form a non-increasing chain that approximates the maximal bisimulation up to A from above. The correctness of the two statements can be easily verified. Regarding the former, for every pair (α, β) from $\approx_{\kappa+1,A}$ there are two possibilities: either (α, β) belongs to the set A in which case it is also included in $\approx_{\kappa,A}$ by definition, or there must exist pairs of matching derivatives (α', β') that appear in $\approx_{\kappa,A}$ and, by inductive reasoning, in all the approximants below. From this we can conclude that (α, β) must belong to $\approx_{\kappa,A}$ as well. The approximant labeled by 0 contains all pairs therefore this sequence indeed forms a non-increasing chain. The correctness of the latter claim is expressed by the lemma below:

Lemma 5 $\approx_A = \bigcap_{\kappa \in \omega_1} \approx_{\kappa,A}$.

Proof: The first direction consists in proving that for any two BPA α and β, if $\alpha \approx_A \beta$ then for every ordinal κ, $\alpha \approx_{\kappa,A} \beta$. This needs to be done by transfinite induction on κ.

1. $\alpha \approx_{0,A} \beta$ is trivially true from the definition of approximants.

2. $\alpha \approx_{\kappa+1,A} \beta$ has to be proved from the premises that $\alpha \approx_A \beta$, and for any pair (α', β'), $\alpha' \approx_A \beta'$ implies that $\alpha' \approx_{\kappa,A} \beta'$. In the case that $(\alpha, \beta) \in A$ we are done as then, by definition, $\alpha \approx_{\kappa+1,A} \beta$. We assume the contrary and consider any transition $\alpha \overset{a}{\Longrightarrow} \alpha'$. As $\alpha \approx_A \beta$, and $(\alpha, \beta) \notin A$, there exists a matching move $\beta \overset{a}{\Longrightarrow} \beta'$ such that $\alpha' \approx_A \beta'$. By the other assumption, $\alpha' \approx_{\kappa,A} \beta'$ and therefore we may conclude that $\alpha \approx_{\kappa+1,A} \beta$.

3. $\alpha \approx_{\lambda,A} \beta$, for a limit ordinal λ, is a straightforward consequence of the induction hypothesis that $\alpha \approx_{\kappa,A} \beta$ for every $\kappa < \lambda$.

The other direction falls into two cases. Firstly we need to realize that the chain will converge before reaching $\approx_{\omega_1,A}$, for the simple reason that we deal with countable algebras. Convergence will occur when we reach a level such that $\approx_{\kappa,A} = \approx_{\kappa+1,A}$, in which case $\approx_A = \approx_{\kappa,A} = \approx_{\mu,A}$, for every $\kappa < \mu$. Hence we assume that for some pair (α, β), $\alpha \approx_{\kappa,A} \beta$ for every $\kappa \in \omega_1$, and we will show that $\alpha \approx_A \beta$. In case $(\alpha, \beta) \in A$ we are done as then, by definition, $\alpha \approx_A \beta$. We assume that $(\alpha, \beta) \notin A$, and consider any move $\alpha \overset{a}{\Longrightarrow} \alpha'$. For every $\kappa \in \omega_1$ there exists a matching transition $\beta \overset{a}{\Longrightarrow} \beta'_\kappa$ with $\alpha' \approx_{\kappa,A} \beta'_\kappa$. However, as β is a process in a countable algebra, there may be only countably many distinct derivatives β'_κ, and hence one β' must occur among these derivatives uncountably often. Since approximants form a non-increasing chain, this β' then satisfies the condition that $\alpha' \approx_{\kappa,A} \beta'$ for every $\kappa \in \omega_1$. So we can conclude that $\approx_{\omega_1,A}$ is in fact closed under expansion and hence included in $\approx_A$. □

3.2 Properties of bisimulation up to

The largest strong (weak) bisimulation is (under the described assumptions) an equivalence relation and both $\sim$ and $\approx$ are congruences. This is the key property which allowed to build known algorithms for deciding bisimilarity as recursive algorithms. Unfortunately bisimulation up to is no longer an equivalence relation. Namely it is the property of transitivity that fails. Nevertheless some special form of transitivity and composition holds even in this case.

Lemma 6 (Transitivity) *If $\alpha \approx_A \beta$ and $\beta \approx \beta'$, then there exists a set A' such that $\alpha \approx_{A'} \beta'$, and all pairs in A and A' are mutually bisimilar, i.e. for every $(\gamma, \delta) \in A$ there exists $(\gamma', \delta') \in A'$ with $\gamma \approx \gamma'$ and $\delta \approx \delta'$, and symmetrically for A'.*

Proof: From any R, weak bisimulation up to A, relating α and β, and any weak bisimulation S relating β and β', we will construct a set up to A' and R', a weak bisimulation up to A', that will contain the pair (α, β'). Additionally, the two sets A and A' will consist of mutually bisimilar couples, as described in the statement of the lemma. The two relations R' and A' are defined as follows:

$$\begin{aligned} A' &= \{(\gamma, \delta) \mid \exists (\gamma, \delta') \in A \ \wedge \ (\delta', \delta) \in S\} \\ R' &= \{(\gamma, \delta) \mid \exists (\gamma, \delta') \in R \ \wedge \ (\delta', \delta) \in S\} \end{aligned}$$

First we shall verify that A' satisfies the required conditions. Any pair (γ, δ) from A' has its pre-image in some pair (γ, δ') from A, where $(\delta', \delta) \in S$. Since S is a bisimulation relation, every pair contained within it must be weakly bisimilar, therefore $\delta' \approx \delta$. Obviously, $\gamma \approx \gamma$

and so we can conclude that every pair from A' has a bisimilar pre-image in A. Obviously, the other implication is also true.

It remains to check that R' is indeed a weak bisimulation up to A'. We shall express the expansion condition by means of the diagram from Figure 1.

$$\begin{tikzcd}
\delta' \arrow[dd, Rightarrow, "a"'] \arrow[rrrrrr, dotted, dash, bend left] & (\gamma,\delta') \in R & \gamma \arrow[dd, Rightarrow, "a"'] & (\gamma,\delta) \in R' & \delta \arrow[dd, Rightarrow, "a"] & (\delta',\delta) \in S & \delta' \arrow[dd, Rightarrow, "a"] \\
 & \mathrm{II} & & \mathrm{I} & & \mathrm{III} & \\
\overline{\delta'} \arrow[rrrrrr, dotted, dash, bend right] & (\overline{\gamma},\overline{\delta'}) \in R & \overline{\gamma} & (\overline{\gamma},\overline{\delta}) \overset{?}{\in} R' & \overline{\delta} & (\overline{\delta'},\overline{\delta}) \in S & \overline{\delta'}
\end{tikzcd}$$

Figure 1: Case analysis

The starting point is the pair (γ,δ) from R' at the top of square I, for which there must exist some (γ,δ') in R (top of square II) satisfying $(\delta',\delta) \in S$ (top of square III). Either (γ,δ) is contained in A' or we need to verify the expansion condition for the pair. We will assume the latter, i.e. $(\gamma,\delta) \notin A'$, from which also follows that (γ,δ') does not belong to A, and we will check the transitions.

If γ does an $\xRightarrow{a}$ and evolves into $\overline{\gamma}$, then in diagram II we have a matching move from δ' into $\overline{\delta'}$ where $(\overline{\gamma},\overline{\delta'}) \in R$. The $\xRightarrow{a}$ transition of δ' (in diagram III) evokes a matching transition of δ owing to δ' and δ being in S, with the resulting pair $(\overline{\delta'},\overline{\delta})$ also in S. Therefore, the pair of matching derivatives $(\overline{\gamma},\overline{\delta})$ belongs to R'.

If δ does $\xRightarrow{a}$ into some $\overline{\delta}$ then, in diagram III, there must be a matching transition of δ' resulting in some $\overline{\delta'}$, where $(\overline{\delta},\overline{\delta'}) \in S$. The transition $\delta' \xRightarrow{a} \overline{\delta'}$ also appears as the left-most transition in diagram II, where from the assumption that $(\gamma,\delta') \notin A$ follows that γ has a matching transition into some $\overline{\gamma}$. The pair $(\overline{\gamma},\overline{\delta'})$ is in R and $(\overline{\delta'},\overline{\delta})$ belongs to S hence, from the definition of R', we can conclude that $(\overline{\gamma},\overline{\delta})$ is included in R'. We have verified that the expansion condition holds and therefore R' is a bisimulation up to A'.

Lastly, we need to verify that if $\gamma \equiv \epsilon$ or $\delta \equiv \epsilon$ then $(\gamma,\delta) \in A'$, which readily follows from the assumptions that we have made. □

Unfortunately, it seems impossible to say anything more precise about the exact correspondence of cardinalities of A and A', because the size of the (minimal w.r.t. inclusion) set up to depends on the size of branching of the two processes that we want to relate.

Lemma 7 (Composition) *Whenever* $\alpha_1 \approx_B \beta_1$ *and* $\gamma\alpha_2 \approx_A \delta\beta_2$, *for every* $(\gamma, \delta) \in B$, *then* $\alpha_1\alpha_2 \approx_A \beta_1\beta_2$.

Proof: From the assumption that $\alpha_1 \approx_B \beta_1$ we can assume the existence of a bisimulation relation R up to B, and a set of bisimulation relations up to A for every pair (γ, δ) from B, that we denote $R_{(\gamma,\delta)}$. We define a relation $S = \{(\gamma\alpha_2, \delta\beta_2) \mid (\gamma, \delta) \in R\} \cup \bigcup_{(\gamma,\delta)\in B} R_{(\gamma,\delta)}$, and verify that this relation is a bisimulation up to A. We need to check only those pairs $(\gamma\alpha_2, \delta\beta_2)$, where $(\gamma, \delta) \in R$.

If (γ, δ) belongs directly to B then from our assumptions, $(\gamma\alpha_2, \delta\beta_2)$ belongs to $R_{(\gamma,\delta)}$ and so we are done. In the other case we need to verify the expansion condition for $(\gamma\alpha_2, \delta\beta_2)$ w.r.t. A. A schema of the proof is drawn in Figure 2. An initial move $\gamma\alpha_2 \overset{a}{\Longrightarrow}$ may lead to some $\gamma'\alpha_2$ (diagram I) or it may dispose of γ and end up in some α_2' (diagram II). In the first case we have a matching move $\delta\beta_2 \overset{a}{\Longrightarrow} \delta'\beta_2$ which belongs to S by definition.

In the latter case, if γ reduces to ϵ, the process δ will evolve into δ' such that $(\epsilon, \delta') \in B$. Then we can use the assumption that $\epsilon\alpha_2 \approx_A \delta'\beta_2$ and hence, to the transition $\alpha_2 \overset{\tau}{\Longrightarrow} \alpha_2'$ there must be an equivalent move $\beta_2 \overset{\tau}{\Longrightarrow} \beta_2'$ leading to $(\alpha_2', \beta_2') \in R_{(\epsilon,\delta')}$.

Initial moves of δ and the combination $\gamma\alpha_2 \overset{\tau}{\Longrightarrow} \epsilon\alpha_2 \overset{a}{\Longrightarrow} \alpha_2'$ would be solved analogously. The ϵ-condition on A is also easy to verify. □

Now we are ready to define the notion of decomposability we were seeking.

Definition 8 *Let* α, β *be two processes bisimilar up to* A. *We say that processes* α_1, α_2, β_1, β_2 *and a set* B *form a* decomposition of $(\alpha, \beta)A$ up to B *if*

— $\alpha \equiv \alpha_1\alpha_2$ *and* $\beta \equiv \beta_1\beta_2$,

— $\alpha_1 \approx_B \beta_1$,

— $\gamma\alpha_2 \approx_A \delta\beta_2$, *for every* $(\gamma, \delta) \in B$.

Intuitively, if we play a bisimulation game for any pair of bisimilar processes, we can always split the original processes into two pairs that will be "almost" bisimilar, up to some termination conditions. That is expressed in the following:

Fact 9 *Every pair* (α, β) *bisimilar up to* A *has some decomposition.*

$$\begin{tikzcd}
\gamma\alpha_2 \arrow[d, Rightarrow, "a"'] \arrow[rrr, dotted, bend left] \arrow[drr, phantom, "\mathrm{I}"] & & \delta\beta_2 \arrow[d, Rightarrow, "a"] & \gamma \arrow[d, Rightarrow, "a"'] & (\gamma,\delta)\in R & \delta \arrow[d, Rightarrow, "a"] \\
\gamma'\alpha_2 & (\gamma'\alpha_2,\delta'\beta_2)\in S & \delta'\beta_2 & \gamma' & (\gamma',\delta')\in R & \delta' \arrow[lll, dotted, bend left]
\end{tikzcd}$$

$$\begin{tikzcd}
\gamma\alpha_2 \arrow[d, Rightarrow, "a"'] \arrow[rrr, dotted, bend left] \arrow[drr, phantom, "\mathrm{II}"] & & \delta\beta_2 \arrow[d, Rightarrow, "a"] & \gamma \arrow[d, Rightarrow, "a"'] & (\gamma,\delta)\in R & \delta \arrow[d, Rightarrow, "a"] \\
\epsilon\alpha_2 \arrow[d, Rightarrow, "\tau"'] & & \delta'\beta_2 \arrow[d, Rightarrow, "\tau"] & \epsilon & (\epsilon,\delta')\in B & \delta' \\
\alpha_2' & (\alpha_2',\beta_2')\in R_{(\epsilon,\delta')} & \beta_2' & & &
\end{tikzcd}$$

Figure 2: Case analysis

4 Expansion trees

The notion of expansion tree is due to Hirshfeld [6] who put forward this idea in order to construct (semi)decision procedure for strong bisimulation on BPP and BPA-processes. The idea was then developed further, namely by Jančar and Moller in [8], and Hirshfeld in [7] . We first summarize this method for strong bisimulation on BPA processes.

Definition 10 *Let* $V \neq \emptyset$ *and* U *be two sets of pairs* (α, β)*.* U *is called a* strong expansion *of* V *if it is a minimal set (w.r.t. inclusion) satisfying the following property: for every pair* $(\alpha, \beta) \in V$*, for every action* a*,*
— if $\alpha \xrightarrow{a} \alpha'$ *then* $\beta \xrightarrow{a} \beta'$ *with* $(\alpha', \beta') \in U$*;*
—if $\beta \xrightarrow{a} \beta'$ *then* $\alpha \xrightarrow{a} \alpha'$*with* $(\alpha', \beta') \in U$*.*

A binary relation R is a strong bisimulation iff it is a strong expansion of itself. A nonempty set V does not have any expansion if it contains a pair (α, β) such that $\alpha \not\sim_1 \beta$, that is either α is able to emit an action β is not able to emit or vice versa. The set $V = \{(\epsilon, \epsilon)\}$ has an empty expansion $\emptyset$ as neither of processes is able to emit any action. Moreover, for finitely branching processes, if a set V is finite then every strong expansion of V is finite and the number of different expansions of V is finite, too.

The above mentioned properties give a hint how to decide bisimilarity: starting with a singleton containing the given pair expand it until a set which is an expansion of itself is achieved. This process is embodied into an expansion tree.

An *expansion tree* is a (generally infinite) tree whose nodes are labeled by sets of pairs of vertices, in which the children of a node are precisely the expansions of that node. A leaf of a tree is *successful* if it is empty; other leaves are *unsuccessful.* A branch is *successful* if it is infinite or finishes with a successful node. We may observe that the union of all nodes along a successful branch forms a bisimulation. The correctness of the expansion tree construction is spelled out in the following:

Theorem 11 *[8] $\alpha \sim \beta$ iff the expansion tree rooted at $\{(\alpha, \beta)\}$ has a successful branch.*

As we are dealing with strong bisimulation on BPA the finiteness of an expansion as well as finite branching of an expansion tree are guaranteed. However, what we need is the finite witness property which guarantees that if there are successful branches then as least one of these is finite. In such a case the breadth– first search of the expansion treee would give the decidability of bisimilarity. When dealing with strong bisimilarity on BPA it may happen that all successful branches are of infinite length. To overcome this obstacle one has to introduce modification rules into the construction of expansion trees. In their paper [8], Jančar and Moller introduce the following rules:

Rule 1 (Congruence rule) Omit from node U the pair (α, β) if it belongs to the least congruence containing $U^{\Uparrow}$, where $U^{\Uparrow}$ denotes the union of all ancestor nodes to U.

Rule 2 (Decomposition rule) If $(X\alpha, Y\beta)$ is in U where X and Y are normed, then create a new sibling node $U' = U \setminus \{(X\alpha, Y\beta)\} \cup \{(X, Y\gamma), (\gamma\alpha, \beta)\}$, where $|X|$=$|Y\gamma|$ (and symmetrically).

Rule 3 (Replacement rule) If $(X\alpha, Y\beta)$ is in U and some $(X\alpha', Y\beta')$ is in $U^{\Uparrow}$, then create a new sibling node $U' = U \setminus \{(X\alpha, Y\beta)\} \cup \{(\alpha, \alpha'), (\beta, \beta')\}$.

Obviously, when these rules are applied to the construction we need to verify that correctness is preserved. That boils down to checking that no *false* bisimulation witness is created, i.e. no pairs that would imply bisimilarity of two originally non-bisimilar processes are added. This is guaranteed by the following correctness criterion for (modified) expansion trees.

Lemma 12 *[8] For any node* $V \neq \emptyset$ *and for any* $n \in \mathbb{N}$, $V \subseteq \sim_{n+1}$ *iff* V *has a child* $U \subseteq \sim_n$. *As a consequence,* $V \subseteq \sim$ *iff* V *has a child* $U \subseteq \sim$.

Rule 1 ensures that a pair is not considered if it can be composed from pairs that occurred previously (we use the fact that bisimilarity is a congruence). Rule 2 allows us to replace pairs by their decompositions which are strictly smaller in size (here the strong norm is taken as size criterion). However, one can easily find a pair which is not decomposable in the sense of Rule 2. Then Rule 3 will eventually be applied. Efficiency of the modification rules is asserted by a theorem from [4] that states that the number of undecomposable pairs is in some sense finite and therefore the modified expansion tree with bisimilar pair in its root always contains a finite successful branch. Hence the strong bisimilarity on BPA is semidecidable.

When dealing with weak bisimulation, we have to consider *weak expansions* and *weak expansion trees* that are obtained by replacing single transitions by composite ones. As in the case of the strong bisimulation, in order to cope with infiniteness we will introduce some modification rules that will employ decomposition and bisimulation up to. To this end, we need to define a generalized notion of expansion tree, *expansion tree up to.*

Definition 13 *Let* $V \neq \emptyset$ *and* U *be two sets of elements* $(\alpha, \beta)A$. U *is called a* weak expansion up to *of* V *if it is a minimal set (w.r.t. inclusion) satisfying the following property: for every pair* $(\alpha, \beta)A \in V$,
— *either* $(\alpha, \beta) \in A$,
— *or, for every action* a,

if $\alpha \overset{a}{\Longrightarrow} \alpha'$ *then* $\beta \overset{a}{\Longrightarrow} \beta'$ *with* $(\alpha', \beta')A \in U$*;*

if $\beta \overset{a}{\Longrightarrow} \beta'$ *then* $\alpha \overset{a}{\Longrightarrow} \alpha'$*with* $(\alpha', \beta')A \in U$.

The notion of *successful leaf*, resp. *successful branch*, generalizes to expansion trees up to in the obvious sense (in particular, an unsuccessful leaf contains an element $(\alpha, \beta)A$ with $\alpha \not\approx_{1,A} \beta$). We are proposing the following generalization of the modification rules of [8], in the spirit of [7], for weak bisimulation and general BPA.

Rule 4 (Omitting rule) Omit $(\alpha, \beta)A$ from a node U if any of the following occurs:

1. $(\alpha, \beta)A$ appears in $U^{\Uparrow}$;

2. (α, β) belongs to A;

3. $\alpha \equiv \beta$ and $(\epsilon, \epsilon) \in A$;

4. $\alpha \equiv \beta$, and they are unnormed processes.

Rule 5 (Decomposition rule) If $(X\alpha, Y\beta)A$ belongs to U, then construct a sibling node U' by replacing $(X\alpha, Y\beta)A$ by the set $\{(X, Y)B\} \cup \{(\gamma\alpha, \delta\beta)A \mid (\gamma, \delta) \in B\}$, where B is a new set up to.

Rule 6 (Replacement rule) If $(X\alpha, Y\beta)A$ is in U and some $(X\alpha', Y\beta')A'$ is in $U^{\Uparrow}$, then create a sibling node U' by replacing $(X\alpha, Y\beta)A$ with the set $\{(\alpha, \alpha')(\epsilon, \epsilon), (\beta, \beta')(\epsilon, \epsilon)\} \cup \{(\gamma, f(\gamma))(\epsilon, \epsilon), (\delta, f(\delta)(\epsilon, \epsilon) \mid (\gamma, \delta) \in A\} \cup \{(g(\gamma), \gamma)(\epsilon, \epsilon), (g(\delta), \delta)(\epsilon, \epsilon) \mid (\gamma, \delta) \in A'\}$, where $f : A \longrightarrow A'$, and $g : A' \longrightarrow A$ are arbitrary functions.

Rule 4 describes pairs whose presence in the tree is superfluous. Rule 5 is an analog of Decomposition Rule 2, and Rule 6 is a weak bisimulation analog of Replacement Rule 3. Obviously, as well as with strong bisimulation expansion trees, we need to check that the correctness of the construction has not been affected, in particular that no false witness can be added in this way. The correctness criterion needs to reflect the fact that for weak bisimulation approximants, convergence (to the maximal relation) may occur at any ordinal less than ω_1. Furthermore, we are dealing with pairs bisimilar up to. Both facts are taken into account in the criterion below.

Proposition 14 *For any node $V \neq \emptyset$ and for any $\mu < \omega_1$, there exists$(\alpha, \beta)A$ in V such that $(\alpha, \beta) \not\in\approx_{\mu,A}$ iff for every child U, there exist $\kappa < \mu$ and $(\alpha', \beta')A'$ in U such that $(\alpha', \beta') \not\in\approx_{\kappa,A'}$.*

As a consequence of Proposition 14 together with the convergence criterion ($\approx_A = \bigcap_{\kappa\in\omega_1} \approx_{\kappa,A}$) we arrive at Proposition 15:

Proposition 15 *For any node $V \neq \emptyset$, $\{(\alpha, \beta) \mid (\alpha, \beta)A \in V\} \subseteq \approx_A$, for every A, iff there exists a child U with $\{(\alpha, \beta) \mid (\alpha, \beta)A \in U\} \subseteq \approx_A$, for every A.*

Clearly, if we start with a tree rooted at $(\alpha, \beta)(\epsilon, \epsilon)$ for a bisimilar pair then the root satisfies condition of Proposition 15 and so it has a child also satisfying the condition, and so on. The sequence of such nodes forms a successful branch, finite or infinite. On the other hand, if the initial pair is not equivalent at some $\approx_\mu$, then every branch determines a sequence of inequivalent elements $(\alpha, \beta) \not\in\approx_{\kappa,A_\kappa}$ where κ is decreasing. Since every decreasing sequence of ordinals is finite every branch will eventually reach a node containing some $(\alpha, \beta) \not\in\approx_{1,A_1}$ which denotes failure. This argument is reflected in the theorem that follows.

Theorem 16 *If a (modified) expansion tree* T *rooted at* $(\alpha, \beta)(\epsilon, \epsilon)$ *satisfies Proposition 14, then* $\alpha \approx \beta$ *iff there exists a successful branch in* T.

This theorem states that every rule respecting Proposition 14 maintains safeness. The next step is therefore to prove that Rules 4, 5 and 6 satisfy Proposition 14. The way of doing so is to assume that, given some node and its successors satisfying the condition of Proposition 14, any new child arising by application of the rules will also respect it.

Rule 4 specifies when checking a pair $(\alpha, \beta)A$ would be superfluous, either because it has been considered previously (case 1), or its bisimilarity up to A can be proved by some simple argument (cases 2, 3, 4). The correctness of Rule 5 comes as a consequence of the following lemma:

Lemma 17 *If* $(\alpha_1, \beta_1) \in \approx_{\kappa,B}$ *and* $(\gamma\alpha_2, \delta\beta_2) \in \approx_{\kappa,A}$ *for every* $(\gamma, \delta) \in B$*, then* $(\alpha_1\alpha_2, \beta_1\beta_2) \in \approx_{\kappa,A}$.

Proof: Would be formally done by transfinite induction. First we need to consider the case when $\kappa = 0$; that holds trivially as by definition, all pairs are equivalent at $\approx_0$.
The successor case ($P(\kappa) \Rightarrow P(\kappa + 1)$) is spelt out as follows:

$$\underbrace{[(\alpha_1, \beta_1) \in \approx_{\kappa,B} \wedge \forall(\gamma, \delta) \in B.(\gamma\alpha_2, \delta\beta_2) \in \approx_{\kappa,A} \Rightarrow (\alpha_1\alpha_2, \beta_1\beta_2) \in \approx_{\kappa,A}]}_{P(\kappa)} \Longrightarrow$$

$$\underbrace{[(\alpha_1, \beta_1) \in \approx_{\kappa+1,B} \wedge \forall(\gamma, \delta) \in B.(\gamma\alpha_2, \delta\beta_2) \in \approx_{\kappa+1,A} \Rightarrow (\alpha_1\alpha_2, \beta_1\beta_2) \in \approx_{\kappa+1,A}]}_{P(\kappa+1)}$$

The goal therefore is to prove that, from the induction hypothesis $P(\kappa)$ and assumptions $(\alpha_1, \beta_1) \in \approx_{\kappa+1,B}$ and $(\gamma\alpha_2, \delta\beta_2) \in \approx_{\kappa+1,A}$, for every $(\gamma, \delta) \in B$, we can conclude that $(\alpha_1\alpha_2, \beta_1\beta_2) \in \approx_{\kappa+1,A}$. We will again make use of graphical description of the situation (Fig. 3). We assume an initial move of $\alpha_1\alpha_2$ which may either end up in some $\gamma\alpha_2$ (see diagram I), or lead to some α_2' with α_1 removed along the way (diagram II). In the first case we have a matching equivalent move of $\beta_1 \overset{a}{\Longrightarrow} \beta_1'$ with $\alpha_1' \approx_{\kappa,B} \beta_1'$. By applying induction hypothesis to the pair (α_1', β_1') we obtain that $(\alpha_1'\alpha_2, \beta_1', \beta_2) \in \approx_{\kappa,A}$, which then validates the desired claim $\alpha_1\alpha_2 \approx_{\kappa+1,A} \beta_1\beta_2$.

In the second case we need to use the fact that if we reach ϵ from α_1, a matching equivalent move of β_1 leads to some δ where $(\epsilon, \delta) \in B$. Then we can use the induction hypothesis to conclude that $(\alpha_2, \delta\beta_2) \in \approx_{\kappa+1,A}$ from which the equivalence of $\alpha_1\alpha_2$ and $\beta_1\beta_2$ at $\approx_{\kappa+1,A}$ follows. Moves initiating in β_1 and the combination $\alpha_1\alpha_2 \overset{\tau}{\Longrightarrow} \epsilon\alpha_2 \overset{a}{\Longrightarrow} \alpha_2'$ would be solved analogously.

$$\begin{tikzcd}
\alpha_1\alpha_2 \arrow[dd, Rightarrow, "a"'] \arrow[rrr, dotted, bend left] & \overset{?}{\approx}_{\kappa+1,A} & \beta_1\beta_2 \arrow[dd, Rightarrow, "a"] & \alpha_1 \arrow[dd, Rightarrow, "a"'] & \approx_{\kappa+1,B} & \beta_1 \arrow[dd, Rightarrow, "a"] \\
 & \mathrm{I} & & & & \\
\alpha_1'\alpha_2 & \approx_{\kappa,A} & \beta_1'\beta_2 & \alpha_1' & \approx_{\kappa,B} \arrow[lll, dotted, bend left] & \beta_1'
\end{tikzcd}$$

$$\begin{tikzcd}
\alpha_1\alpha_2 \arrow[dd, Rightarrow, "a"'] \arrow[rrr, dotted, bend left] & \overset{?}{\approx}_{\kappa+1,A} & \beta_1\beta_2 \arrow[dd, Rightarrow, "a"] & \alpha_1 \arrow[dd, Rightarrow, "a"'] & \approx_{\kappa+1,B} & \beta_1 \arrow[dd, Rightarrow, "a"] \\
 & \mathrm{II} & & & & \\
\epsilon\alpha_2 \arrow[d, Rightarrow, "\tau"'] & \approx_{\kappa+1,A} & \delta\beta_2 \arrow[d, Rightarrow, "\tau"] & \epsilon & \approx_{\kappa,B} & \delta \\
\alpha_2' & \approx_{\kappa,A} & \beta_2' & & &
\end{tikzcd}$$

Figure 3: Case analysis

The limit case would consist in proving that $\forall\kappa < \lambda.P(\kappa) \Rightarrow P(\lambda)$, and it would proceed analogously to the successor case. The ϵ-condition on $\approx_{\kappa,A}$ is straightforward to verify. □

As a consequence of the previous lemma we obtain that if there is a node V containing some $(\alpha, \beta) \notin \approx_{\mu,A}$, then there must be some $(\alpha', \beta') \notin \approx_{\kappa,A'}$ in a new successor node U', for some $\kappa < \mu$.

In order to prove safeness of Rule 6 we need to build a sequence of auxiliary results concerning restricted transitivity for approximants up to. In order to make our notation more concise we shall write $A \approx A'$ whenever for every $(\gamma, \delta) \in A$ there exists $(\gamma', \delta') \in A'$ with $\gamma \approx \gamma'$ and $\delta \approx \delta'$, and symmetrically for A'.

Lemma 18 *If $\alpha \approx_{\kappa,A} \beta$ and $\beta \approx \beta'$, then there exists a set A' such that $\alpha \approx_{\kappa,A'} \beta'$ and $A \approx A'$.*

Proof: The flavor of the proof is similar to the analogous lemma for bisimulation, however here we need to employ the principle of transfinite induction. For a fixed α, β, $\beta\,'$, and A, the set A' is defined using A and S, some fixed weak bisimulation relating β and β':

$$A' = \{(\gamma, \delta) \mid \exists(\gamma, \delta') \in A \ \land \ (\delta', \delta) \in S\}$$

As in the proof of Lemma 7, it is not difficult to verify that indeed, $A \approx A'$, hence it remains to test the expansion condition. The case for $\kappa = 0$ is clear, and so we continue with the successor step. The induction hypothesis $P(\kappa)$ is the statement $\alpha \approx_{\kappa,A} \beta \land \beta \approx \beta' \Rightarrow \alpha \approx_{\kappa,A'} \beta'$. We are going to assume that $\alpha \approx_{\kappa+1,A} \beta$ and $\beta \approx \beta'$ and prove that $\alpha \approx_{\kappa+1,A'} \beta'$.

$$\begin{tikzcd}
\beta \arrow[r, phantom, "\approx_{\kappa+1,A}"] \arrow[d, Rightarrow, "a"'] \arrow[dr, phantom, "\mathrm{II}"] \arrow[rrr, dotted, bend left, no head] & \alpha \arrow[r, phantom, "\overset{?}{\approx}_{\kappa+1,A'}"] \arrow[d, Rightarrow, "a"'] \arrow[dr, phantom, "\mathrm{I}"] & \beta' \arrow[r, phantom, "(\beta, \beta') \in S"] \arrow[d, Rightarrow, "a"] \arrow[dr, phantom, "\mathrm{III}"] & \beta \arrow[d, Rightarrow, "a"] \\
\overline{\beta} \arrow[r, phantom, "\approx_{\kappa,A}"] \arrow[rrr, dotted, bend right, no head] & \overline{\alpha} \arrow[r, phantom, "\approx_{\kappa,A'}"] & \overline{\beta'} \arrow[r, phantom, "(\overline{\beta}, \overline{\beta'}) \in S"] & \overline{\beta}
\end{tikzcd}$$

Figure 4: Case analysis

From the definition of A' we can conclude that $(\alpha, \beta) \in A$ if and only if, $(\alpha, \beta') \in A'$. Therefore we can assume that if there is a move $\alpha \overset{a}{\Longrightarrow} \overline{\alpha}$, then there exists $\beta \overset{a}{\Longrightarrow} \overline{\beta}$, where $\overline{\alpha} \approx_{\kappa,A} \overline{\beta}$ (Figure 4, square II). Then we have a matching bisimilar transition $\beta' \overset{a}{\Longrightarrow} \overline{\beta}'$ (square III). Now we can apply the induction hypothesis on the pairs $(\overline{\alpha}, \overline{\beta}) \in \approx_{\kappa,A}$ and $\overline{\beta} \approx \overline{\beta}'$. Here we need to realize that $(\overline{\beta}, \overline{\beta}')$ is a different pair than the original (β, β'), however as the former is a derivative of the latter, we may use the bisimulation S to define the new set up to and thus we obtain the same set A' with $\overline{\alpha} \approx_{\kappa,A'} \overline{\beta}'$. Therefore we may conclude that indeed, $\alpha \approx_{\kappa+1,A'} \beta'$.

If we start from a transition $\beta' \overset{a}{\Longrightarrow} \overline{\beta}'$, we make use of a matching bisimilar move $\beta \overset{a}{\Longrightarrow} \overline{\beta}$ (diagram III). Then in square II we have a move $\alpha \overset{a}{\Longrightarrow} \overline{\alpha}$ with $\overline{\alpha} \approx_{\kappa,A} \overline{\beta}$, and using an analogous argument, we can conclude that $\alpha \approx_{\kappa+1,A'} \beta'$.

For a limit ordinal λ, the proof relies on the fact that, $\alpha \approx_{\lambda,A} \beta$ if and only if, $\alpha \approx_{\kappa,A} \beta$, for every ordinal $\kappa < \lambda$. The argument is analogous to the successor case. □

Corollary 19 *If $X\alpha \approx_{\kappa,A} Y\beta$, $\alpha \approx \alpha'$ and $\beta \approx \beta'$, then there exists a set A' such that $X\alpha' \approx_{\kappa,A'} Y\beta'$ and $A \approx A'$.*

However, note that in order to obtain the corollary we need to employ a symmetric variant of Lemma 18 where we substitute a bisimilar pair on the left hand side. The reason for that is that in general, approximants up to (and also bisimulation up to) are not symmetric relations.

As a consequence of the previous lemma we obtain that if there is a node V containing some $(\alpha, \beta) \notin \approx_{\mu,A}$, then there must be some $(\alpha', \beta') \notin \approx_{\kappa,A'}$ in a new successor node U', for some $\kappa < \mu$.

5 Applications

In the previous section we have sketched the way of building up the weak expansion tree for a given pair of processes. Now we shall discuss applicability of this approach to deciding weak bisimilarity.

Necessary conditions for a (modified) expansion tree to be an algorithm are:

1. the tree is *finitely* branching
2. every vertex is labeled by a *finite* set
3. if the root is labeled by a (weakly) bisimilar pair then the tree has a *finite* successful branch.

The first condition is not valid as a finite set can have infinite number of different weak expansions due to the composite transitions. Nevertheless its invalidity is not critical. Searching the tree by dove-tailing technique results in semidecision procedure (if there is a finite successful branch than it is found otherwise the search never halts).

The second condition also need not be true. There are two sources of infinity. Firstly, while expanding a finite set we can come to an infinite one owing to composite transition on the attacker side. Secondly, infiniteness can arise while decomposing a pair of processes, namely in the set up to. A simple example is:

$$\begin{array}{llll} X \xrightarrow{a} \epsilon & Y \xrightarrow{a} \epsilon & B \xrightarrow{b} \epsilon & U \xrightarrow{b} U \\ X \xrightarrow{a} XB & Y \xrightarrow{a} Y & & \end{array}$$

Although $XU \sim YU$ we cannot decompose the pair as $X \not\sim Y$, moreover, at a closer look we find out that any set A with the property that X is bisimilar to Y up to A is infinite and must contain $\{(B^i, \epsilon) \mid i \in \mathbb{N}\}$. However, (finiteness of) the decision procedure is based on the fact that any two nonbisimilar variables have only finitely many nonbisimilar completions.

One can avoid these problems by considering the variant of weak bisimulation in which attacker is allowed to do only simple transitions (for the first type of infinity) or by allowing compact finite representation of the sets up to (i.e. via a finite or pushdown automaton). But there is still the third condition we have to cope with. As the next example shows this obstacle is the most serious one.

The following BPA represents an algebra where violation of condition 3 appears, i.e. there exists a bisimilar pair for which the modified expansion tree has no finite successful branch.

$$\begin{array}{llll} X \xrightarrow{\tau} ZY & X \xrightarrow{a} XW & Z \xrightarrow{\tau} ZW & Y \xrightarrow{c} Y \\ X \xrightarrow{\tau} \epsilon & & Z \xrightarrow{\tau} \epsilon & W \xrightarrow{b} \epsilon \end{array}$$

All relevant (in)equivalence relationships are summarized below:

1. $Y \approx Y\alpha$, for any process α;

2. $XW^iY \approx XW^jY$, for every i, j;

3. $XW^i \not\approx XW^j$, for every $i \neq j$;

4. $W^iY \not\approx W^jY$, for every $i \neq j$.

As Y is an unnormed variable, the first equivalence is easy to observe. To verify item 3., we assume that $i < j$, and observe that after XW^i disposes of X it will do exactly b^i to reach ϵ, however XW^j can in any case do at least b^{i+1} as $i < j$. Similarly in case 4., if $i < j$ then W^iY can do c after b^i which cannot be matched by W^jY. In order to test equivalence 2. we first analyze all possible (composite) moves of X. They are

$$\begin{array}{lll} X \overset{\tau}{\Longrightarrow} \epsilon & X \overset{a}{\Longrightarrow} XW^{i+1}Y & X \overset{b}{\Longrightarrow} W^kY \\ X \overset{\tau}{\Longrightarrow} ZW^kY & X \overset{a}{\Longrightarrow} ZW^kY & X \overset{c}{\Longrightarrow} Y \\ X \overset{\tau}{\Longrightarrow} W^kY & X \overset{a}{\Longrightarrow} W^kY & \end{array}$$

Firstly, XW^iY may dispose of the X in front, then the other process XW^jY evolves into W^iYW^jY by means of the sequence $XW^jY \xrightarrow{\tau} ZYW^jY \xrightarrow{\tau^i} ZW^iYW^jY \xrightarrow{\tau} W^iYW^jY$, which is equivalent to W^iY by equivalence 1. The other interesting move is $XW^iY \xrightarrow{a} XW^{i+1}Y$ that is matched by $XW^jY \xrightarrow{a} XW^{j+1}Y$. The remaining possibilities consist in X generating ZW^kY or W^kY to which the other side responds by creating an exact copy (hence we obtain two bisimilar processes $Z^eW^kYW^iY$ and $Z^eW^kYW^jY$, where $e \in \{0, 1\}$).

Before we present the construction of a weak expansion tree we will make some observations about decomposability of bisimilar pairs in this algebra. From 3. and 4. above follows that for distinct i and j the pair (XW^iY, XW^jY) has no classical decomposition, i.e. there is *no* way of splitting XW^iY and XW^jY into two pairs of bisimilar processes. Furthermore, every bisimulation relating the pair is infinite and has no finite base as it must contain the set $\{(XW^{i+k}Y, XW^{j+k}Y \mid k \in \mathbb{N}\}$, which is not finitely generated.

At a closer look we may note that in any bisimulation play leading from the pair (XW^iY, XW^jY), X on either side may evolve into an unnormed process by choosing to perform $X \xrightarrow{\tau} ZY$, or it may disappear by doing $X \xrightarrow{\tau} \epsilon$. To the latter move the only correct response (of the other X) is $X \overset{\tau}{\Longrightarrow} W^kY$, where k depends on i or j and the current depth of the play. Hence we may conclude that in general, $X \approx_A X$ for any set A containing $(\epsilon, W^lY), (W^mY, \epsilon)$, where $l, m \in \mathbb{N}$.

Figure 5 represents a sketch of a construction of weak expansion tree for the pair (XWY, XW^2Y), that only contains correct expansions and correct applications of modification rules. We will make use of equivalence 1. above and only consider those processes that contain at most one Y, as the final variable. We make the following conventions: in order to save space the set up to $\{(\epsilon, \epsilon)\}$ is denoted by ε; pairs that are not underlined are those omitted in further construction by application of Rule 6. We either omit identical pairs if the set up to contains (ϵ, ϵ) (such as $(W^iY, W^iY)\varepsilon$), or identical pairs of unnormed processes. The other application of omitting rule is whenever a pair belongs to the respective set up to (e.g. $(\epsilon, W^2Y)A$, where $A = \{(\epsilon, W^2Y), (W^3Y, \epsilon)\}$). The original root is labeled by $\{(XWY, XW^2Y)\varepsilon\}$, however a new root labeled by $\emptyset$ is added as a result of application of Rule 5 to the original one. The rightmost branch actually after a few steps becomes identical to the branch on the left which is denoted by an arrow in the picture.

The correct choices of sets up to when applying Rule 5 are influenced by the only correct response to the transition $X \xrightarrow{\tau} \epsilon$. When we decompose the original pair (XWY, XW^2Y), the only correct set up to is $B = \{(\epsilon, WY), (W^2Y, \epsilon)\}$ as $X \approx_B X$ and also $WY \approx WY$, and $W^2Y \approx W^2Y$. When we move to (XW^2Y, XW^3Y) we need to consider $B' = A = \{(\epsilon, W^2Y), (W^3Y, \epsilon)\}$. Then, as X keeps generating further copies of W, also the exponents of W in the consecutive sets up to grow. The sets are finite but unbounded in size of its elements. As the sets are all distinct (w.r.t. weak bisimilarity), any infinite branch cannot be terminated as a successful finite branch by the presented rules.

6 Conclusions

In this paper we have attempted to generalize the method of expansion trees for semideciding weak bisimilarity of BPA-processes. The main idea was to split a given problem (of deciding whether a given pair is weakly bisimilar) to a number a smaller tasks of the same type which would lead to a recursive procedure. In the Application section we have demonstrated an example of BPA-processes where even after application of the modifica-

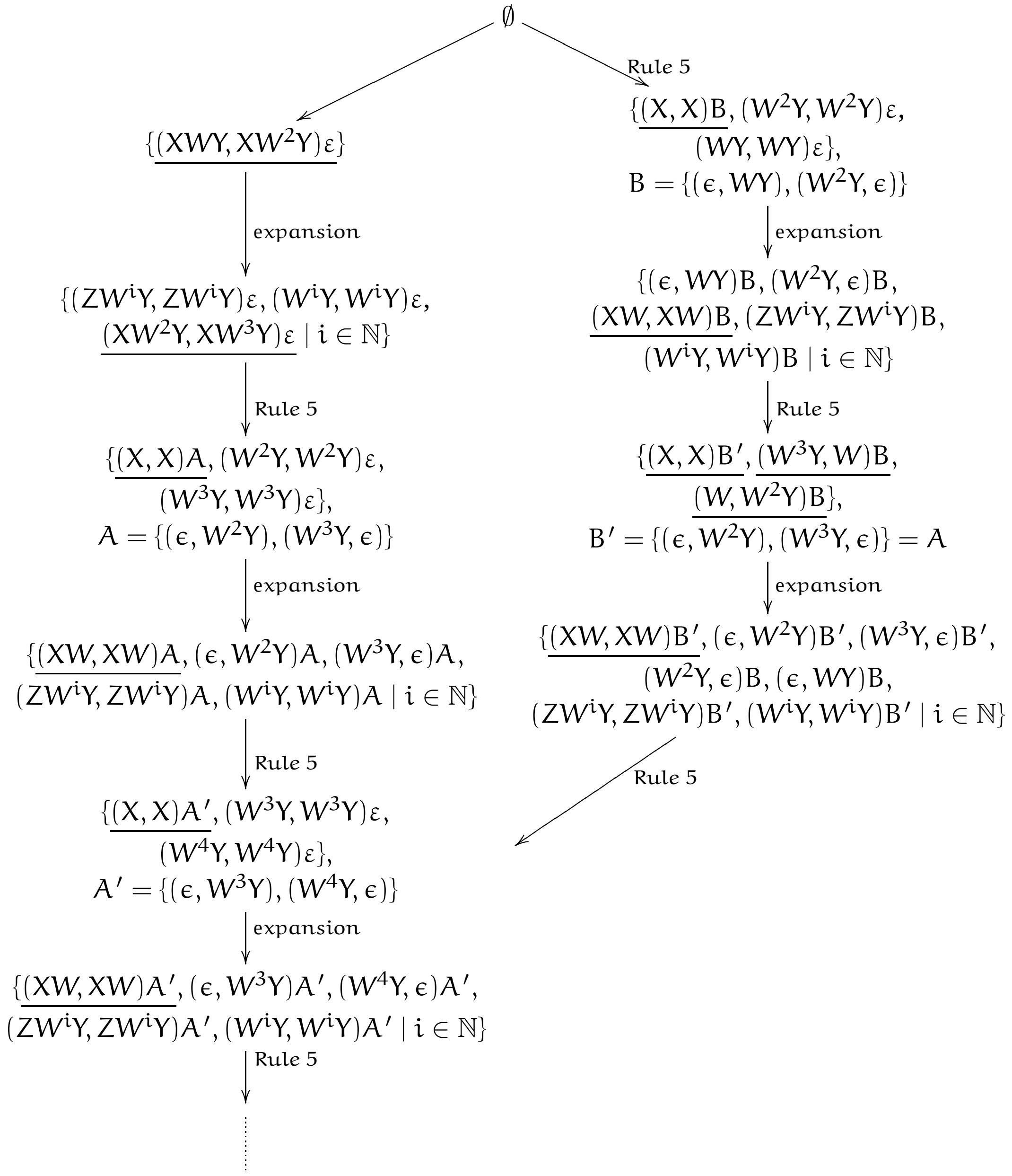

$\emptyset$
Rule 5
$\{\underline{(XWY, XW^2Y)\varepsilon}\}$
expansion
$\{(ZW^iY, ZW^iY)\varepsilon, (W^iY, W^iY)\varepsilon,$
$\underline{(XW^2Y, XW^3Y)\varepsilon} \mid i \in \mathbb{N}\}$
Rule 5
$\{\underline{(X, X)A}, (W^2Y, W^2Y)\varepsilon,$
$(W^3Y, W^3Y)\varepsilon\},$
$A = \{(\epsilon, W^2Y), (W^3Y, \epsilon)\}$
expansion
$\{\underline{(XW, XW)A}, (\epsilon, W^2Y)A, (W^3Y, \epsilon)A,$
$(ZW^iY, ZW^iY)A, (W^iY, W^iY)A \mid i \in \mathbb{N}\}$
Rule 5
$\{\underline{(X, X)A'}, (W^3Y, W^3Y)\varepsilon,$
$(W^4Y, W^4Y)\varepsilon\},$
$A' = \{(\epsilon, W^3Y), (W^4Y, \epsilon)\}$
expansion
$\{\underline{(XW, XW)A'}, (\epsilon, W^3Y)A', (W^4Y, \epsilon)A',$
$(ZW^iY, ZW^iY)A', (W^iY, W^iY)A' \mid i \in \mathbb{N}\}$
Rule 5
$\{\underline{(X, X)B}, (W^2Y, W^2Y)\varepsilon,$
$(WY, WY)\varepsilon\},$
$B = \{(\epsilon, WY), (W^2Y, \epsilon)\}$
expansion
$\{(\epsilon, WY)B, (W^2Y, \epsilon)B,$
$\underline{(XW, XW)B}, (ZW^iY, ZW^iY)B,$
$(W^iY, W^iY)B \mid i \in \mathbb{N}\}$
Rule 5
$\{\underline{(X, X)B'}, \underline{(W^3Y, W)B},$
$\underline{(W, W^2Y)B}\},$
$B' = \{(\epsilon, W^2Y), (W^3Y, \epsilon)\} = A$
expansion
$\{\underline{(XW, XW)B'}, (\epsilon, W^2Y)B', (W^3Y, \epsilon)B',$
$(W^2Y, \epsilon)B, (\epsilon, WY)B,$
$(ZW^iY, ZW^iY)B', (W^iY, W^iY)B' \mid i \in \mathbb{N}\}$
Rule 5

Figure 5: Modified expansion tree for (XWY, XW^2Y)

tion rules suggested in this paper we obtain larger and larger processes which results in non-termination of the proposed procedure.

The example presented in the previous section is an example of a process algebra where the maximal weak bisimulation does not have a finite Caucal base, moreover every weak bisimulation relating e.g. the pair (XW^2Y, XW^3Y) also fails to have a finite base. However, we are able to provide a finite description of a Caucal base of any such bisimulation (for instance by means of a pushdown automaton). In general, any recursive description of a Caucal base suffices to semidecide weak bisimilarity. The existence of a recursive Caucal base of the maximal weak bisimulation and its efficient construction remain open questions. This would be one possible way of attacking the (semi)decidability problem for weak bisimilarity on BPA.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings of PARLE'87*, volume 259 of *Lecture Notees in Computer Science*. Springer, 1987.

[2] O. Burkart, D. Caucal, and B. Steffen. An elementary decision procedure for arbitrary context-free processes. In *Proceedings of MFCS'95*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1995.

[3] D. Caucal. Graphes canoniques de graphes algébriques. In *Informatique théorique et Applications*, volume 24(4), pages 339–352. RAIRO, 1990.

[4] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *Proceedings of CONCUR'92*, volume 630 of *Lecture Noter in Computer Science*, pages 138–147. Springer, 1992.

[5] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, (32):137–161, 1985.

[6] Y. Hirshfeld. Deciding equivalences in simple Process Algebras. Technical report, LFCS Report Series, University of Edinburgh, 1994.

[7] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulation. In Steffen B. and Caucal D., editors, *Proceedings of INFINITY'96*, volume 5 of *ENTCS*, 1996.

[8] P. Jančar and F. Moller. Techniques for decidability and undecidability of bisimilarity. In *Proceedings of CONCUR'99*, volume 1664 of *Lecture Noter in Computer Science*. Springer, 1999.

[9] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[10] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings of GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[11] J. Stříbrná. Approximanting weak bisimulation on Basic Process Algebra. In *Proceedings of MFCS'99*, volume 1672 of *Lecture Notes in Computer Science*, pages 336–375. Springer, 1999.

[12] J. Stříbrná. *Decidability and complexity of equivalences for simple process algebras*. Ph.D. Thesis, University of Edinburgh, 1999.

Rewriting in the partial algebra of typed terms modulo ACI

Thomas Colcombet

Thomas.Colcombet@irisa.fr

Abstract

We study the partial algebra of typed terms with an associative commutative and idempotent operator (typed ACI-terms). The originality lies in the representation of the typing policy by a graph which has a decidable monadic theory.

In this paper we show on two examples that some results on ACI-terms can be simply raised to the level of typed ACI-terms. The examples are the results on rational subsets (closure by complement, decidability of the emptyness) and the property reachability problem for ground rewrite systems (equivalently process rewrite systems).

1 Introduction

Exact verification of programs is known for long to be undecidable. To bypass this limit, a solution is to abstract programs into weaker formal models on which decision procedures are possible.

The pushdown processes have words as states and the transitions are defined by a finite set of suffix (or prefix but not both) rewriting rules. Pushdown processes model faithfully the control flow and the stack mechanism of programs. Highly complex properties can be automatically checked on such systems (e.g. monadic second order theory [17] and model checking of the μ-calculus). However those systems lack parallelism features. The Petri nets are defined as systems of vectors addition. They contain subtle parallelism facilities but lack expressiveness for control flow. The difficult problem of reachability is decidable for Petri nets [14, 11].

The two models have been combined into process rewrite systems [16, 15] which have a decidable property reachability problem. Those process rewrite systems can be seen as ground rewrite systems of terms with an associative-commutative symbol.

Since then, pushdown systems were extended to higher order pushdown systems (processes with stacks of stacks of stacks . . .), leading to a hierarchy of systems (or graphs) having a decidable monadic second order theory [3]. It is thus natural to try to combine those higher order systems with Petri nets and obtain «higher order process rewrite systems.»

We believe that the correct way for this integration is the use of types. The idea originates from [4] where graphs solution of infinite equational systems were shown to have a decidable first order theory with reachability if the equational system (as a graph) has a decidable monadic theory. If the equational system is finite, then those graphs exactly coincide with ground rewrite systems of typed terms with a finite typing policy (the result was already known for this class of systems [7]). In fact the proof remains valid for infinite equational systems, but there is an infinite number of types.

In this paper, we follow this direction and replace terms by terms with an associative-commutative symbol. Those terms are restricted by a typing policy which «has a decidable monadic theory». We show that decidability results on process rewrite systems can be raised to the level of «typed process rewrite systems».

The remaining of the paper is divided as follows. In Section 2 we study the untyped ACI terms. In Section 3 the corresponding study is performed in the typed case. In Section 4 we study the typed process rewrite systems.

2 Untyped ACI terms

2.1 Terms with an ACI symbol

In this section, we introduce typed terms with an associative commutative symbol.

From now on, A stands for a finite set of *symbols*. We consider the algebra $\mathcal{T}(A)$ of (finite) terms built with the constant 0, the unary symbols $a \in A$ and the binary operator $+$:

$$t ::= 0 \mid a(t) \mid t + t \,.$$

The terms are considered modulo associativity and commutativity of the $+$ operator and idempotency of $+$ for 0:

$$t + t' = t' + t, \quad t + (t' + t'') = (t + t') + t'' \quad \text{and} \quad t = 0 + t \,.$$

For any unary symbol $a \in A$ and any term $t \in \mathcal{T}(A)$, we write a instead of $a(0)$ and call such terms *constants*. We say that a term t is *rooted* if it has a symbol of A at its root (and not 0 or $+$). We will also use the notation $\Sigma_{i=1}^{n} t_i$ to denote $t_1 + \ldots + t_n$ and the notation nt for $\Sigma_{i=1}^{n} t$. With this notations, it is natural to decompose a term t as $\Sigma_{i=1}^{n} t_i$ where all the t_i are rooted. This decomposition is at the core of most inductive proofs presented in this paper.

The *branches* of a term t are the words in $br(t)$ defined inductively by $br(0) = \{\epsilon\}$, $br(t + t') = br(t) \cup br(t')$ and $br(a(t)) = \{aw \mid w \in br(t)\}$. One can notice that the branches are by definition prefix closed. The *height* of a term is the greatest length of its branches.

2.2 Rational sets of terms

In this section, we study rational subsets of the algebra $\mathcal{T}(A)$. We prove that those subsets are closed by complementation.

A (bottom-up tree) *automaton* is a triple (Q, δ, F) where Q is a finite set of states, δ is a set of transitions of the form $q \leftarrow 0$ or $q \leftarrow a(q')$ or $q \leftarrow q' + q''$ (where q,q',q'' belong to Q and a to A) and F is a subset of Q of *accepting states.*

We define $\overset{\delta}{\leftarrow}$ as the smallest relation between states and terms closed by the following deduction rules:

$$\frac{}{q \overset{\delta}{\leftarrow} 0} \text{ if } q \leftarrow 0 \in \delta \ , \qquad \frac{q' \overset{\delta}{\leftarrow} t' \quad q'' \overset{\delta}{\leftarrow} t''}{q \overset{\delta}{\leftarrow} t' + t''} \text{ if } q \leftarrow q' + q'' \in \delta \ ,$$

$$\frac{q' \overset{\delta}{\leftarrow} t'}{q \overset{\delta}{\leftarrow} a(t')} \text{ if } q \leftarrow a(q') \in \delta \ .$$

A term t of $\mathcal{T}(A)$ is *recognized* by the automaton if there exists an accepting state $q \in F$ such that $q \overset{\delta}{\leftarrow} t$. A subset of $\mathcal{T}(A)$ is *rational* if it is the language of terms recognized by an automaton.

The goal of this part is to prove the closure of the rational subsets of $\mathcal{T}(A)$ by complement. Following the classical approach, the proof gives a deterministic representation to rational subsets. This cannot be done directly with automata: in their deterministic form (in the meaning that $\overset{\delta}{\leftarrow}$ associates at most one state to a given tree), those automata have only the expressive power of recognizable subsets and recognizable subsets are strictly included into rational subsets when an ACI operator is present[1].

Thus, we use vector automata: automata with vectors of integers as states. Vector automata are very much like hedge automata [1] (or forest automata): automata using infinite sets of transitions for the recognition of tree languages over unranked alphabets. Vector automata apply the same technique but with the extra commutativity property. Their use allows us to simply reuse all the results about rational subsets in commutative monoids (and in particular their closure by all boolean operations).

Given a finite set Q, we consider $Q^{(\star)}$ the commutative monoid generated by Q. The operation is written $+$ and the neutral element is denoted 0. Thus, elements of $Q^{(\star)}$ are of

[1] For instance, consider over the alphabet $\{a, b\}$ the set of trees $\{na + nb \mid n \in \mathbb{N}\}$. This set is rational, but not recognizable.

the form $\Sigma_{i=1}^{n} q_i$ for $q_i \in Q$. Another representation of $Q^{(\star)}$ is as vectors indexed by Q: the commutative monoid $\mathbb{N}^Q$. The rational subsets of $Q^{(\star)}$ are well known. It coincides with semi-linear subsets of $\mathbb{N}^Q$, or equivalently with subsets of $\mathbb{N}^Q$ satisfying a formula of Pressburger's arithmetic. Those rational subsets have the property to be closed by all the boolean operations (and in particular complementation [18, 8]). The monoid $Q^{(\star)}$ can be identified with terms of $\mathcal{T}(Q)$ of height 0 or 1: terms equal to a sum of constants. The notations are compatible. Furthermore, this identification preserves rational subsets. This remark is at the origin of vector automata.

A *vector automaton* is a triple $\mathcal{A} = (Q, (R_{q,a})_{q \in Q, a \in A}, F)$ where Q is a finite set of *state constants*, for any state constant q and any symbol $a \in A$, $R_{q,a}$ is a rational subset of $Q^{(\star)}$, and F is a rational subset of $Q^{(\star)}$ of *accepting states*. We call *state* of the automaton the elements of $Q^{(\star)}$.

We define the relation $\overset{\mathcal{A}}{\leftarrow}$ between a state u in $Q^{(\star)}$ and a term t in $\mathcal{T}(A)$ by the following inference rules:

$$\frac{q_1 \overset{\mathcal{A}}{\leftarrow} t_1 \quad \ldots \quad q_n \overset{\mathcal{A}}{\leftarrow} t_n}{\Sigma_{i=1}^{n} q_i \overset{\mathcal{A}}{\leftarrow} \Sigma_{i=1}^{n} t_i}\ , \tag{1}$$

$$\frac{u \in R_{q,a} \quad u \overset{\mathcal{A}}{\leftarrow} t}{q \overset{\mathcal{A}}{\leftarrow} a(t)}\ . \tag{2}$$

The automaton recognizes a term t if $u \overset{\mathcal{A}}{\leftarrow} t$ for some $u \in F$. We write as usual $L_{\mathcal{A}}$ the set of terms recognized by the automaton. It is important to notice that for $u \overset{\mathcal{A}}{\leftarrow} t$ then t is a rooted term iff u is a constant state (in Q). More generally, if $\Sigma_{i=1}^{k} q_i \overset{\mathcal{A}}{\leftarrow} \Sigma_{i=1}^{k'} t_i$ for t_i rooted terms, then $k = k'$ and, up to permutation of the indices, $q_i \overset{\mathcal{A}}{\leftarrow} t_i$ for all i.

The following lemma justifies this approach.

LEMMA **1** *A subset of $\mathcal{T}(A)$ is rational iff it is recognized by a vector automaton.*

We say that a vector automaton is *deterministic* if for a fixed, the sets $R_{q,a}$ are disjoint. The direct consequence of determinism is that for all term $t \in \mathcal{T}(A)$ there is at most one state $u \in Q^{(\star)}$ such that $u \overset{\mathcal{A}}{\leftarrow} t$. A vector automaton is *complete* if for all a, $\cup_{q \in Q} R_{q,a} = Q^{(\star)}$. The direct consequence of completeness is that for any $t \in \mathcal{T}(A)$, there is some $u \in Q^{(\star)}$ such that $u \overset{\mathcal{A}}{\leftarrow} t$. Under both constraints, $\overset{\mathcal{A}}{\leftarrow}$ is a mapping from $\mathcal{T}(A)$ to $Q^{(\star)}$.

LEMMA **2** *Every rational subset of $\mathcal{T}(A)$ is recognized by a deterministic and complete vector automaton.*

PROOF: Let (Q, δ, F) be a vector automaton. We construct a deterministic and complete vector automaton (Q', R', F') which recognizes the same terms.

As for determinizing finite automata, we set $Q' = 2^Q$.

Firstly we describe how to translate the rational subsets of $Q^{(\star)}$ into their 'equivalent' in $Q'^{(\star)}$: for R a subset of $Q^{(\star)}$, we define $R\!\uparrow \subseteq Q'^{(\star)}$ by

$$R\!\uparrow = \{(\Sigma_{i=1}^{n} s_i) \in Q'^{(\star)} \mid \exists (\Sigma_{i=1}^{n} q_i) \in R,\ \ \forall i, q_i \in s_i\}\,.$$

For $s \in Q'$ a state constant and $a \in A$ a symbol, we set the corresponding transition by:

$$R'_{s,a} = \{v \in Q'^{(\star)} \mid \forall\, q \in Q,\ \ q \in s \Leftrightarrow v \in R_{q,a}\!\uparrow\}\,. \tag{3}$$

The set of accepting states is $F' = F\!\uparrow$.

Validity: To make this construction valid, we need to ensure that the subsets used are rational. If the set R is rational then $R\!\uparrow$ also is[2]. Hence, F' is rational. Using the equality (equivalent to the definition)

$$R'_{s,a} = \bigcap_{q \in s} R_{q,a}\!\uparrow \ - \bigcup_{q \in Q - s} R_{q,a}\!\uparrow\,,$$

it follows that $R'_{s,a}$ is rational.

Determinism of $\mathcal{A}'$: Let v be in $R'_{s,a}$ and $R'_{s',a}$, then for all $q \in Q$, $q \in s$ iff $v \in R_{q,a}\!\uparrow$ (definition of $R'_{s,a}$) and $q \in s'$ iff $v \in R_{q,a}\!\uparrow$ (definition of $R'_{s',a}$). Thus, $s = s'$.

Completeness of $\mathcal{A}'$: Let v be a state in $Q'^{(\star)}$ and a a symbol. Let s be $\{q \in Q \mid v \in R_{q,a}\!\uparrow\}$, then $v \in R_{s,a}$ (definition of $R_{s,a}$).

Correctness: We prove the two following properties simultaneously by induction on the height h of t:

- for s a constant state of Q' and t a rooted term, then

$$s \overset{\mathcal{A}'}{\leftarrow} t \ \Leftrightarrow\ s = \{q \in Q \mid q \overset{\mathcal{A}}{\leftarrow} t\}\,, \tag{4}$$

- for a state v in $Q'^{(\star)}$ and a term t such that $v \overset{\mathcal{A}'}{\leftarrow} t$,

$$\forall R \subseteq Q^{(\star)}, \quad v \in R\!\uparrow \Leftrightarrow (\exists u \in R,\ u \overset{\mathcal{A}}{\leftarrow} t)\,. \tag{5}$$

For $h = 0$, there is no rooted terms: (4) is satisfied. By (1), $v \overset{\mathcal{A}}{\leftarrow} 0$ iff $v = 0$. By definition of $\uparrow$, $0 \in R\!\uparrow$ iff $0 \in R$. Hence (5) is satisfied.

Let $h \geq 1$ be an integer, we suppose that properties (4) and (5) are satisfied by all terms of height $< h$.

[2] It can be shown using e.g. a formula of the arithmetic of Pressburger.

(4) Let $a(t)$ be a rooted term of height h. By completeness, there is a $s \in Q'$ such that $s \overset{\mathcal{A}'}{\leftarrow} a(t)$. Let ν be such that $\nu \in R'_{s,a}$ and $\nu \overset{\delta'}{\leftarrow} t$. Such a ν exists (2). By definition (3), for all q, $q \in s$ iff $\nu \in R_{q,a}\uparrow$. Applying hypothesis of induction (5), $q \in s$ iff there is some $u \in R_{q,a}$ verifying $q \overset{\mathcal{A}}{\leftarrow} t$. Thus by rule (2), $q \in s$ iff $q \overset{\mathcal{A}}{\leftarrow} t$.

(5) Let t be a term. It can be written $\Sigma_{i=1}^{n} t_i$ with the t_i rooted. Let ν be such that $\nu \overset{\mathcal{A}'}{\leftarrow} t$. By (4) and (1), there is n state constants $s_1,\ldots,s_n$ such that $\nu = \Sigma_{i=1}^{n} s_i$ and for all i, $s_i \overset{\mathcal{A}}{\leftarrow} t_i$. By definition of $\uparrow$, $\nu \in R\uparrow$ iff there is $\Sigma_{i=1}^{n} q_i \in R$ with for all i, $q_i \in s_i$. But by (4), $s_i = \{q \mid q \overset{\mathcal{A}}{\leftarrow} t_i\}$. Hence, $\nu \in R\uparrow$ iff there is $\Sigma_{i=1}^{n} q_i \in R$ with for all i, $q_i \overset{\mathcal{A}}{\leftarrow} t_i$, or equivalently by (1), iff there is $u \in R$ with $u \overset{\mathcal{A}}{\leftarrow} t$.

Property (5) together with the definition of F' gives $L_{\mathcal{A}'} = L_{\mathcal{A}}$. □

COROLLARY **1** *The rational subsets of* $\mathcal{T}(A)$ *are closed by complement.*

According to (5), it is sufficient to replace the set of accepting states by its complementary in a deterministic and complete vector automaton recognizing a rational subset to obtain a vector automaton recognizing the complement of the rational subset.

3 Typed ACI terms

3.1 Partial algebra of typed terms with an ACI symbol

A *typing policy* is described by a set (which can be infinite) of *types* Θ, and a typing function τ_a from types to types for all symbol $a \in A$. The intended meaning is that the symbol a, when applied to an argument of type $\tau_a(\theta)$ has type θ. We will often refer to the typing policy as a graph: the vertices are the types, and there is an edge between type θ and type θ' labelled by a if $\tau_a(\theta) = \theta'$. Notice that it does not correspond to the arrow notation used in classical function types. As τ is a function, the graph is deterministic.

We say that a term t has type θ for the typing policy (Θ,τ), written $(\Theta,\tau) \vdash t : \theta$ if one can derive the judgment $(\Theta,\tau) \vdash t : \theta$ by the following rules:

$$\frac{\theta \in \Theta}{(\Theta,\tau) \vdash 0 : \theta}, \qquad \frac{(\Theta,\tau) \vdash t : \tau_a(\theta) \quad a \in A}{(\Theta,\tau) \vdash a(t) : \theta}, \qquad \frac{(\Theta,\tau) \vdash t : \theta \quad (\Theta,\tau) \vdash t' : \theta}{(\Theta,\tau) \vdash t + t' : \theta}.$$

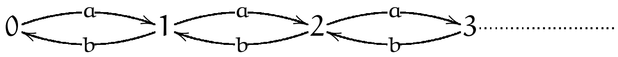
The second rule can only be applied if τ_a is defined. For this reason, some terms may not be typable. We call *typed term* the couple of a term t and a type θ, and we write it $t : \theta$. The set of typed terms $t : \theta$ such that $(\Theta,\tau) \vdash t : \theta$ is written $\mathcal{T}^{\Theta,\tau}(A)$. We write $\mathcal{T}_{\theta}^{\Theta,\tau}(A)$ the restriction of $\mathcal{T}^{\Theta,\tau}(A)$ to typed terms of type θ.

The following property gives a simple graphical interpretation of typing.

PROPOSITION **1** *The terms in* $\mathcal{T}_\theta^{\Theta,\tau}(A)$ *are the terms such that all branches corresponds to a path of origin* θ *in the graph* (Θ,τ)[3].

Example 1: If $\Theta = \{\theta\}$ and $\tau_a(\theta) = \theta$ for all $a \in A$ then $\mathcal{T}_\theta^{\Theta,\tau}(A) = \mathcal{T}(A)$ (up to type removal). It corresponds to the domain of (untyped) process rewrite systems.

Example 2: Let Θ be the natural integers, A be $\{a,b\}$ and τ be such that $\tau_a(n) = n+1$, $\tau_b(n+1) = n$. Graphically,



The paths of origin 0 in this graph contain more a's than b's. Reciprocally, to all word over $\{a,b\}$ such that all prefixes contain more a's than b's corresponds a path of origin 0 in the graph. Thus, according to Proposition 1, terms of type 0 are such that all the branches contain more a's than b's (recall the branches are prefix closed). This is an example of an infinite typing policy.

From now and on, the typing policy is fixed. Thus, we simplify slightly the notation $(\Theta,\tau) \vdash t : \theta$ by writing $\vdash t : \theta$. Furthermore, by convention, we will suppose that there is a type $\bullet \in \Theta$ of out-degree 0 (no term but 0 has type $\bullet$), and for any newly introduced symbol q, $\tau_q(\theta) = \bullet$ for $\theta \in \Theta - \{\bullet\}$. Thus, all newly introduced symbol behaves like a constant of any type (but $\bullet$).

We need now a way to perform computations on the typing policy, even if there is an infinite number of types. The monadic second order logic presented in the next section serves this purpose.

3.2 Monadic second order logic

In this part, we briefly recall the basis of monadic second order logic (MSO).

The MSO logic expresses properties on graphs. Our purpose is to apply it to the typing policy. We thus adapt slightly the usual notations to fit with this use. A *MSO formula* follows the syntax:

$$\begin{array}{rcl} \Phi & ::= & \exists\theta, \Phi \mid \forall\theta,\ \Phi \\ & \mid & \exists X, \Phi \mid \forall X,\ \Phi \\ & \mid & \tau_a(\theta) = \theta \mid \theta \in X \\ & \mid & \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid \textit{true} \mid \textit{false}\ . \end{array}$$

The *first order variables* $(\theta,\theta',\ldots)$ range over types while the *monadic second order variables* (written in capital letters X,Y,...) range over sets of types. All the boolean connectives are

[3] Notice that, as the graph is deterministic, there is no ambiguity about those paths.

allowed as well as any quantification over first and second order variables. Atomic predicates allow to test the typing policy ($\theta = \tau_a(\theta')$) and the membership of a first order variable in a second order variable ($\theta \in X$). We allow ourselves to use some more complex notations. For instance we can use variables over known finite domains (e.g the states or the rules of an automaton) and use quantification over them. The reader must keep in mind that all those extensions are only syntactic sugar and can be encoded into standard MSO formulas as defined previously.

Whenever a typing policy (Θ, τ) satisfies a MSO formula Φ, we write $(\Theta, \tau) \models \Phi$ and say that (Θ, τ) is a *model* of Φ. A typing policy is said to have a *decidable MSO theory* if one can decide, given a MSO formula, whether the typing policy is a model of the formula. The typing policy being fixed, we just write $\models \Phi$ instead of $(\Theta, \tau) \models \Phi$.

As a useful example, we define for any term t the predicate $\mathit{hastype}_t$ such that for any type θ, $\models \mathit{hastype}_t(\theta)$ iff $\vdash t{:}\theta$. The predicate is built by induction:

$$\mathit{hastype}_{\Sigma_{i=1}^{n} a_i(t_i)}(\theta) \;\; \equiv \;\; \forall i \in [1, n],\ \exists \theta',\ \tau_{a_i}(\theta) = \theta' \wedge \mathit{hastype}_{t_i}(\theta')$$

The MSO logic has been extensively studied and many classes of (possibly infinite) graphs are known to have a decidable MSO theory [17, 5, 2, 10, 3]. The infinite typing policy of Example 2 belongs to the simplest of those families: pushdown graphs.

3.3 MSO-guarded rational sets of typed terms

In this section we extend the notion of rational subsets of $\mathcal{T}(A)$ to MSO-guarded rational subsets of $\mathcal{T}^{\Theta,\tau}(A)$. To this purpose, we introduce MSO-guarded automata: automata such that transitions can be applied if and only if a certain MSO formula is satisfied by the type of the terms involved in the transition. Apart from this distinction, the techniques involved in this section are very similar to the ones of Section 2.2.

Formally, a *MSO-guarded automaton* over $\mathcal{T}^{\Theta,\tau}(A)$ is a triple (Q, δ, F) where Q is a finite set of states, δ is a finite set of transitions of the form $q \leftarrow_{\Phi} 0$, $q \leftarrow_{\Phi} q' + q''$ or $q \overset{\mathcal{A}}{\leftarrow}_{\Phi} a(q')$ with q, q' and q'' states and Φ a MSO formula, and $F \subseteq Q$ is the set of accepting states. The MSO formulas Φ are called *guards*. Each guard Φ has precisely one free variable and we can bind it by using a functional notation: $\Phi(\theta)$. The MSO-guarded automata behave like standard automata, the only difference being that transitions can be applied if and only if the guard is satisfied by the type of the term involved.

We define the relation $\overset{\delta}{\leftarrow}$ between a state and a typed term as the smallest relation satisfying:

$$\frac{}{q \overset{\delta}{\leftarrow} 0{:}\theta} \text{ if } q \leftarrow_{\Phi} 0 \in \delta \text{ and } \models \Phi(\theta)\ ,$$

$$\frac{q' \overset{\delta}{\leftarrow} t'{:}\tau_a(\theta)}{q \overset{\delta}{\leftarrow} a(t'){:}\theta} \text{ if } q \leftarrow_\Phi a(q') \in \delta \text{ and } \models \Phi(\theta)\ ,$$

$$\frac{q' \overset{\delta}{\leftarrow} t'{:}\theta \quad q'' \overset{\delta}{\leftarrow} t''{:}\theta}{q \overset{\delta}{\leftarrow} t' + t''{:}\theta} \text{ if } q \leftarrow_\Phi q' + q'' \in \delta \text{ and } \models \Phi(\theta)\ .$$

A typed term $t : \theta$ is recognized by the automaton if there is an accepting state $q \in F$ such that $q \overset{\delta}{\leftarrow} t : \theta$. We will write $L_\delta(q)$ the set of typed terms $t : \theta$ such that $q \overset{\delta}{\leftarrow} t : \theta$. A set of typed term $R \subseteq \mathcal{T}^{\Theta,\tau}(A)$ is MSO-guarded rational if there is a MSO-guarded automaton recognizing exactly the typed terms of R. A term can be present with different types in the same rational subset, but, as the type information is always kept along with the term, no confusion arises. On the contrary, the states of the automaton are not typed: it is not necessary. The important point is that the satisfaction of the guard depends only of the type of the term but not at all of the term itself. We will furthermore suppose that the rules enforce the typing of terms: if there is a rule of the form $q \overset{\delta}{\leftarrow}_\Phi a(q')$, then for all type θ, $\models \Phi(\theta) \Rightarrow \mathit{hastype}_a(\theta)$. Thus, this rule can be applied only if it is sensible to consider a term of root a.

The following theorem extends naturally the previous one to MSO-guarded rational sets.

THEOREM **1** *The MSO-guarded rational subsets of $\mathcal{T}^{\Theta,\tau}(A)$ are closed by complementation.*

The proof works as in the untyped case. One defines similarly MSO-guarded vector automata. The trick is that there is only a finite number of possible valuations for $(\Phi_1(\theta), \ldots, \Phi_k(\theta))$ for type θ (where $\Phi_1,\ldots,\Phi_k$ are the guards of an automaton). Given a valuation ν, by combining together the guards using the boolean connectives, it is easy to obtain a new guard Φ_ν such that $\Phi_\nu(\theta)$ is satisfied if and only if $(\Phi_1(\theta), \ldots, \Phi_k(\theta)) = \nu$. It is then sufficient to apply the techniques of the previous demonstration for each of this new guards.

This proof is completely syntactic in the meaning that there is no need to check the satisfaction of the guards. The guards are simply combined together using the boolean connectives. For this reason, the result remains correct if the typing policy has not a decidable monadic theory. This is also the case for the following lemma.

LEMMA **3** *If R is a MSO-guarded rational subset of $\mathcal{T}^{\Theta,\tau}(A)$, then there is a MSO formula empty_R such that:*

$$\models \mathit{empty}_R(\theta) \quad \mathit{iff} \quad R \cap \mathcal{T}^{\Theta,\tau}_\theta(A) = \emptyset\ .$$

PROOF: Let (Q, δ, F) be a MSO-guarded automaton. The principle is to compute the sets of types X_q for $q \in Q$ such that $\theta \in X_q$ if and only if $q \overset{\delta}{\leftarrow} t{:}\theta$ for some t.

$$\begin{array}{llllll}
\multicolumn{6}{l}{\mathit{empty}_R(\theta_0) \equiv \forall (X_q)_{q\in Q},} \\
 & \forall q \leftarrow_\Phi 0 \in \delta, & \forall \theta, & \Phi(\theta) & \Rightarrow \theta \in X_q & (a) \\
\wedge & \forall q \leftarrow_\Phi q' + q'' \in \delta, & \forall \theta, & (\Phi(\theta) \wedge \theta \in X_{q'} \wedge \theta \in X_{q''}) & \Rightarrow \theta \in X_q & (b) \\
\wedge & \forall q \leftarrow_\Phi a(q') \in \delta, & \forall \theta, \theta', & (\Phi(\theta) \wedge \tau_a(\theta) = \theta' \wedge \theta' \in X_{q'}) & \Rightarrow \theta \in X_q & (c) \\
 & \multicolumn{5}{l}{\Rightarrow \exists q \in F,\ \theta_0 \in X_q}
\end{array}$$

The sets (X_q) such that $\theta \in X_q$ iff $q \overset{\delta}{\leftarrow} t : \theta$ for some t are the smallest sets solution of constraints (a), (b) and (c). For instance constraint (a) can be read: «if there is a transition $q \leftarrow_\Phi 0$ in the automaton, then, for all type θ such that the guard is satisfied, there is a term of type θ in $L_\delta(q)$.» As the constraints are continuous (in the meaning of complete partial orders), testing if $\theta_0 \in X_q$ for the smallest solution of (a), (b) and (c) amounts to verify that $\theta_0 \in X_q$ for all solution of (a), (b) and (c). This is what performs the universal quantification over (X_q). □

COROLLARY **2** *If the typing policy has a decidable monadic theory then the emptyness of* R *is decidable.*

4 Ground rewrite systems

4.1 Typed process rewrite systems

In this section, we introduce MSO-guarded rational process rewrite systems. It is a natural extension to types of process rewrite systems. Following the scheme of Mayr's proof [15], we then state a normalization lemma.

DEFINITION **1** *A MSO-guarded rational process rewrite system over* $\mathcal{T}^{\Theta,\tau}(A)$ *labelled by* E *is a finite set* Δ *of rules of the form* $R \xrightarrow{e} R'$ *where* R *and* R' *are MSO-guarded rational subsets of* $\mathcal{T}^{\Theta,\tau}(A)$ *and* e *is a label in* E.

A transition of the process rewrite systems corresponds to the replacement of a subterm by another according to the set of rules: we define inductively between typed terms of same type $t_1{:}\theta$ and $t_2{:}\theta$ the rewrite judgment $t_1 \overset{e}{\Rightarrow}_\Delta t_2{:}\theta$ by

$$\frac{t_1{:}\theta \in R_1 \quad t_2{:}\theta \in R_2 \quad R_1 \xrightarrow{e} R_2 \in \Delta}{t_1 \overset{e}{\Rightarrow}_\Delta t_2{:}\theta},$$

$$\frac{\vdash t{:}\theta \quad t_1 \overset{e}{\Rightarrow}_\Delta t_2{:}\theta}{t + t_1 \overset{e}{\Rightarrow}_\Delta t + t_2{:}\theta}, \qquad \frac{t_1 \overset{e}{\Rightarrow}_\Delta t_2{:}\tau_a(\theta)}{a(t_1) \overset{e}{\Rightarrow}_\Delta a(t_2){:}\theta}.$$

We also define the reflexive and transitive closure $\overset{*}{\Rightarrow}_\Delta$ of $\overset{e}{\Rightarrow}_\Delta$ by:

$$\frac{\vdash t{:}\theta}{t \overset{*}{\Rightarrow}_\Delta t{:}\theta}\,, \qquad \frac{t \overset{*}{\Rightarrow}_\Delta t'{:}\theta \quad t' \overset{e}{\Rightarrow}_\Delta t''{:}\theta}{t \overset{*}{\Rightarrow}_\Delta t''{:}\theta}\,.$$

When $t \overset{*}{\Rightarrow}_\Delta t'{:}\theta$, we will say that there is a path of type θ between t and t' for the rules Δ. We will omit to specify the set of rules when there is no ambiguity about it.

DEFINITION **2** *A MSO-guarded rational process rewrite system Δ is normalized if for all rule $R_1 \xrightarrow{e} R_2 \in \Delta$ there is a MSO formula Φ and two terms t_1, t_2 such that $R_1 = \{t_1 : \theta \mid\ \models \Phi(\theta)\}$, $R_2 = \{t_2{:}\theta \mid\ \models \Phi(\theta)\}$ and (t_1, t_2) has one of the following forms:*

- *sequential rules:* $(a, b(c))$ *or* $(a(b), c)$
- *parallel rules:* $(a, b + c)$, $(a + b, c)$, $(0, a)$, $(a, 0)$ *or* (a, b).

In this case, we write Δ^{seq} the set of sequential rules and Δ^{par} the set of parallel rules. We also use notations similar to MSO-guarded automata for rules: $a \xrightarrow{e}_\Phi b(c) \in \Delta$, $a(b) \xrightarrow{e}_\Phi c \in \Delta, \dots$

PROPOSITION **2** *Given Δ a MSO-guarded rational process rewrite system over $\mathcal{T}^{\Theta,\tau}(A)$, there is a MSO-guarded normalized process rewrite system Δ' over $\mathcal{T}^{\Theta,\tau}(A \cup C)$ (where C is a finite set of new constants of any type) such that for all $t, t' \in \mathcal{T}^{\Theta,\tau}(A)$, $t \overset{*}{\Rightarrow}_\Delta t'$ iff $t \overset{*}{\Rightarrow}_{\Delta'} t'$.*

PROOF: Let $R_1 \xrightarrow{e_1} R'_1, \dots, R_k \xrightarrow{e_k} R'_k$ be the rules in Δ with $R_i = L_{(Q_i,\delta_i,F_i)}$ and $R'_i = L_{(Q'_i,\delta'_i,F'_i)}$. We assume, without loss of generality, that the Q_i's, the Q'_i's and A are all disjoint. Let C be $Q_1 \cup \dots \cup Q_k \cup Q'_1 \cup \dots \cup Q'_k$. We define now Δ' by:

$$\begin{array}{rcl} \Delta' & = & \{t \xrightarrow{\$}_\Phi q \mid \exists i,\ q \xleftarrow{\delta}_\Phi t \in \delta_i\} \\ & \cup & \{q \xrightarrow{e}_{\mathit{true}} q' \mid \exists i,\ q \in F_i,\ e = e_i,\ q' \in F'_i\} \\ & \cup & \{q \xrightarrow{\$}_\Phi t \mid \exists i,\ q \xleftarrow{\delta}_\Phi t \in \delta'_i\} \end{array}$$

The normalized process rewrite system mimics the behavior of the automata by using exactly the same rules (labelled by a dummy symbol \$). Obviously, if $t \overset{*}{\Rightarrow}_\Delta t'$ then $t \overset{*}{\Rightarrow}_{\Delta'} t'$. The other direction is more technical, we do not show it here. □

4.2 Reachability in typed process rewrite systems

In this section we show that (providing that the typing policy has a decidable monadic theory) the reachability problem between constants is decidable for typed process rewrite systems.

Many proofs of reachability in infinite state systems rely on the rationality of the set of reachable states (for instance [7] for ground rewrite systems or [13] for PA processes). Such an approach cannot be used in the case of process rewrite systems (typed or not): the closure properties of rational subset would give the decidability of the equivalence of reachable sets for process rewrite systems and this problem is not decidable (it has been proved for the subclass of Petri Nets [9]).

The core of our approach is a direct translation to typed terms of [15]. The idea is to obtain a representation of the (potentially infinite) set:

$$\Lambda_\Delta = \{(a, b, \theta) \mid a \overset{*}{\Rightarrow}_\Delta b : \theta\}$$

To this purpose, we use the following lemma:

LEMMA **4** *The set Λ_Δ is the smallest set satisfying:*

$$\frac{a \xrightarrow{e}_\Phi b(c) \in \Delta \quad \models \Phi(\theta) \quad (c, c', \tau_b(\theta)) \in \Lambda_\Delta \quad b(c') \xrightarrow{e'}_{\Phi'} a' \in \Delta \quad \models \Phi'(\theta)}{(a, a', \theta) \in \Lambda_\Delta} ,$$

$$\frac{a \overset{*}{\Rightarrow}_{\Delta'} b \qquad \Delta' = \Delta_\theta^{par} \cup \{c \xrightarrow{\$} d : \theta \mid (c, d, \theta) \in \Lambda_\Delta\}}{(a, b, \theta) \in \Lambda_\Delta} .$$

The first rule states that a path of type θ between a and b can start with a rule $a \xrightarrow{e}_\Phi b(c)$ and end with a rule $b(c') \xrightarrow{e'}_{\Phi'} a'$ providing that the guard are satisfied and that there is a path of type $\tau_b(\theta)$ between c and c'. The second rule states that if there is a path of type θ between a and b using the parallel rules and what is already known in Λ_Δ, then there is a path of type θ between a and b. As a consequence, the second rule inserts all the paths of length 0 (i.e. (a, a, θ) with $\vdash a : \theta$). The test $a \overset{*}{\Rightarrow}_{\Delta'} b$ performed by the second rule is handled by the following lemma:

LEMMA **5** *If Δ contains only parallel rules and no guard (equivalently the guard is true), then $a \overset{*}{\Rightarrow}_\Delta b : \theta$ is decidable.*

This is an instance of the reachability problem for Petri nets. It is decidable [14, 11].

In Mayr's proof, the set Λ_Δ is finite (there is no type information) and a saturation algorithm is sufficient for computing it. In our case, we can define this set by monadic formulas:

LEMMA **6** *Given a normalized MSO-guarded process rewrite system Δ over $\mathcal{T}^{\Theta,\tau}(A)$, there is $|A|^2$ monadic formulas $\lambda_{a,b}$ for a, b in A such that:*

$$(a, b, \theta) \in \Lambda_\Delta \textit{ iff } \models \lambda_{a,b}(\theta)$$

The set Λ_Δ can be represented by $|A|^2$ second order variables $(X_{a,b})_{a,b \in A}$:

$$\theta \in X_{a,b} \text{ iff } (a, b, \theta) \in \Lambda_\Delta$$

Lemma 4 describes the set Λ_Δ as the smallest set satisfying some constraints. Thus, a technique similar to the proof of Lemma 3 can be used. The test $a \overset{*}{\Rightarrow}_{\Delta'} b$ with $\Delta' = \Delta_\theta^{par} \cup \{c \overset{\$}{\rightarrow} d : \theta \mid (c, d, \theta) \in \Lambda_\Delta\}$ can be performed using Lemma 5 and the remark that there is only a finite number of possible parallel rules for a given set of symbol A. Thus all the cases can be treated in a MSO formula of exponential size (one for each of the possible sets of parallel rules).

4.3 Property reachability

In this section we show that (providing that the typing policy has a decidable monadic theory), the reachable property problem is decidable (Theorem 2).

Given a MSO-guarded rational process rewrite system Δ with labels in E, a property has the following syntax:

$$\varphi \ ::= \ e \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \ ,$$

with $e \in E$. An atomic property e is satisfied by a term t if there is some t' such that $t \overset{e}{\Rightarrow}_\Delta t'$. The boolean connectives have their standard semantics. An instance of the reachability property problem is: «given a MSO-guarded rational process rewrite system Δ, a term t and a property φ, is it possible to reach a term satisfying φ from t by using the rewriting rules in Δ?».

To show this result, we just have to prove that the set of terms satisfied by a property is MSO-guarded rational. It is sufficient to do this for atomic properties according to the closure by union, intersection and complementation of MSO-guarded rational sets.

LEMMA **7** *Given a label* $e \in E$*, the set of typed terms* $t : \theta$ *such that* $t \overset{e}{\Rightarrow}_\Delta t' : \theta$ *for some term* t' *is MSO-guarded rational.*

PROOF: Thanks to the closure of MSO-guarded rational subsets by union, it is sufficient to show the result for only one rule labelled by e. Let $R_1 \overset{e}{\rightarrow} R_2$ be such a rule. One can apply this rule on a term t if and only if it contains a subterm t' in R_1 of type θ and there is a term of type θ in R_2. Let $\mathcal{A} = (Q, \delta, F)$ be the automaton recognizing R_1. We construct the automaton (Q', δ', F') recognizing the set of terms such that $t \overset{e}{\Rightarrow}_\Delta t' : \theta$ for some term t' by:

$$\begin{aligned}
Q' &= Q \cup \{q_0, q_1\} \\
\delta' &= \delta \\
&\cup \ \{q_1 \leftarrow_\Phi t \mid q \leftarrow_\Psi t \in \delta,\ q \in F,\ \Phi(\theta) = \Psi(\theta) \wedge \neg \mathit{empty}_{R_2}(\theta)\} \\
&\cup \ \{q_1 \leftarrow q_1 + q_0\} \\
&\cup \ \{q_0 \leftarrow a(q_0) \mid a \in A\} \cup \{q_0 \leftarrow 0,\ q_0 \leftarrow q_0 + q_0\} \\
F' &= \{q_1\}
\end{aligned}$$

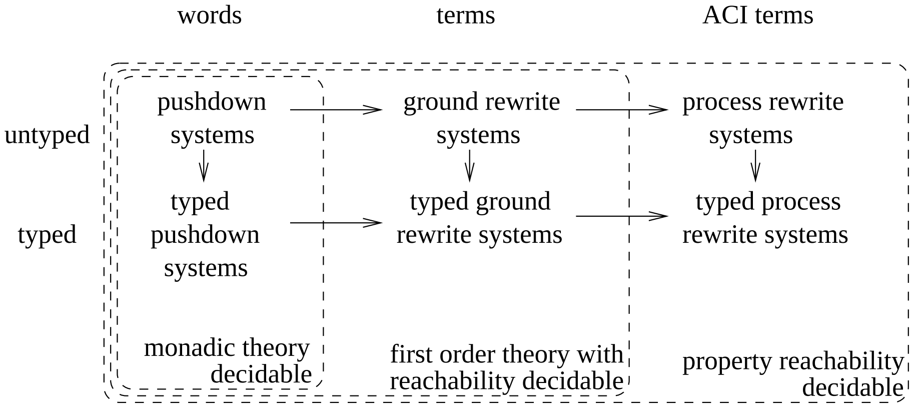


Figure 1: A hierarchy of ground rewrite systems.

The states in Q perform the same computation as in $\mathcal{A}$. The state q_0 recognizes any term of $\mathcal{T}^{\Theta,\tau}(A)$. Notice the use of the *empty* predicate for checking that there is a term of type θ in R_2. □

THEOREM **2** *If the typing policy has a decidable monadic theory then the reachable property problem of typed process rewrite systems is decidable.*

PROOF: Given a tree t and a type θ, then the set $R_1 = \{t : \theta\}$ is MSO-guarded rational. According to Lemma 7, the set R_2 of typed terms satisfying the property is MSO-guarded rational. We add to the system the new symbols a_1 and a_2 and the transitions $a_i \rightarrow_{true} R_1$ and $R_2 \rightarrow_{true} a_2$. Then, the satisfaction of the property reachability problem amounts to check the satisfaction of $\lambda_{a_1,a_2}(\theta)$. □

5 Conclusion

In this paper we studied the partial algebra of typed terms with an ACI operator. We have shown that results on untyped terms could be simply raised to the level of typed terms. This phenomenon can be presented more generally as depicted in Figure 1 where different known systems are presented in the common framework of ground rewriting. Each of the families of systems can be of finite or infinite degree (finite number of rewrite rules or rational set of rules).

The pushdown systems correspond to a finite set of prefix rewriting rules on words [17]. The corresponding systems of infinite degree are the prefix recognizable graphs [2]. The first typed version can be found in [6] but for different purposes. A study of those systems can be found in [3] in a different framework: there is no explicit references to types. Given a deterministic graph (which can be seen as the typing policy), it is unfolded and then an inverse rational substitution is applied to it. Those two transformations exactly correspond to considering a «typed prefix recognizable graph» over this typing policy. All those systems seen as graphs share a decidable monadic theory.

The ground rewrite systems in the free algebra of terms were first studied in [7] and then in a structural way in [12]. The typed version in [4] is presented in the framework of graphs solution of infinite equational systems with the vertex replacement with product operators. The infinite equational system exactly corresponds to the typing policy. All those systems share a decidable first order theory with reachability, but, because they contain the infinite grid, the monadic theory is undecidable.

The ground rewrite systems over the term algebra with ACI symbols are presented in [15] under the name of process rewrite systems. The present paper extends those results to the partial algebra of typed terms with ACI symbols. Those systems share the decidability of the property reachability. The first order theory with reachability is undecidable (one can encode the problem of equivalence of reachable states in Petri nets, and it has been shown undecidable in [9]).

References

[1] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Ver sion 1.

[2] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, volume 1099 of *LNCS*, pages 194–205, 1996.

[3] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, 2002.

[4] T. Colcombet. On families of graphs having a decidable first order theory with reachability. In *ICALP'02*, 2002.

[5] B. Courcelle. The monadic second order logic of graphs ix: Machines and their behaviours. In *Theoretical Computer Science*, volume 151, pages 125–162, 1995.

[6] W. Damm. Languages defined by higher type program schemes. In A. Salomaa and M. Steinby, editors, *ICALP'77*, volume 52 of *LNCS*, pages 164–179, 1977.

[7] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.

[8] S. Eilenberg and M.-P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969.

[9] M. Hack. The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems. In *15th annual symposium on switching and automata theory*, New York, 1974.

[10] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen, editor, *FOSSACS'02*, LNCS, 2002.

[11] S. Kosaraju. Decidability of reachability in vector addition systems. In *14th Annual Symposium on Theory of Computing*, 1982.

[12] C. Löding. Ground tree rewriting graphs of bounded tree width. In *Stacs O2*. LNCS, 2002.

[13] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, volume 1466, pages 50–66. Springer, 1998.

[14] E. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on computing*, 13:441–460, 1981.

[15] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[16] R. Mayr and M. Rusinowitch. Reachability is decidable for ground ac rewrite systems. In *INFINITY'98*, 1998.

[17] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[18] E. Spanier S. Ginsburg. Semigroups, presburger formulas and languages. *Pacific J. Maths 16*, pages 285–296, 1966.

The synchronized graphs trace the context-sensitive languages

Chloé RISPAL

Irisa, Campus de Beaulieu, Rennes, France

E-mail: rispal@univ-mlv.fr

Abstract

Morvan and Stirling have proved that the context-sensitive languages are exactly the traces of graphs defined by transducers with labelled final states. We prove that this result is still true if we restrict to the traces of graphs defined by synchronized transducers with labelled final states. From their construction, we deduce that the context-sensitive languages are the languages of path labels leading from and to rational vertex sets of letter-to-letter rational graphs.

1 Introduction

As for formal languages, an infinite graph hierarchy exists. First of all, Muller and Schupp [MS 85] have defined the transition graphs of pushdown automata. Then, Courcelle has defined the family of equational graphs which are the graphs generated by deterministic graph grammars [Co 90]. Caucal has extended these families to prefix recognizable graphs which are the prefix transitions of recognizable systems [Cau 96]. More recently, Morvan has introduced the rational graphs which are recognized by word transducers with labelled final states [Mo 00]. Finally, Caucal has presented the transition graphs of Turing machines [Cau 01].

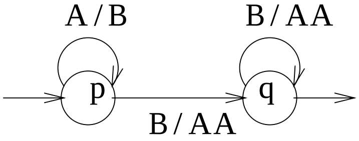
A trace of a graph is the language of path labels leading from and to finite vertex sets. Traces of graphs are a link between infinite graph hierarchy and the Chomsky hierarchy of languages. The traces of finite graphs are the rational languages, the traces of prefix recognizable graphs are the context-free languages [Cau 96], the traces of rational graphs are the context-sensitive languages [MoS 01] and finally, the traces of Turing graphs are the recursively enumerable languages [Cau 01].

A particular rational relation is the left-synchronized relation which is recognized by a letter-to-letter transducer followed by a recognizable relation for each final state [EM 65] and [FS 93]. These left-synchronized relations form a boolean algebra and are recognized by deterministic transducers. A graph is synchronized if it is isomorphic to some graph having words as vertices and such that each labelled transition is a left-synchronized relation. The synchronized graphs are the automatic graphs of Blumensath and Grädel [BG 00]. In this paper, we adapt the construction of Morvan and Stirling [MoS 01] to prove that the context-sensitive languages are exactly the traces of synchronized graphs. We also characterize the context-sensitive languages as the languages of path labels leading from and to rational vertex sets of letter-to-letter rational graphs.

2 Rational synchronized graphs

Let N be a finite alphabet. We denote by N^* the set of words over letters of N, and we write ε for the empty word.
A *transducer* T is defined by a finite subset of $Q{\times}N^*{\times}N^*{\times}Q$ of labelled edges where Q is a finite set of states, by a set $I \subseteq Q$ of initial states, and by a set $F \subseteq Q$ of final states. So a transducer is a finite automaton labelled by pairs of words. Any transition (p,u,v,q) of a transducer T will be denoted by $p \xrightarrow[T]{u/v} q$ or by $p \xrightarrow{u/v} q$ when T is understood.
A path $p_0 \xrightarrow{u_1/v_1} p_1 \dots p_{n-1} \xrightarrow{u_n/v_n} p_n$ with $u = u_1...u_n$ and $v = v_1...v_n$ is labelled u/v and is denoted by $p_0 \overset{u/v}{\underset{T}{\Longrightarrow}} p_n$. A path is *successful* if it leads from an initial state to a final one. A pair $(u,v) \in N^*{\times}N^*$ is recognized by a transducer if there exists a successful path labelled u/v. A relation is *rational* if it is recognized by a transducer.

Example 2.1 The following transducer:



with initial state p and final state q recognizes the rational relation

$$\{\, (A^nB^m\,,\, B^nA^{2m}) \mid n \geq 0, m > 0 \,\}.$$

From studies concerning rational relations, Elgot and Mezei [EM 65] and then Frougny and Sakarovitch [FS 93] have defined the subfamily of left-synchronized relations.

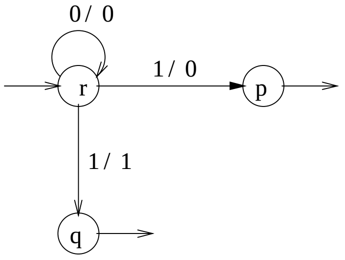
If a transducer has labels over $N{\times}N$ it is called a *letter-to-letter 2-automaton*: it is a transducer labelled by pairs of letters instead of pairs of words. Adding a rational terminal function completing one side of the recognized pairs, it recognizes a left-synchronized relation.

Definition 2.2 A relation over $N^*{\times}N^*$ is *left-synchronized* if it is recognized by a letter-to-letter 2-automaton with terminal function taking values in

$$\mathrm{Dif}_{\mathrm{Rat}} \;=\; (\mathrm{Rat}(N^*) \times \{\varepsilon\}) \,\cup\, (\{\varepsilon\} \times \mathrm{Rat}(N^*))$$

That is a left-synchronized relation is a finite union of elementary relations of the form $R.S$ where $R \in \mathrm{Rat}((N \times N)^*)$ and $S \in \mathrm{Dif}_{\mathrm{Rat}}$

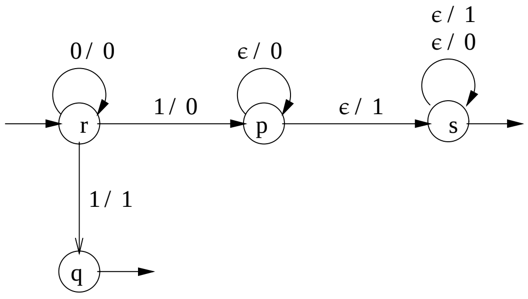
Example 2.3 For all integer p, the relation $|_p$ defined by $x|_p y$ if x is a power of p dividing y, is left-synchronized. For instance, in base two with weak weigths on the left, $|_2$ is recognized by the following letter-to-letter 2-automaton:



with the terminal function f defined by $f(p) = (\varepsilon, 0)^*(\varepsilon, 1)\{(\varepsilon, 0), (\varepsilon, 1)\}^*$ and $f(q) = (\varepsilon, \varepsilon)$

As the terminal function is rational, it can be introduced in the transducer. A *left-synchronized transducer* is a transducer such that each path leading from an initial vertex to a final one can be divided in two parts: The first one only contains edges of the form
$\{p\xrightarrow{A/B}q | p, q \in Q \wedge A, B \in N\}$ while the second part contains edges of the form
$\{p\xrightarrow{A/\epsilon}q | p, q \in Q \wedge A \in N\}$ exclusive or $\{p\xrightarrow{\epsilon/B}q | p, q \in Q \wedge B \in N\}$.

Example 2.4 The following left-synchronized transducer recognizes the left-synchronized relation of example 2.3.



A *right-synchronized* relation is defined symmetrically using a rational initial function. The left-synchronized relations form a subfamily of rational relations with useful closure properties.

Theorem 2.5 [EM 65] *The synchronized relations form a boolean algebra.*

We will use also particular left-synchronized relations. A binary relation R is *recognizable* if it is a finite union of products $S \times T$ where $S, T \in Rat(N^*)$. A binary relation R over words is of *bounded length difference* if there exists an integer b such that $||u| - |v|| \leq b$ for any $(u, v) \in R$.

Proposition 2.6 [FS 93] *The family of synchronized relations contains the recognizable relations and the rational relations of bounded length difference.*

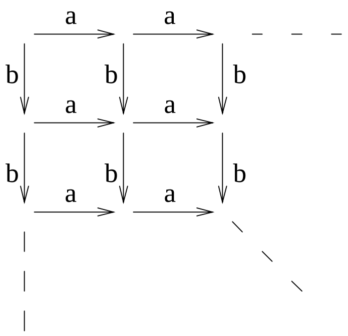
Proof. Let $\mathcal{A}$ be a finite set of labels. A simple edge labelled *graph* is a subset of $V{\times}\mathcal{A}{\times}V$ where V is an arbitrary set of *vertices*. For any label $a \in \mathcal{A}$, the a-*transition* of

a graph G is the relation $\xrightarrow[G]{a} := \{\ (s,t)\ |\ (s,a,t)\ \in\ G\ \}$. A graph $G \subseteq N^*{\times}\mathcal{A}{\times}N^*$ is *left-synchronized* if for each $a \in \mathcal{A}$, the relation $\xrightarrow[G]{a}$ is left-synchronized. An arbitrary graph is synchronized if it is isomorphic to some left-synchronized graph.

Definition 2.7 A graph G is *synchronized* (respectively *rational, rational of bounded length difference*) if it is isomorphic to some graph $G \subseteq N^*{\times}\mathcal{A}{\times}N^*$ such that for each $a \in \mathcal{A}$, the relation $\xrightarrow[G]{a}$ is left-synchronized (respectively rational, rational and of bounded length difference).

Note that the synchronized graph family is also the closure by isomorphism of the rigth-synchronized graphs.

Example 2.8 The following *grid* :



is synchronized because we can code its vertices by words to get the following left-synchronized graph G defined by $\xrightarrow[G]{a} \;=\; (A,A)^*(B,A)(B,B)^*(\varepsilon,B)$ and $\xrightarrow[G]{b} \;=\; (A,A)^*(B,B)^*(\varepsilon,B)$. Note that it is also a rational graph of bounded length difference.

Synchronized graphs are the automatic graphs of Blumensath and Grädel [BG 00]. These graphs have a decidable first order theory. But the accessibility of these graphs is undecidable in general.

3 Traces of synchronized graphs

A *trace* of a graph G is the language $L(G,I,F)$ of path labels leading from a set I of initial vertices to a set F of final vertices:

$$L(G,I,F) \;=\; \{\, u \mid \exists\, s \in I\; \exists\, t \in F,\; s \overset{u}{\underset{G}{\Longrightarrow}} t \,\}$$

but with the condition that I and F are finite.

Morvan and Stirling [MoS 01] have proved that the traces of rational graphs are the context-sensitive languages. So any trace of a synchronized graph is a context-sensitive language. It remains to show that any context-sensitive language L is also the trace of a synchronized graph. We get this result by adapting the construction of [MoS 01].

We only need to find a left-synchronized graph $G \;\subseteq\; N^*{\times}\mathcal{A}{\times}N^*$ and two rational sets $I, F \in Rat(N^*)$ such that $L = L(G,I,F)$.

Lemma 3.1 *Let* $G \subseteq N^* {\times} \mathcal{A} {\times} N^*$ *be a left-synchronized graph.*
Let $I, F \in \mathrm{Rat}(N^*)$ *and* $i, f \notin N$.
There exists a left-synchronized graph $H \subseteq (N^* \cup \{i,f\}) {\times} \mathcal{A} {\times} (N^* \cup \{i,f\})$ *such that*

$$L(G,I,F) \ = \ L(H,\{i\},\{f\}).$$

Proof. **i)** For all $a \in \mathcal{A}$, the relation $\xrightarrow[G]{a}$ is left-synchronized.
We define the graph $G^{'}$ by erasing all edges of G leading to a terminal state of F. This graph $G^{'}$ is still left-synchronized as for all $a \in \mathcal{A}$, the relation

$$\xrightarrow[G^{'}]{a} \ := \ \xrightarrow[G]{a} \ \cap \ \overline{N^* {\times} F}$$

is a synchronized relation as the intersection of a synchronized relation with a recognizable relation (using Theorem 2.5 and Proposition 2.6). For all $a \in \mathcal{A}$, we denote

$$F_a \ := \ \mathrm{Dom}(\xrightarrow[G]{a} \ \cap \ N^* {\times} F)$$

the set of vertices which are source of an erased edge. This set is rational as a domain of a rational relation. Then we create new edges leading from those vertices to the vertex f. More precisely, we define the graph $\overline{G}$ such that for all $a \in \mathcal{A}$,

$$\xrightarrow[\overline{G}]{a} \ := \ \xrightarrow[G^{'}]{a} \ \cup \ F_a {\times} a {\times} \{f\}.$$

This relation is left-synchronized as the union of a left-synchronized relation with a recognizable set. Moreover and by construction,

$$L(G,I,F) \ = \ L(\overline{G},I,\{f\}).$$

ii) Denoting by $\widetilde{u}$ the mirror of $u \in \mathcal{A}^*$ and by G^{-1} the graph such that $p \xrightarrow[G]{a} q$ if and only if $q \xrightarrow[G^{-1}]{a} p$, we apply $i)$ in order to get a unique initial vertex :

$$L(\overline{G},I,\{f\}) \ = \ L(\widetilde{\overline{G}^{-1}},\{f\},I) \ \overset{(i)}{=} \ L(\widetilde{\overline{\overline{G}^{-1}}},\{f\},\{i\}) \ = \ L(\overline{\overline{G}^{-1}},\{i\},\{f\}).$$

□

There are different ways to characterize a context-sensitive language L. As Morvan and Stirling [MoS 01], we choose the 'left' form due to Penttonen [Pe 74].

Definition 3.2 A rewriting system $\Gamma = \Gamma_1 \cup \Gamma_2$ is a *2-system* if every rule of Γ_2 is of the form $AB \longrightarrow AC$ with $B \neq C$ and every rule of Γ_1 is of the form $A \longrightarrow a$ where A, B, C are letters of the non-terminal alphabet N and $a \in \mathcal{A}$.

Context-sensitive languages are obtained by derivation of a 2-system from a linear language.

Theorem 3.3 [Pe 74] *There exists a linear language* L_{Lin} *such that every context-sensitive language is* $\{v \in \mathcal{A}^* \mid \exists\, u \in L_{Lin}\,,\ u \xrightarrow[\Gamma]{*} v\}$ *for some 2-system* Γ.

Given a context-sensitive language L, we first look for a graph G_{Lin} such that $L = L(G_{Lin}, L_{Lin}, \{\varepsilon\})$. Let Γ be a 2-system. From Γ_2, we define the relation R_2 recognized by the following transducer T_2:

$$\begin{array}{rcll l}
I & \xrightarrow{[B/[A} & (A,B,A) & \text{for all } A,B \in N & \text{(type 1)} \\
(A,B,C) & \xrightarrow{B/D} & (A,B,D) & \text{for all } A,B,C,D \in N \text{ such that } BC \xrightarrow[\Gamma_2]{} BD & \text{(type 2)} \\
(A,B,C) & \xrightarrow{D/C} & (A,D,C) & \text{for all } A,B,C,D \in N & \text{(type 3)} \\
(A,B,C) & \xrightarrow{]A/]} & F & \text{for all } A,B,C \in N & \text{(type 4)}
\end{array}$$

This transducer starting at I and ending at F recognizes pairs of the form

$$([AA_1 \ldots A_m]B, [BB_1 \ldots B_m])$$

meaning that under the successive context $A, A_1, \ldots, A_m$ the letter B can be rewritten successively $B, B_1, \ldots, B_m$. If the context does not change: $A_i = A_{i+1}$, and one can apply a rule $A_iB_i \xrightarrow[\Gamma_2]{} A_{i+1}B_{i+1}$. Note that it is possible even if $B_i = B_{i+1}$ as a rule of type 3 can be applied with $B = D$. If the context changes: $A_i \neq A_{i+1}$, we copy the letter $B_i = B_{i+1}$.
Note that R_2 is a bounded length difference relation.

Example 3.4 Let $\Gamma_2 = \{(AB, AC)\,,\ (AC, AD)\,,\ (DA, DE)\,,\ (EA, EE)\}$.
We have $[AAA]B\ \ R_2\ \ [BCD]$ because under the context A, letter B can be rewritten to C and then to D. The following derivation:

$$ABAA \xrightarrow[\Gamma_2]{} ACAA \xrightarrow[\Gamma_2]{} ADAA \xrightarrow[\Gamma_2]{} ADEA \xrightarrow[\Gamma_2]{} ADEE$$

is represented as follows:

A	B	A	A
A	C	A	A
A	D	A	A
A	D	E	A
A	D	E	E

We have $[AAAAA]B\ \ R_2\ \ [BCDDD]$ and $[BCDDD]A\ \ R_2\ \ [AAAEE]$
and $[AAAEE]A\ \ R_2\ \ [AAAAE]$.

Let us give an elementary property of transducer T_2.

Lemma 3.5 *If* $I \overset{[UA/[BVC}{\underset{\Gamma_2}{\Longrightarrow}} s$ *with* $A, B, C \in N$ *and* $U, V \in N^*$ *then* $s = (B, A, C)$.

Proof. By induction on the length of any non-empty derivation from I.
□

Consider a word $X_1 \in L_{Lin}$ of size n and a derivation $X_1 \underset{\Gamma_2}{\longrightarrow} X_2 \underset{\Gamma_2}{\longrightarrow} \dots \underset{\Gamma_2}{\longrightarrow} X_m$ to a word X_m. Given the m successive letters at a position i according to the derivation, the transducer gives the m successive letters at position $i+1$.
For any words $X, Y \in N^*$ of the same length n, we denote by $X \triangle Y$ the cardinal of $\{\, 1 \leq i \leq n \mid X(i) \neq Y(i) \,\}$.

Lemma 3.6 *The two following properties are equivalent:*
a) $X_1 \underset{\Gamma_2}{\longrightarrow} X_2 \underset{\Gamma_2}{\longrightarrow} \dots \underset{\Gamma_2}{\longrightarrow} X_m$
b) $[X_1(i-1)X_2(i-1)\dots X_m(i-1)]X_1(i)\ R_2\ [X_1(i)\dots X_m(i)]$ *for all* $2 \leq i \leq |X_1|$;
$|X_{j-1}| = |X_j|$, $X_{j-1} \triangle X_j = 1$ *and* $X_{j-1}(1) = X_j(1)$ *for all* $2 \leq j \leq m$.

Proof. The words $X_1, \dots, X_m$ of same length n are represented as follows.

X_1	$X_1(1)$	$X_1(2)$	$\dots$	$X_1(i-1)$	$X_1(i)$	$\dots$	$X_1(n)$
X_2	$X_2(1)$	$X_2(2)$	$\dots$	$X_2(i-1)$	$X_2(i)$	$\dots$	$X_2(n)$
X_3	$X_3(1)$	$X_3(2)$	$\dots$	$X_3(i-1)$	$X_3(i)$	$\dots$	$X_3(n)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
X_j	$X_j(1)$	$X_j(2)$	$\dots$	$X_j(i-1)$	$X_j(i)$	$\dots$	$X_j(n)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
X_{m-1}	$X_{m-1}(1)$	$X_{m-1}(2)$	$\dots$	$X_{m-1}(i-1)$	$X_{m-1}(i)$	$\dots$	$X_{m-1}(n)$
X_m	$X_m(1)$	$X_m(2)$	$\dots$	$X_m(i-1)$	$X_m(i)$	$\dots$	$X_m(n)$

i) Let us show that (a) $\Longrightarrow$ (b).
By definition of Γ_2, we have, for all $2 \leq j \leq m$,

$$|X_{j-1}| = |X_j| \text{ and } X_{j-1} \triangle X_j = 1 \text{ and } X_{j-1}(1) = X_j(1)\,.$$

Let us show that

$$[X_1(i-1)X_2(i-1)\dots X_m(i-1)]X_1(i)\ R_2\ [X_1(i)\dots X_m(i)]$$

by induction on $m \geq 1$.

Basis case : $m = 1$. For all $2 \leq i \leq |X_1|$, we have

$$[X_1(i-1)]X_1(i) \; R_2 \; [X_1(i)]$$

considering the path

$$I \xrightarrow[T_2]{[X_1(i-1)/[X_1(i)} (X_1(i), X_1(i-1), X_1(i)) \xrightarrow[T_2]{]X_1(i)/]} F.$$

Inductive case : $m \Longrightarrow m+1$.

Suppose the implication for a derivation of length m and let $X_1 \xrightarrow[\Gamma_2]{} \ldots \xrightarrow[\Gamma_2]{} X_m \xrightarrow[\Gamma_2]{} X_{m+1}$. There exists $2 \leq k \leq |X_1|$ such that $X_m(k) \neq X_{m+1}(k)$ and for all $i \neq k$, $X_m(i) = X_{m+1}(i)$. Let $2 \leq i \leq |X_1|$. We want to show that

$$[X_1(i-1)\ldots X_m(i-1)X_{m+1}(i-1)]X_1(i) \;\; R_2 \;\; [X_1(i)\ldots X_{m+1}(i)].$$

By inductive hypothesis, we have

$$[X_1(i-1)\ldots X_m(i-1)]X_1(i) \;\; R_2 \;\; [X_1(i)\ldots X_m(i)].$$

Using Lemma 3.5, we have

$$I \overset{[X_1(i-1)\ldots X_m(i-1)/[X_1(i)\ldots X_m(i)}{\underset{T_2}{\Longrightarrow}} (X_1(i), X_m(i-1), X_m(i)).$$

We distinguish the two complementary cases below.

Case 1 : $i \neq k$. We add an edge of type 3.

$$\begin{aligned} (X_1(i), X_m(i-1), X_m(i)) \;&=\; (X_1(i), X_m(i-1), X_{m+1}(i)) \\ &\xrightarrow[T_2]{X_{m+1}(i-1)/X_{m+1}(i)} (X_1(i), X_{m+1}(i-1), X_{m+1}(i)). \end{aligned}$$

Case 2 : $i = k$. We have the rule $X_m(i-1)X_m(i) \;\; \Gamma_2 \;\; X_{m+1}(i-1)X_{m+1}(i)$.

To this rule is associated the following edge of type 2:

$$(X_1(i), X_m(i-1), X_m(i)) \xrightarrow[T_2]{X_{m+1}(i-1)/X_{m+1}(i)} (X_1(i), X_{m+1}(i-1), X_{m+1}(i)).$$

Finally, we add the edge leading to the final state:

$$(X_1(i), X_{m+1}(i-1), X_{m+1}(i)) \xrightarrow[T_2]{]X_1(i)/]} F.$$

We get the result for $m+1$ and the direct implication.

ii) Let us show that (b) $\Longrightarrow$ (a).

Suppose that $[X_1(i-1)\ldots X_m(i-1)]X_1(i) \;\; R_2 \;\; [X_1(i)\ldots X_m(i)]$ for all $2 \leq i \leq |X_1|$

and $|X_{j-1}| = |X_j|$ and $X_{j-1} \triangle X_j = 1$ and $X_1(j-1) = X_1(j)$ for all $2 \leq j \leq m$.
Let $2 \leq j \leq m$. Let us show that $X_{j-1} \underset{\Gamma_2}{\longrightarrow} X_j$.
As $X_{j-1} \triangle X_j = 1$, there exists a unique $2 \leq k \leq |X_1|$ such that $X_{j-1}(k) \neq X_j(k)$.
Moreover $X_{j-1}(1) = X_j(1)$ so $k \neq 1$ and $X_{j-1}(k-1) = X_j(k-1)$.
We have $[X_1(k-1)\ldots X_m(k-1)]X_1(k)\ R_2\ [X_1(k)\ldots X_m(k)]$.
Lemma 3.5 gives the existence of the following edge

$$(X_1(k), X_{j-1}(k-1), X_{j-1}(k)) \underset{T_2}{\xrightarrow{X_j(k-1)/X_j(k)}} (X_1(k), X_j(k-1), X_j(k)).$$

This edge is of type 2 and gives the existence of the following rule of Γ_2

$$X_{j-1}(k-1)X_{j-1}(k) \longrightarrow X_j(k-1)X_j(k).$$

□

Let L be a context-sensitive language obtained by derivation of a 2-system Γ from L_{Lin}. Adding to T_2 the set of edges $\{F \xrightarrow{A/A} F \mid A \in N\}$, we get a transducer recognizing a rational graph G_{Lin} of bounded length difference with edges of the form $[U]AW \to [AV]W$. If $X_1 \in L_{Lin}$ with $|X_1| = n$ and $X_1 \underset{\Gamma_2}{\overset{m-1}{\longrightarrow}} X_m$, the graph G_{Lin} contains the following path:

$$[X_1(1)^m]X_1(2)\ldots X_1(n) \to [X_1(2)\ldots X_m(2)]X_1(3)\ldots X_1(n) \ \ldots \to [X_1(n)\ldots X_m(n)]$$

If we add edges of the form $[U] \to \varepsilon$ for any word U and if we label edges of G such that $[U]AW \xrightarrow{a} [AV]W$ if the last letter of U can be derived to a according to Γ_1 then we get a left-synchronized graph G such that $L = L(G, L_{Lin}, \{\varepsilon\})$. The problem is that L_{Lin} is not rational. In order to reduce L_{Lin} to a rational set, we complete T_2 to a transducer generating words of L_{Lin} successively from left to right.
Let Gr be a grammar in Greibach normal form generating L_{Lin} from a non-terminal S. Each rule of Gr is of the form $Z \to AW$ where $Z \in N_r$ is a non-terminal of Gr, $A \in N$ is a terminal (which is also a non-terminal of Γ) and $W \in N_r^*$ is a non-terminal word of Gr. Let the transducer

$$T_2' := T_2 \cup \{F \xrightarrow{Z/U} F' \mid (Z, U) \in Gr\} \cup \{F' \xrightarrow{Z/Z} F' \mid Z \in N_r\}$$

where F' is a new state of the transducer. We denote by R_2' the relation recognized by T_2' from I to F'. This relation is still of bounded length difference. Let

$$L_{Rat} := \{\, [A^m]BW \mid S \underset{Gr}{\overset{2}{\longrightarrow}} ABW \ \wedge\ A, B \in N \ \wedge\ W \in N_r^* \ \wedge\ m \geq 1 \,\}.$$

Let us reformulate Lemma 3.6 for derivations starting from L_{Lin}.

Lemma 3.7 *Let* $X_1, \ldots, X_m \in \mathbb{N}^*$ *and* $n = |X_1|$.

The two following properties are equivalent:

a) $X_1 \underset{\Gamma_2}{\longrightarrow} X_2 \underset{\Gamma_2}{\longrightarrow} \ldots \underset{\Gamma_2}{\longrightarrow} X_m$ *and* $X_1 \in L_{Lin}$

b) *There exists* $W_1, \ldots, W_{n-1} \in N_\Gamma^*$ *such that*

$[X_1(1)\ldots X_m(1)]X_1(2)W_1 \in L_{Rat}$ *and* $W_{n-1} = \varepsilon$

and $[X_1(n-1)\ldots X_m(n-1)]X_1(n) \;\; R_2 \;\; [X_1(n)\ldots X_m(n)]$

and $[X_1(i-1)\ldots X_m(i-1)]X_1(i)W_{i-1} \;\; R_2' \;\; [X_1(i)\ldots X_m(i)]X_1(i+1)W_i$

for all $2 \le i < n$

and $|X_{j-1}| = |X_j|$ *and* $X_{j-1} \triangle X_j = 1$ *and* $X_{j-1}(1) = X_j(1)$ *for all* $2 \le j \le m$.

Proof. i) We suppose (a) and show (b).

As $X_1 \in L_{Lin}$, we consider the derivation from S to X_1 according to Gr: there exists non-terminal words $W_1, \ldots, W_{n-2}$ of Gr such that

$$S \underset{Gr}{\overset{2}{\longrightarrow}} X_1(1)X_1(2)W_1 \underset{Gr}{\longrightarrow} \ldots \underset{Gr}{\longrightarrow} X_1(1)\ldots X_1(n-1)W_{n-2} \underset{Gr}{\longrightarrow} X_1(1)\ldots X_1(n)$$

By Lemma 3.6, we have for all $2 \le i \le |X_1|$

$$[X_1(i-1)\ldots X_m(i-1)]X_1(i) \;\; R_2 \;\; [X_1(i)\ldots X_m(i)]$$

and $|X_{j-1}| = |X_j|$ and $X_{j-1} \triangle X_j = 1$ and $X_{j-1}(1) = X_j(1)$ for all $2 \le j \le m$.

Let $2 \le i \le n-1$. We know that W_i is obtained from W_{i-1} by the rewriting of the non-terminal $W_{i-1}(1)$:

$$W_{i-1} \; = \; ZV \underset{Gr}{\longrightarrow} UV \; = \; X_2(i+1)W_i.$$

We complete the preceeding path leading to F with the edge $F \overset{Z/U}{\longrightarrow} F'$ and then with edges $F' \underset{T_2'}{\overset{Z/Z}{\longrightarrow}} F'$ for V. Thus, we have

$$[X_1(i-1)\ldots X_m(i-1)]X_1(i)W_{i-1} \;\; R_2' \;\; [X_1(i)\ldots X_m(i)]X_1(i+1)W_i\,.$$

ii) We suppose (b) and show (a).

We cut the paths

$$[X_1(i-1)\ldots X_m(i-1)]X_1(i)W_{i-1} \;\; R_2' \;\; [X_1(i)\ldots X_m(i)]X_1(i+1)W_i$$

which become

$$[X_1(i-1)X_2(i-1)\ldots X_m(i-1)]X_1(i) \;\; R_2 \;\; [X_1(i)\ldots X_m(i)]\,.$$

By Lemma 3.6, we have $X_1 \underset{\Gamma_2}{\longrightarrow} X_2 \underset{\Gamma_2}{\longrightarrow} \ldots \underset{\Gamma_2}{\longrightarrow} X_m$.

By hypothesis $[X_1(1)\ldots X_m(1)]X_1(2)W_1 \in L_{Rat}$ and $X_1(1) = \ldots = X_m(1)$. So

$S \xrightarrow[Gr]{2} X_1(1)X_1(2)W_1$. Thus $S \xrightarrow[Gr]{*} X_1(1)\ldots X_1(n) = X_1$ hence $X_1 \in L_{Lin}$.
□

The transducer T_2' successively generates letters of X_1. Let us construct a graph of bounded length difference such that the language of path labels leading from the rational vertex set L_{Rat} to a rational vertex set F_{Rat} is the context-sensitive language defined by Γ.

> **Proposition 3.8** *Any context-sensitive language is the language* $L(G, L_{Rat}, F_{Rat})$ *of path labels leading from a rational set of vertices* L_{Rat} *to another* F_{Rat} *and where* G *is a graph of bounded length difference.*

Proof. Let L be a context-sensitive language. There exists a 2-system Γ such that

$$L = \{ v \in \mathcal{A}^* \mid \exists\, u \in L_{Lin},\ u \xrightarrow[\Gamma]{*} v \}.$$

For all letter $a \in \mathcal{A}$, we denote by

$$N_a := \{ A \in N \mid A \xrightarrow[\Gamma_1]{} a \}$$

the set of non-terminals generating the terminal a in Γ.
We define the graph G_0 such that for any $a \in \mathcal{A}$,

$$\xrightarrow[G_0]{a} := R_2' \cap [N^*N_a]NN_r^* \times ([N^+]NN_r^* \cup [N^+]) .$$

As R_2' is a bounded length difference relation, so G_0 is and the following graph:

$$G := G_0 \cup \bigcup_{a \in \mathcal{A}} \{ [UA] \xrightarrow{a} [UA]\$ \mid U \in N^* \wedge A \in N_a \}$$

is also of bounded length difference.
We recall that

$$L_{Rat} := \{ [A^m]BW \mid S \xrightarrow[Gr]{2} ABW \wedge m \geq 1 \}$$

where S is the axiom of Gr and let

$$F_{Rat} := [N^*]\$.$$

We have

$$u \in L \text{ with } |u| = n > 1$$

$$\Longleftrightarrow$$

there exists $X_1, \ldots, X_m \in N^*$ of length n such that

$$X_1 \in L_{Lin} \text{ and } X_1 \underset{\Gamma_2}{\longrightarrow} X_2 \underset{\Gamma_2}{\longrightarrow} \ldots \underset{\Gamma_2}{\longrightarrow} X_m \text{ and } X_m(i) \underset{\Gamma_1}{\longrightarrow} u(i) \text{ for all } 1 \leq i \leq n$$

$$\Longleftrightarrow \text{(by Lemma 3.7)}$$

$$\text{there exists non-terminal words } W_1, \ldots, W_{n-1} \text{ of } Gr \text{ such that}$$

$$[X_1(1)\ldots X_m(1)]X_1(2)W_1 \in I_{Rat}\ , W_{n-1} = \varepsilon$$

$$[X_1(1)\ldots X_m(1)]X_1(2)W_1 \xrightarrow[G_0]{u(1)} [X_1(2)\ldots X_m(2)]X_1(3)W_2 \xrightarrow[G_0]{u(2)} \ldots \xrightarrow[G_0]{u(n-1)} [X_1(n)\ldots X_m(n)] \text{ and}$$

$$X_m(n) \in N_{u(n)}$$

$$\Longleftrightarrow$$

$$u \in L(G, L_{Rat}, F_{Rat})$$

Thus

$$L \;=\; L(G, L_{Rat}, F_{Rat}) \;\cup\; \{\, u \in L \mid |u| \leq 1 \,\}$$

□

It remains to apply Lemma 3.1 to get the following proposition:

Proposition 3.9 *Any context-sensitive language is trace of a synchronized graph.*

Proof. Any synchronized graph is a rational graph, hence any trace of a synchronized graph is a context-sensitive language [MoS 01]. Proposition 3.9 gives the converse.

Theorem 3.10 *The context-sensitive languages are the traces of synchronized graphs.*

Moreover, using Lemma 3.1, we get that any language $L(G, L_{Rat}, F_{Rat})$ of path labels leading from and to a rational vertex set of a graph G of bounded length difference is a context-sensitive language as the trace of a synchronized (thus rational) graph. Proposition 3.8 gives the converse.

Theorem 3.11 *The context-sensitive languages are the languages* $L(G, L_{Rat}, F_{Rat})$ *of path labels leading from and to a rational vertex set of a graph* G *of bounded length difference.*

The synchronized relation of bounded length difference R_2' we used in the proof of Proposition 3.8 can be completed into a letter-to-letter relation.

Lemma 3.12 *Let* $R \subseteq N^* \times N^*$ *be a left-synchronized relation and let* $\diamond$ *be a symbol such that* $\diamond \notin N$. *We can transform* R *into a letter-to-letter relation* R_l *such that*
$\forall (U, V) \in N^* \times N^*, \forall n \geq 0,$

$$(U \xrightarrow[R]{n} V) \iff (\exists k \geq 0, \exists k' \geq 0 \text{ such that } U\diamond^k \xrightarrow[R_l]{n} V\diamond^{k'})$$

Proof. Let T be a left-synchronized transducer recognizing R. We construct the transducer T_l from T replacing each edge of the form $p \xrightarrow{\epsilon/A} q$ (respectively $p \xrightarrow{A/\epsilon} q$) with $A \in N$ by the edge $p \xrightarrow{\diamond/A} q$ (respectively $p \xrightarrow{A/\diamond} q$). Then for each final vertex f of T, create a new final state f' of T_l and add the edges $f \xrightarrow{\diamond/\diamond} f'$ and $f' \xrightarrow{\diamond/\diamond} f'$.
□

Proposition 3.13 *Any context-sensitive language is the language* $L(G, L_{Rat}, F_{Rat})$ *of path labels leading from a rational set of vertices* L_{Rat} *to another* F_{Rat} *and where* G *is a letter-to-letter rational graph.*

Proof. Using Proposition 2.6 we get that R_2' is a left-synchronized relation. Let $\diamond$ be a symbol such that $\diamond \notin N \cup N_r$. Using Lemma 3.12, we complete R_2' into a letter-to-letter relation R_l. We get the result adapting the proof of Proposition 3.8 with

$$\xrightarrow[G_0]{a} := R_l \cap [N^* N_a] N N_r^* \diamond^* \times ([N^+] N N_r^* \diamond^* \cup [N^+] \diamond^*)$$

$$G := G_0 \cup \bigcup_{a \in \mathcal{A}} \{ [UA]\diamond^k \xrightarrow{a} \$^{|[UA]|+k} \mid U \in N^* \wedge A \in N_a \}$$

$$L_{Rat} := \{ [A^m] B W \diamond^k \mid S \xrightarrow[Gr]{2} ABW \wedge m \geq 1 \wedge k \geq 0 \}$$

and

$$F_{Rat} := \$^+$$

□

The converse is given by Theorem 3.11.

Theorem 3.14 *The context-sensitive languages are the languages* $L(G, L_{Rat}, F_{Rat})$ *of path labels leading from and to a rational vertex set of a letter-to-letter rational graph* G.

4 Conclusion

Since synchronized binary relations form a boolean algebra and are recognized by deterministic 2-automata, the consideration of context-sensitive languages as traces of synchronized graphs could help for the conjecture of determinism of context-sensitive languages [Ku 64]: does any context-sensitive language can be recognized by a deterministic linear bounded Turing machine ? The characterization of context-sensitive langages using rational letter-to-letter graphs could also be useful to solve this problem as every connex component of a rational letter-to-letter graph is a finite graph. In [Car 01] Arnaud Carayol considers globally deterministic sets of transducers (i.e. in a case of non-determinism, only one output produced is accepted). He shows that the traces of those graphs with rational initial vertex sets are deterministic context-sensitive langages. His proof suggests that we could have worked directly on LBA Turing machin instead of using Pentonnen form.

Acknowledgements

Many thanks to Didier Caucal for his support and his help during the redaction of this paper.

References

[BG 00] A. BLUMENSATH and E. GRÄDEL *Automatic structures*, LICS 15th, 51–62 (2000).

[Car 01] A. CARAYOL *A characterization of context-sensitive langages*, report (2001).

[Cau 96] D. CAUCAL *On infinite transition graphs having a decidable monadic theory*, ICALP 23rd, LNCS 1099, F. Meyer auf der Heide, B. Monien (Eds.), 194–205 (1996) [a full version will appear in Theoretical Computer Science].

[Cau 01] D. CAUCAL *On transition graphs of Turing machines*, MCU 3rd, LNCS 2055, M. Margenstern, Y.Rogozhin (Eds.), 177–189 (2001).

[Co 90] B. COURCELLE *Graph rewriting: an algebraic and logic approach*, Handbook of Theoretical Computer Science, Volume B, 193–242 (1990).

[EM 65] C. ELGOT and J. MEZEI *On relations defined by generalized finite automata*, IBM J. Res. Dev. 9, 47–68 (1965).

[FS 93] C. FROUGNY and J. SAKAROVITCH *Synchronized rational relations of finite and infinite words*, Theoretical Computer Science 108, 45–82 (1993).

[Ku 64] S.-Y. KURODA *Classes of languages and linear-bounded automata*, Information and Control 7, 207–223 (1964).

[Mo 00] C. MORVAN *On rational graphs*, FOSSACS 3rd, Berlin, J. Tiuryn (Ed.), LNCS 1784, 252–266 (2000).

[MoS 01] C. MORVAN and C. STIRLING *Rational graphs trace context-sensitive langages*, MFCS 26th, Mariánské Lázně, A. Pultr and J. Sgall (Eds.), LNCS, (2001).

[MS 85] D. MULLER and P. SCHUPP *The theory of ends, pushdown automata, and second-order logic*, TCS 37, 51–75 (1985).

[Pe 74] PENTTONEN *One-sided and two-sided context in formal grammars*, Information and Control 25, 371–392 (1974).

[Ra 69] M.O. RABIN *Decidability of second order-theories and automata on finite trees*, Trans. Amer. Math. Soc 141, 1–35 (1969).

[RS 59] M. RABIN and D. SCOTT *Finite automata and their decision problems*, IBM J. Res. Dev. 3, 125–144 (1959) ; republished: E. Moore (Ed.), *sequential machines*, Addison-Wesley (1965).

Abstract Families of Graphs

Tanguy URVOY

turvoy@irisa.fr

IRISA, Campus de Beaulieu,

35042 Rennes, France

Topics: infinite state processes, automata and formal languages.

Abstract

A natural way to describe a family of languages is to use rational transformations from a generator. Infinite graphs are natural models for verification. We study families of infinite graphs that are described from generators by transformations preserving monadic second order logic decision. We call them Abstract Family of Graphs (AFG). We make a link with language theory by showing that traces of AFG are rational cones and traces of principal AFG are AFL. We generalize some properties of prefix recognizable graphs to AFG and we apply these tools and the notion of geometrical complexity to study sub-families of prefix recognizable graphs (REC). We exibit a strictly increasing chain of AFG in REC.

Introduction

A large number of families of formal languages arising from either computer science or linguistics were investigated for their properties. These families first described with sophisticated acceptors, were also described by combinatorial or algebraic properties. A well known example is the Chomsky-Schützenberger characterization of context-free languages as rational transductions from the Dyck language [7]. A lot of language families were found to be closed by rational transductions and some other operations like union or concatenation. From these closure operations, Ginsburg and Greibach have derived the Abstract Family of Languages (AFL) [11]. In the same paper, they have defined the Abstract Family of Acceptors (AFA). They showed the direct relation between language families and machine families. The notion of AFA fell into disuse but the one of AFL was intensively studied and refined. Many structures defined by closure operations like rational cones, or cylinders were also considered (See [2] or [1] for a survey).

The development of model checking rised up a new interest for behavioral properties of machines. A main task of model checking is to decide if a given machine modelized by a (finite or infinite) graph satisfy or not a given logical sentence. This problem is unsolvable in general but Muller and Shupp showed that Monadic Second Order (MSO) logic, is decidable for the transition graphs of pushdown automata [15]. This result has been extended to HR-equational graphs by Courcelle [8] and to prefix recognizable graphs by Caucal [5]. Different characterizations of these graph families are summarized in [3].

Verification of logical properties is not the only interest of graph families: they also provide a natural tool to study formal languages. A trace of a graph is the language of path labels from and to finite sets of vertices. A main graph hierarchy is constructed in parallel of the Chomsky hierarchy: the traces of finite graphs are the rational languages, the traces of prefix recognizable graphs are the context-free languages [5], the traces of rational graphs are the context-sensitive languages [14] and the traces of Turing graphs are the recursively enumerable languages [6]. A survey about this hierarchy is done in [16].

This present paper studies the relation between languages families and graphs families. In [6], the family of prefix recognizable graphs is generated from the Dyck graph Δ_2 (See Figure 4) with two closure operations: rational coloring and inverse rational substitution. We study these operators in a more general context: Given a graph Γ, we consider the family of graphs generated from Γ by rational coloring, inverse rational substitution and product with a finite graph. We call Abstract Family of Graphs (AFG) a graph family that is closed by these operators. We show that traces of AFG are rational cones, we also show that traces of principal AFG are AFL. We show that geometrical complexity of graphs is preserved by AFG transformations: this is a criterion to compare Graph families. We apply these tools to sub-families of prefix recognizable graphs like linear or polynomial graphs. We use geometrical complexity to show that polynomial graphs constitute a strictly increasing chain of AFG.

This paper is divided into three sections. In Section 1 some facts about languages families are reviewed. In Section 2 we define generic graph families and we study their properties. In section 3 we apply the concepts of Section 2 to sub-families of prefix recognizable graphs. We exhibit a strictly increasing chain of AFG.

1 Languages Families

The reader is supposed to be familiar with the basic notions of formal languages described in [12]. We recall here briefly some notions about language families. See [1], [2] or [10] for a complete reference.

1.1 Language Transformations

A monoid *morphism* f from X^* to Y^* is a mapping which associates a word of Y^* to each letter of X. It is extended to words by morphism: $f(\varepsilon) = \varepsilon$ and $f(au) = f(a)f(u)\ \forall a \in X, u \in X^*$. When $f^{-1}(\varepsilon) = \{\varepsilon\}$, f is called *non-erasing*; when $f(X) \subseteq Y \cup \{\varepsilon\}$, f is called *alphabetic*. The *projection* or *erasing* $p_a : X^* \longrightarrow Y^*$ is the alphabetic morphism defined by $p_a(a) = \varepsilon$ and $p_a(b) = b\ \forall b \neq a$. The *copy* $c : X^* \longrightarrow Y^*$ is an alphabetic morphism defined by an injection from X to Y. A *rational substitution* of domain X is a mapping which associates a rational language to each letter of X. A *finite substitution* is a rational substitution which associate a finite language to each letter. Substitutions are extended to words by morphism rules.

We use the notation $\mathrm{Fin}(M)$ to design the finite subsets of a set M. If M is a monoid, we use the notation $\mathrm{Rat}(M)$ to design its rational subsets. A *rational transduction* (or simply *transduction*) between two monoids X^* and Y^* is a relation $T \in \mathrm{Rat}(X^* \times Y^*)$; it is *faithful* or *non-erasing* if for any $y \in Y^*$, $T^{-1}(y) \in \mathrm{Fin}(X^*)$. A non-erasing transduction is the composition of an inverse morphism, an intersection with a rational set and a non-erasing morphism. A transduction is the composition of a non-erasing transduction and a morphism. Note that inverse rational (resp. finite) substitutions are particular cases of transductions (resp. non-erasing transductions).

1.2 AFL and Rational Cones

A *family of languages* is a nonempty set of languages that is closed under copy. A *cone* is a family of languages that is closed by transduction. A cone C is *principal* of *generator* Λ if $\Lambda \in C$ and if any L in C is the image of Λ by a transduction. If we adopt the notation of [10], we write $C = \mathcal{C}(\Lambda)$. The relation $L \preceq L'$ when L is the image by a transduction of L' is a way to mesure the “complexity” of a language, in that sense the generators of C are its most complex languages. Consider the Dyck language $D_2 = aD_2\overline{a}D_2 + bD_2\overline{b}D_2 + \varepsilon$, which is the language of “well formed” words over two types of parentheses. The Chomsky-Schützenberger theorem [7] gives a characterization of context-free languages as being the principal cone $\mathrm{Alg} = \mathcal{C}(D_2)$.

There are different types of language families classified by closure properties: Figure 1 gives a summary of these properties for cone and AFL. All families of the Chomsky hierarchy except context sensitive languages are full-AFL. The family Ocl of one-counter languages is a principal full-AFL. The family Lin of linear languages and the family Rocl of restricted one counter languages are example of principal cones. The intersection of two principal cones is trivialy a cone, but showing the principality of this intersection is a difficult problem. It was

types	closure properties					
	inverse morphism	rational $\cap$	non-erasing morphism	erasing	$\cup$	L^+
semi-cone	×	×	×			
cone	×	×	×	×		
AFL	×	×	×		×	×
full-AFL	×	×	×	×	×	×

Figure 1: Different types of languages families.

conjectured for a long time that $\mathrm{Lin} \cap \mathrm{Rocl} = \mathcal{C}(S)$ with $S = \{a^n b^n \mid n \geq 1\}$. This conjecture has been shown to be false in [4] by considering acceptor's structure.

2 Graphs Families

2.1 Preliminaries on graphs

Given an arbitrary set of vertices V, and an arbitrary set of symbols Σ, a *graph* G is a subset of $V \times \Sigma \times V$ where $\Sigma_G := \{a \in \Sigma \mid (s,a,t) \in G\}$ is finite and $V_G := \{s \in V \mid \exists t, (s,a,t) \in G \vee (t,a,s) \in G\}$ is countable. The existence of an arc (s,a,t) of *source* s and of *goal* t in G is denoted $s \xrightarrow[G]{a} t$, or simply $s \xrightarrow{a} t$ when G is understood. We write $s \overset{w}{\Longrightarrow} t$ with a word $w = w_1 \ldots w_n \in \Sigma_G^*$ the existence of a path $s \xrightarrow{w_1} s_1 \ldots s_{n-1} \xrightarrow{w_n} t$ in G.

When $I \in \mathrm{Fin}(V_G)$ and $F \in \mathrm{Fin}(V_G)$, the language $L(G,I,F) = \{w \in \Sigma_G^* \mid \exists i \in I, f \in F,\ i \overset{w}{\underset{G}{\Longrightarrow}} f\}$ is called a *trace* of G. With a special symbol $\tau \in \Sigma_G$, we may also consider the trace with ε-transitions $p_\tau(L(G,I,F))$. A graph G is deterministic if for all $s,t,t' \in V_G$ we have $s \xrightarrow[G]{a} t \wedge s \xrightarrow[G]{a} t' \Rightarrow t = t'$. It is *strongly connected* when for any $s,t \in V_G$ there is a word w such that $s \overset{w}{\underset{G}{\Longrightarrow}} t$, it is simply *connected* if $G \cup G^{-1}$ is *strongly connected* ($G^{-1} = \{(t,a,s) \mid s \xrightarrow[G]{a} t\}$). The distance between two vertices in a connected graph G is the function defined by $d_G(s,t) = \min\{|w| \mid s \overset{w}{\underset{G \cup G^{-1}}{\Longrightarrow}} t\}$. The degree of a vertex s is $d_G(s) = |\{(s,a,t) \in G \cup G^{-1}\ a \in \Sigma_G \wedge t \in V_G\}|$.

2.2 Graph Transformations

The most general aproach to transform a graph by preserving MSO decidability is to use all MSO-interpretations [9]. Our approach is different since we want to define graphs families in relation with languages families: we want a reduced set of transformations corresponding to languages transformations. Given a graph G, a transformations T is *internal* if $V_{T(G)} \subseteq V_G$.

2.2.1 Internal Transformations

copy: Given a bijection c between two alphabets, the *copy* by c of a graph G is the graph $c(G) = \{(s, c(a), t) \mid s \xrightarrow[G]{a} t\}$.

Rational Coloring: Let G be a graph and let f be a substitution from Σ to $Rat(\Sigma_G^*)$. The *coloring* of G from the source $V \in Fin(V_G)$ according to f is the graph $\#_{V,f}G = G \cup \{(t, a, t) \mid \exists w \in f(a), v \in V,\ v \overset{w}{\underset{G}{\Longrightarrow}} t\}$. To enlight notations we write $\#_f G$ when no confusion is possible about V.

Inverse Substitution: Let G be a graph and f be a substitution from an alphabet Σ to $2^{\Sigma_G^*}$. The *inverse substitution of* G *according to* f is the graph $f^{-1}(G) = \{(s, a, t) \mid \exists w \in f(a), s \overset{w}{\underset{G}{\Longrightarrow}} t\}$.

ε-closure: The ε-closure is a particular inverse substitution. Let G be a graph with a special symbol τ for ε-transitions and let # be a transparent color. Consider the rational substitution ϕ from $\Sigma_G - \{\tau\}$ to $Rat((\Sigma_G + \#)^*)$ defined by $\phi(a) = \#\tau^* a \tau^* \#$ for all $a \in \Sigma_G - \{\tau, \#\}$. We call *epsilon closure* of G the graph $\overline{G} = \phi^{-1}(G)$. Let p_τ be the erasing of τ. If f is the mapping defined by $f(\#) = \{\varepsilon\}$ and if $\varepsilon \notin p_\tau(L(G, I, F))$ then the trace with ε-transition $p_\tau(L(G, I, F))$ is $L(\overline{f^{-1}(G)}, I, F)$.

2.2.2 Other Transformations

Isomorphism If f is a mapping of domain V_G then $f(G) = \{(f(s), a, f(t)) \mid s \xrightarrow[G]{a} t\}$. A *graph isomorphism* between G and H is an injective mapping $f : V_G \longrightarrow V_H$ such that $f(G) = H$. The graphs G and H are isomorphic and we write $G \sim H$ when there is a graph isomorphism between G and H.

Products: To study the intersection between the traces of two graphs, it may be useful to consider the *synchronized product* of these graphs but the *concatenation* and *desynchronized product* are more easy to handle. Given graphs G and H, we define products between G and H as subsets of $(V_G \times V_H) \times \Sigma_G \cup \Sigma_H \times (V_G \times V_H)$. Let v be a vertex of H. The *concatenation* of H to G at v is the graph $G \cdot_v H = \{((s, s'), a, (t, t')) \mid (s \xrightarrow[G]{a} t \wedge s' = t' = v) \vee (s = t \wedge s' \xrightarrow[H]{a} t')\}$. The *desynchronized product* of G by H is the graph $G \square H = \{((s, s'), a, (t, t')) \mid (s \xrightarrow[G]{a} t \wedge$

$s' = t') \vee (s' \xrightarrow[H]{a} t' \wedge s = t)\}$. The *synchronized product* of G by H is the graph $G \times H = \{((s,s'),a,(t,t')) \mid s \xrightarrow[G]{a} t \ \wedge\ s' \xrightarrow[H]{a} t'\}$.

2.3 Abstract Graphs Families

An *internal graph family* is a non-empty set of graphs that is closed by copy. A *graph family* is an internal graph family that is closed by isomorphism. Some operations like union or intersection of graphs may be well defined in an internal family but does not make sense when considered up to isomorphism.

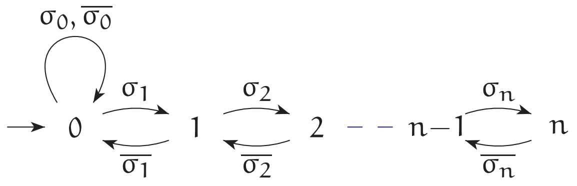
2.3.1 Closure Operators

Each graph transformation considered in Section 2.1 is naturally extended to graph families: Let $\mathcal{F}$ be a graph family, we denote respectively by $\#\mathcal{F}$, $\mathrm{Fin}^{-1}(\mathcal{F})$, $\mathrm{Rat}^{-1}(\mathcal{F})$ and $\overline{\mathcal{F}}$ the closures by rational coloring, inverse finite substitution, inverse rational substitution and ε-closure of $\mathcal{F}$. We denote by Fin the family of finite graphs; We denote by $\mathcal{F} \cdot \mathrm{Fin}$,$\mathcal{F} \square \mathrm{Fin}$ and $\mathcal{F} \times \mathrm{Fin}$ the smallest family of graphs containing $\mathcal{F}$ that is respectively closed by concatenation, desynchronized product and synchronized product with a finite graph.

2.3.2 Finite Product Equivalence

We consider a particular family of finite graphs for concatenation and show the equivalence of finite products. Let $n \in \mathbb{N}$ and the alphabet $\Sigma_n = \Sigma_{n,+} \cup \Sigma_{n,-}$ where $\Sigma_{n,+} = \{\sigma_0, \ldots, \sigma_n\}$ and $\Sigma_{n,-} = \{\overline{\sigma_i} \mid \sigma_i \in \Sigma_{n,+}\}$. We extend the $\overline{a}$ bijection to words by the rules $\overline{a \cdot w} = \overline{w} \cdot \overline{a}$ and $\overline{\overline{a}} = a$. The *segment* $\Delta_0^n \subset \mathbb{N} \times \Sigma_n \times \mathbb{N}$ is the graph

$$\Delta_0^n = \{(i-1, \sigma_i, i) \mid 1 \leq i \leq n\} \cup \{(i, \overline{\sigma_i}, i-1) \mid 1 \leq i \leq n\} \cup \{(0, \sigma_0, 0), (0, \overline{\sigma_0}, 0)\}$$



This graph is deterministic and strongly connected, for any $0 \leq i, j \leq n$, it verifies: $i \overset{\sigma_i \ldots \sigma_j}{\underset{\Delta_0^n}{\Longrightarrow}} j$ and $j \overset{\overline{\sigma_i \ldots \sigma_j}}{\underset{\Delta_0^n}{\Longrightarrow}} i$. When $\mathcal{F}$ is a graph family, we denote by $\mathcal{F} \cdot_0 \Delta_0^n$ the smallest family of graphs containing $\mathcal{F}$ that is closed under concatenation with Δ_0^n at 0. An illustration of the segment concatenation is given in Figure 2.

Property 1 *If $\mathcal{F} = \mathrm{Fin}^{-1}(\mathcal{F})$ these four statements are equivalents:*

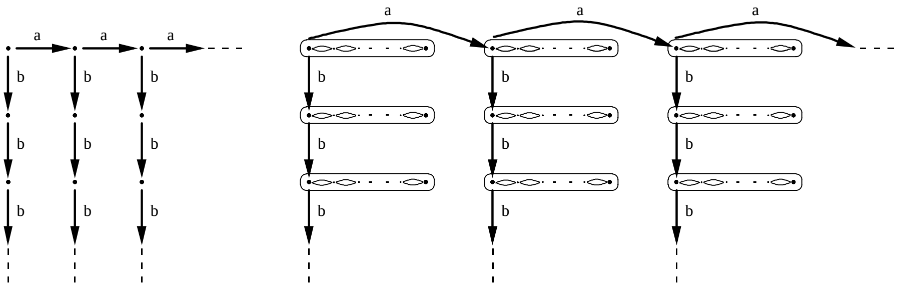


Figure 2: The comb tree concatenation with Δ_0^n

1. $\mathcal{F} = \mathcal{F} \cdot_0 \Delta_0^n \ \forall n \in I\!N$;

2. $\mathcal{F} = \mathcal{F} \cdot Fin$;

3. $\mathcal{F} = \mathcal{F} \square Fin$;

4. $\mathcal{F} = \mathcal{F} \times Fin$.

PROOF.

$(1 \Rightarrow 2)$ Let us show that $\mathcal{F} \cdot Fin \subseteq \mathcal{F}$.

Let $G \in \mathcal{F}$ and H be a finite graph of vertices $V_H = \{s_0, \ldots, s_n\}$. Let $c(G)$ be a copy of G such that $c(\Sigma_G) \cap \Sigma_n = \emptyset$. We have $G \cdot_{s_0} H \sim \{((s,0), a, (t,0)) \mid s \xrightarrow[G]{a} t\} \cup \{((s,i), a, (s,j)) \mid s \in V_G \ \wedge \ s_i \xrightarrow[H]{a} s_j\}$ hence $G \cdot_{s_0} H \sim h^{-1}(c(G) \cdot_0 \Delta_0^n) \in \mathcal{F}$ with $h : \ \Sigma_G \cup \Sigma_H \ \longrightarrow \ c(\Sigma_G) \cup Fin(\Sigma_n^*)$ defined by

$$h(a) = c(a) + \sum_{s_i \xrightarrow[H]{a} s_j} \overline{\sigma_0 \ldots \sigma_i} \, \sigma_0 \ldots \sigma_j$$

$(2 \Rightarrow 3)$ Let us show that $\mathcal{F} \square Fin \subseteq \mathcal{F}$.

Let $G \in \mathcal{F}$ and H be a finite graph of vertices $V_H = \{s_0, \ldots, s_n\}$. Let $c(G)$ be a copy of G such that $c(\Sigma_G) \cap \Sigma_n = \emptyset$.

$$\begin{gathered} c(G) \square H \sim \\ \{((s,i), c(a), (t,i)) \mid s \xrightarrow[G]{a} t \ \wedge \ s_i \in V_H\} \cup \{((s,i), a, (s,j)) \mid s \in V_G \ \wedge \ s_i \xrightarrow[H]{a} s_j\} \end{gathered}$$

we have $G \square H \sim h^{-1}(c(G) \cdot_0 \Delta_0^n) \in \mathcal{F}$ with h defined by

$$h(a) = \sum_{i=0}^{n} \overline{\sigma_0 \ldots \sigma_i} \, c(a) \ \sigma_0 \ldots \sigma_i \ + \sum_{s_i \xrightarrow[H]{a} s_j} \overline{\sigma_0 \ldots \sigma_i} \, \sigma_0 \ldots \sigma_j$$

$(3 \Rightarrow 4)$ Let us show that $\mathcal{F} \times Fin \subseteq \mathcal{F}$.

Let $G \in \mathcal{F}$ and H be a finite graph. Let $c(G)$ be a copy of G such that $c(\Sigma_G) \cap \Sigma_H = \emptyset$. We have $G \times H = h^{-1}(c(G)\Box H) \in \mathcal{F}$ with $h(a) = c(a)a \;\; \forall a \in \Sigma_G \cap \Sigma_H$ because

$$h^{-1}(c(G)\Box H) = \{((s,s'),a,(t,t')) \mid (s,s') \xrightarrow[c(G)\Box H]{c(a)} (t,s') \wedge (t,s') \xrightarrow[c(G)\Box H]{a} (t,t')\}$$

$(4 \Rightarrow 1)$ Let us show that $\forall n \in I\!N,\ \mathcal{F} \cdot_0 \Delta_0^n \subseteq \mathcal{F}$.
Let $G \in \mathcal{F}$ and $n \in I\!N$. Let $c(G)$ be a copy of G such that $c(\Sigma_G) \cap \Sigma_n = \emptyset$. We define the projection $p: \; c(\Sigma_G) \cup \Sigma_n \longrightarrow c(\Sigma_G) \cup \{\varepsilon\}$ by $p(a) = a \;\; \forall a \in c(\Sigma_G)$ and $p(\sigma) = \varepsilon \;\; \forall \sigma \in \Sigma_n$ We also define g by $g(a) = \sigma_0 \;\; \forall a \in c(\Sigma_G)$ and $g(\sigma) = \sigma \;\; \forall \sigma \in \Sigma_n$. Then we have $c(G) \cdot_{s_0} \Delta_0^n = p^{-1}(c(G)) \times g^{-1}(\Delta_0^n)$. We then project $c(\Sigma_G) \cup \Sigma_n$ to $\Sigma_G \cup \Sigma_n$: $G \cdot_{s_0} \Delta_0^n = h^{-1}(c(G) \cdot_{s_0} \Delta_0^n) \in \mathcal{F}$ with $h(a) = c(a) \;\; \forall a \in \Sigma_G$ and $h(\sigma) = \sigma \;\; \forall \sigma \in \Sigma_n$.

□

2.3.3 Abstract Families

As for languages, we define different types of families by their closure properties: Figure 3 summarize these definitions. The Rat^{-1} operator does not appear in Figure 3, the following property explain this disparition.

	closure properties			
types	$\mathcal{F} = \#\mathcal{F}$	$\mathcal{F} = Fin^{-1}(\mathcal{F})$	$\mathcal{F} = \overline{\mathcal{F}}$	$\mathcal{F} = \mathcal{F} \cdot \Delta_0^n$
AFG	×	×		×
full-AFG	×	×	×	×

Figure 3: Different types of graphs families.

Property 2 *If* $\mathcal{F} = \mathcal{F} \cdot Fin$ *then* $\overline{Fin^{-1}(\mathcal{F})} = Rat^{-1}(\mathcal{F})$

PROOF.
$\subseteq$
Let G be a graph of $\overline{Fin^{-1}(\mathcal{F})}$. Let $H \in \mathcal{F}$ and let f be a mapping from $\Sigma_G \cup \{\tau, \#\}$ to $Fin(\Sigma_H^*)$ such that $G = \overline{f^{-1}(H)}$. Recall that ϕ is defined by $\phi(a) = \#\tau^* a \tau^* \#$ for any $a \in \Sigma_G$, the composition of f by ϕ is a mapping from Σ_G to $Rat(\Sigma_H^*)$ hence $G = \phi^{-1}(f^{-1}(H))$ is in $Rat^{-1}(\mathcal{F})$.
$\supseteq$
Let G be a graph of $Rat^{-1}(\mathcal{F})$. Let $H \in \mathcal{F}$ and let f be a mapping from Σ_G to $Rat(\Sigma_H^*)$ such that $G = f^{-1}(H)$. We suppose that $\tau, \# \notin \Sigma_H$ and $\Sigma_H \cap \Sigma_n = \emptyset$ for any $n \in I\!N$ (we may take a copy of H in $\mathcal{F}$). Let $\mathcal{A} \subseteq Q \times \Sigma_H \times Q$ be a finite automaton such that for each letter $a \in \Sigma_G$ there is an initial state i_a and a final set F_a with $L(\mathcal{A}, \{i_a\}, F_a) = f(a)$. Let $n = |Q|$, and N be

a bijection between Q and $\{1, \ldots, n\}$. Let us consider the graph $K = H \cdot_0 \Delta_0^n$. We define the mapping g from $\Sigma_G \cup \{\tau, \#\}$ to $\mathrm{Fin}(\Sigma_K^*)$ by $g(\#) = \sigma_0$, $g(a) = \sigma_0 \ldots \sigma_{N(i_a)}$ $\forall a \in \Sigma_G$ and

$$g(\tau) = \sum_{a \in \Sigma_G, q \in F_a} \overline{\sigma_0 \ldots \sigma_{N(q)}} + \sum_{p \xrightarrow[\mathcal{A}]{a} q} \overline{\sigma_0 \ldots \sigma_{N(p)}} a \sigma_0 \ldots \sigma_{N(q)}$$

By construction, we have $G \sim \overline{g^{-1}(K)}$.

□

A family $\mathcal{F}$ is *principal* if there is a strongly connected and deterministic graph $\Gamma \in \mathcal{F}$ such that $\mathcal{F}$ is the closure of $\{\Gamma\}$ by AFG operators, we write $\mathcal{F} = \mathrm{AFG}(\Gamma)$. An AFG $\mathcal{F} = \mathrm{AFG}(\Gamma)$ is principal if and only if $\mathcal{F} = \mathrm{Fin}^{-1}(\#\Gamma \cdot_0 \Delta_0^n)$.

Most operators do not commute, for example from the graph $\Gamma = \{(0, a, 1), (1, a, 0)\}$ we have a strict inclusion $\#\mathrm{Fin}^{-1}(\Gamma) \subset \mathrm{Fin}^{-1}(\#\Gamma)$. Principality is a strong property since it can give some sense to the union operator.

Lemma 3 *If $\mathcal{F}$ is a principal AFL then*

1. *$\mathcal{F}$ is closed under union with finite graphs;*
2. *$\mathcal{F}$ is closed under disjoint union.*

PROOF.

Let Γ be a generator of $\mathcal{F}$.

(1) union with a finite graph:

Let $G \in \mathcal{F}$ and H a finite graph. As Γ is a generator of $\mathcal{F}$ there exist two mappings f, g, an integer n and a set $V \in \mathrm{Fin}(V_\Gamma \times \{0, \ldots, n\})$ such that for $K = \Gamma \square \Delta_0^n$, we have $G \sim f^{-1}(\#_{V,g} K)$. Let \$ be a special symbol and h the mapping defined by $h(\$) = \varepsilon$ We color one vertex $v \in V$ with \$ by $\#_{v,h} K$. As Γ is strongly connected and deterministic, there is a mapping i from Σ_H to $\mathrm{Fin}(\#\Sigma_\Gamma^*)$ such that $H \sim i^{-1}(\#_{v,h} K)$. Let j be the mapping defined by $j(a) = f(a) \cup i(a)$ when $a \in \Sigma_G \cap \Sigma_H$, $j(a) = f(a)$ when $a \in \Sigma_G - \Sigma_H$ and $j(a) = i(a)$ when $a \in \Sigma_H - \Sigma_G$. By construction $G \cup H \sim j^{-1}(\#_{v,h} \#_{V,g} K) \in \mathcal{F}$.

(2) disjoint union:

Let $G, H \in \mathcal{F}$. As Γ is a generator of $\mathcal{F}$ there exist four mappings f, g, h, i with $\mathrm{Dom}(g) \cap \mathrm{Dom}(i) = \emptyset$, a set $V \in \mathrm{Fin}(V_\Gamma \times \{0, \ldots, n\})$ and an integer n such that for $K = \Gamma \square \Delta_0^n$, we have $G \sim f^{-1}(\#_{V,g} K)$ and $H \sim h^{-1}(\#_{V,i} K)$. Let $c(\Delta_2)$ be a copy of Δ_2 such that $c(\Sigma_2) \cap \Sigma_K = \emptyset$. We have $G \sim I = f^{-1}(\#_g(K \cdot_0 \Delta_2))$ and $H \sim J = j^{-1}(\#_i(K \cdot_0 \Delta_2))$ with j defined by $j(a) = c(\overline{\sigma_1}) h(a) c(\sigma_1)$. By construction we have $V_I = V_\Gamma \times \{0, \ldots n\} \times \{0\}$,

$V_J = V_\Gamma \times \{0, \dots n\} \times \{1\}$ hence $I \cup J \in \mathcal{F}$.

$\square$

2.3.4 Traces of AFG

We denote the traces of a family $\mathcal{F}$ by $L(\mathcal{F}) := \{L(G, I, F) \mid G \in \mathcal{F} \wedge I, F \in Fin(V_G)\}$.

Theorem 4 *If $\mathcal{F}$ is an AFG (resp. a full-AFG) then $L(\mathcal{F})$ is a semi-cone (resp. a cone), If $\mathcal{F}$ is a principal AFG (resp. a principal full-AFG) then $L(\mathcal{F})$ is an AFL (resp. a full-AFL).*

PROOF.

Cone and semi-cone:
Let $\mathcal{F}$ be an AFG and $L \in L(\mathcal{F})$. There is a graph $G \in \mathcal{F}$, and two sets $I_G, F_G \in Fin(V_G)$ such that $L = L(G, I_G, F_G)$. To show that $L(\mathcal{F})$ is a cone (resp. a semi-cone) we show that $L(\mathcal{F})$ is closed under inverse morphism, intersection with rational set, and morphism (resp. non-erasing morphism).
Inverse morphism: Let $f : \Sigma_G^* \longrightarrow \Sigma^*$ be a morphism and g defined by $g(a) = \{f(a)\}$ its corresponding finite substitution. We have $f^{-1}(L) = L(g^{-1}(G), I_G, F_G)$ hence $f^{-1}(L) \in L(\mathcal{F})$.
Intersection with a rational set: Let $R \in Rat(\Sigma_G^*)$, let us show that $L \cap R \in L(\mathcal{F})$. Let $\mathcal{A}$ be the finite automaton recognizing R: $R = L(\mathcal{A}, i_A, F_A)$. The intersection of $L(G, I_G, F_G)$ and R is $L(G \times \mathcal{A}, I_F \times \{i_A\}, F_G \times F_A)$. By Property 1 we have $G \times \mathcal{A} \in \mathcal{F}$ hence $L \cap R \in L(\mathcal{F})$.
Non erasing rational substitution (non-erasing morphism): Let $f : \Sigma_G \longrightarrow Rat(\Sigma^+)$ be a non-erasing rational substitution (f may also be a morphism). Let a be a letter of Σ_G. As $\varepsilon \notin f(a)$, there is a finite automaton $\mathcal{A} \subseteq Q \times \Sigma \times Q$ with two distinct vertices $i_a, f_a \in Q$ such that $f(a) = L(\mathcal{A}, \{i_a\}, \{f_a\})$. For convenience we take $Q = \{0, \dots, n\}$, $i_a = 0$ and $f_a = n$. Let $c(G)$ be a copy of G such that $c(\Sigma_G) \cap \Sigma = \emptyset$, and $c(\Sigma_G) \cap \Sigma_n = \emptyset$. We define $h : \Sigma \longrightarrow Fin((c(\Sigma_G) \cup \Sigma_n)^*)$ for each letter $a \in \Sigma_G$ by

$$h(a) = \sum_{p \xrightarrow[\mathcal{A}]{a} q} \overline{\sigma_0 \dots \sigma_p}\sigma_0 \dots \sigma_q + \sum_{p \xrightarrow[\mathcal{A}]{a} n} \overline{\sigma_0 \dots \sigma_p} c(a)$$

By construction, the graph $H = h^{-1}(c(G) \cdot_0 \Delta_0^n)$ verifies

$$L(H, I_G, F_G) = f(L(G, I_G, F_G)) = f(L)$$

Rational substitution (morphism): To extend the preceding proof to any rational substitution, we use an automaton with τ transitions and we do the ε-closure to get $L(\overline{H}, I, F) = f(L)$.

AFL and full-AFL:
Let $\mathcal{F}$ be a principal AFG of generator Γ. Let L and L' in $L(\mathcal{F})$. To show that $L(\mathcal{F})$ is an AFL we show that $L \cup L'$, L^* and $L \cdot L'$ stay in $L(\mathcal{F})$. Let $G \in \mathcal{F}$ such that $L = L(G, I_G, F_G)$ and let $H \in \mathcal{F}$ such that $L' = L(H, I_H, F_H)$ and $V_G \cap V_H = \emptyset$.
Union: We have $L \cup L' = L(G \cup H, I_G \cup I_H, F_G \cup F_H)$. From Lemma 3 we have $L \cup L' \in \mathcal{F}$ hence $L \cup L' \in L(\mathcal{F})$.
L^* operation: To get L^* the transformation is the same as for finite automata: Let C be the graph defined by $C \subseteq V_G \Sigma_G \times \Sigma$ and

$$C = \{(s, a, t) \mid s \in F_G \ \wedge \ \exists v \in I_G, v \xrightarrow[H]{a} t\}$$

We have $L^* = L(G \cup C, I_G, F_H)$, from Lemma 3 $G \cup C \in \mathcal{F}$ hence $L^* \in L(\mathcal{F})$
Concatenation: To get $L \cdot L'$ the transformation is the same as for finite automata: we take the graph $C = \{(s, a, t') \mid s \in F_G \ \wedge \ t \xRightarrow[H]{a} t'\}$, so that $L \cdot L' = L(G \cup C \cup H, I_G, F_H)$, from Lemma 3 we deduce $L \cdot L' \in L(\mathcal{F})$.

□

3 Application to Pushdown Graphs

In this section, we consider some properties of *pushdown graphs* and *prefix recognizable graphs* from the AFG point of view. We define the notion of *geometrical complexity* which is preserved by AFG transformations. We then consider subfamilies of prefix recognizable graphs like *one-reversal pushdown graphs* or *polynomial graphs*.

3.1 Pushdown Graphs

A *pushdown automaton* is a finite set of rules $R \subseteq QS \times T \times QS^*$ where Q is a finite set of *states*, S is a (disjoint) *stack* alphabet and T is an alphabet of *terminals*. A *configuration* of the machine is defined by a word $qw \in QS^*$ where q is a state and w is a stack word. Its transition graph restricted to a rational set $L \subseteq QS^*$ of acceptable configurations is the graph

$$G := (RS^*)_{|L} = \{uw \xrightarrow{a} vw \mid u \xrightarrow[R]{a} v \wedge uw, vw \in L\} \qquad (1)$$

Any graph isomorphic to such a graph is called a *pushdown graph*. We denote this family PDG.

3.1.1 Prefix Recognizable Graphs:

The family of prefix recognizable graphs is a natural extension of pushdown graphs to recognizable sets of rewriting rules. We give here an internal representation, similar to the one of Equation (1): Let X be an alphabet, we denote $U \xrightarrow{a} V := \{(u,a,v) \mid u \in U \wedge v \in V\}$ the graph of transitions from $U \subseteq X^*$ to $V \subseteq X^*$ labelled by $a \in \Sigma$. A *recognizable graph* is a finite union of graphs $U \xrightarrow{a} V$ where $U, V \in \mathrm{Rat}(X^*)$. A *prefix recognizable graph* is a graph of the form

$$(RX^*)_{|L} = \{uw \xrightarrow{a} vw \mid u \xrightarrow[R]{a} v \wedge uw, vw \in L\} \tag{2}$$

Where R is a recognizable graph and $L \in \mathrm{Rat}(X^*)$ is a set of acceptable configurations. Any graph isomorphic to such a graph is called a *prefix recognizable graph* We denote this family REC.

3.1.2 The Dyck Graph:

We define the two parentheses semi-Dyck graph by $\Delta_2 = \{u \xrightarrow{a} au \mid u \in (a+b)^*\} \cup \{u \xrightarrow{b} bu \mid u \in (a+b)^*\} \cup \{bu \xrightarrow{\bar{b}} u \mid u \in (a+b)^*\} \cup \{au \xrightarrow{\bar{a}} u \mid u \in (a+b)^*\}$. A representation of Δ_2 is given in Figure 4. The trace $L(\Delta_2, \{\varepsilon\}, \{\varepsilon\})$ is the Dyck language D_2 presented in Section 1.2.

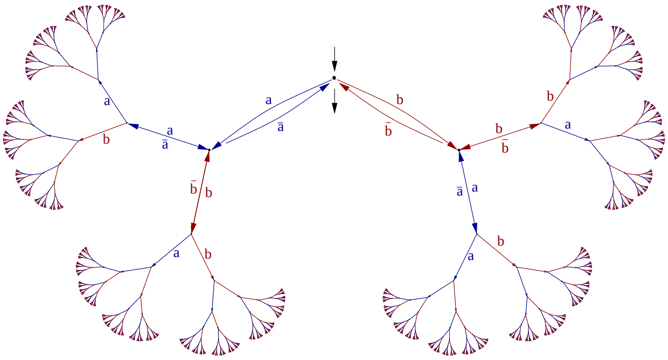


Figure 4: The two letters Dyck graph Δ_2.

Theorem 5 *[5]*

1. $\mathrm{PDG} = \mathrm{Fin}^{-1}(\#\{\Delta_2\})$;

2. $\mathrm{REC} = \mathrm{Rat}^{-1}(\#\{\Delta_2\})$;

We show easily that PDG is a principal full-AFL, and from Property 2 we deduce this result originally due to Knapik:

Property 6 $\mathsf{REC} = \overline{\mathsf{PDG}}$.

3.2 Other Generators:

3.2.1 One-reversal Pushdown Graphs

A *one-reversal pushdown automaton* is a pushdown automaton which first writes informations in the stack and secondly reads these informations. We define the diamond graph of Figure 5 by

$$\begin{array}{rcll} \Diamond_2 & = & \{(u, a, au) \mid u \in (a+b)^*\} & \cup \ \{(u, b, bu) \mid u \in (a+b)^*\} \\ & \cup & \{(u, \tau, \#u) \mid u \in (a+b)^*\} & \cup \ \{(\#au, \overline{a}, \#u) \mid u \in (a+b)^*\} \\ & \cup & \{(\#bu, \overline{b}, \#u) \mid u \in (a+b)^*\} & \end{array}$$

The trace $p_\tau(L(\Diamond_2, \varepsilon, \#))$ is the symmetric language $S_2 = aS_2\overline{a} + bS_2\overline{b} + \varepsilon$. The $\Diamond_2$ graph is a generator of the *one-reversal pushdown graphs* which traces define the cone of linear languages Lin.

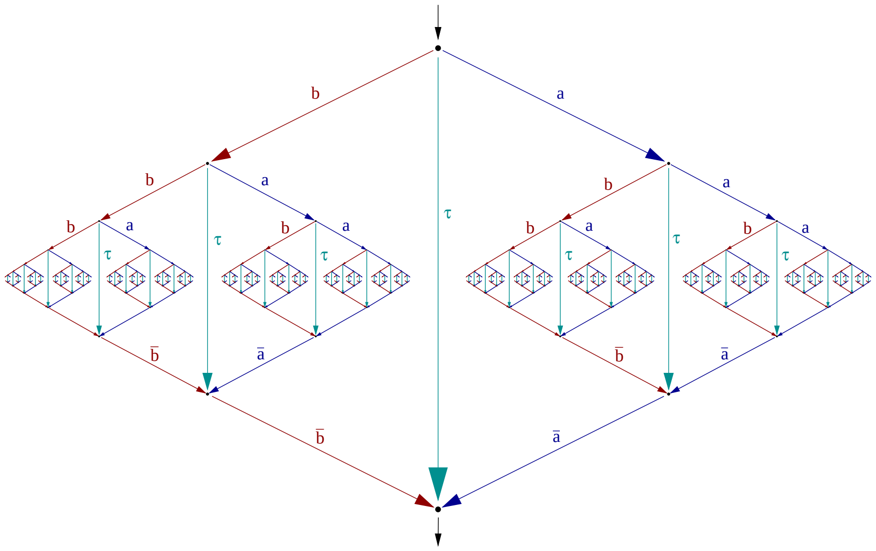


Figure 5: one-reversal pushdow graphs generator: $L\left(\overline{\mathsf{Fin}^{-1}(\#\Diamond_2)}\right) = \mathsf{Lin}$.

3.2.2 Linear Pushdown Graphs

A *one-counter automaton* is a pushdown automaton with only one stack symbol. As only the length of the stack can be used for computation, it can also be considered as a finite state machine associated with an integer register. The traces of *one-counter automaton* transition graphs define the Ocl AFL which generator is the Dyck set with one symbol $D_1 = aD_1\overline{a}D_1 + \varepsilon$. The linear Dyck graph Δ_1 described on Figure 6, is a generator of this graph family.

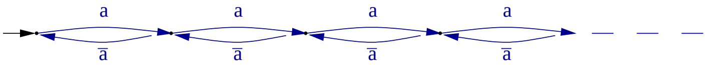


Figure 6: The one letter Dyck graph Δ_1.

3.3 Geometrical Complexity

We will show that *geometrical complexity* is a mesure preserved by AFG transformations. This criterion is used to show strict inclusion (up to isomorphism) between graph families.

3.3.1 General Definition

Let G be a graph of finite degree and let $v \in V_G$. The distance between two vertices in a same connected component of G is the function $d_G(s,t) = min\{|w| \mid s \underset{G \cup G^{-1}}{\overset{w}{\Longrightarrow}} t\}$. Let us consider the set $B_G(v,n) := \{s \in V_G \mid d_G(v,s) \leq n\}$. We call *complexity* of G from v the function defined by $c_{G,v}(n) = |B_{G,v}(n)|$. Let $f, g : I\!N \longrightarrow I\!N$ be two functions, g *dominate homothetically* f and we write $f \preceq g$ when $\exists \lambda, n_0,\ \ \forall n \geq n_0,\ f(n) \leq g(\lambda n)$. We say that f and g are *homothetically equivalent* and we write $f \equiv g$ when both $f \preceq g$ and $g \preceq f$.

Property 7 *Let G be a connected graph of finite degree, for any $s, t \in V_G$ we have $c_{G,s} \equiv c_{G,t}$.*

PROOF. Let $\delta = d_G(s,t)$. For any $n \in I\!N$ we have $B_G(t,n) \subseteq B_G(s, \delta + n)$. For every $n \geq \delta$ we have $c_{G,t}(n) \leq c_{G,s}(2n)$ i.e. $c_{G,t} \preceq c_{G,s}$. By permutation we get $c_{G,s} \preceq c_{G,t}$ therefore $c_{G,s} \equiv c_{G,t}$.

□

This property allows us to define a *complexity order* $c_G(n)$ for each connected graph G. This homothetic domination is a criterion to classify graph families.

Property 8 *Let G, H be two connected graphs of finite degree such that H is image by AFG transformations from G then $c_H \preceq c_G$.*

PROOF. The rational coloring does not change the complexity of a graph, so does the finite product. Let h be a finite substitution such that $H \sim K = h^{-1}(G)$. Let $\lambda = \max\{|w| \mid \exists a \in \Sigma_G,\ w \in h(a)\}$. Let $v \in V_G$. Let us show that $c_{H,v}(n) \leq c_{G,v}(\lambda n)$ *i.e.* $B_K(v,n) \subseteq B_G(v, \lambda n)$. If $d_K(v,s) \leq n$ then $\exists u, |u| \leq n \wedge v \underset{K}{\overset{u}{\Longrightarrow}} s$ thus $v \underset{G}{\overset{h(u)}{\Longrightarrow}} s$. As $|h(u)| \leq \lambda |u| \leq \lambda n$, we have $s \in B_G(v, \lambda n)$. Thus $B_K(v,n) \subseteq B_G(v, \lambda n)$ hence $c_{H,v}(n) \leq c_{G,v}(\lambda n)$.

□

3.3.2 Complexity in REC

The concept of *geometrical complexity* was initialy studied by [13]. He first defined the complexity for pushdown graphs and gave an extended definition for HR-equational graphs (which may be have infinite degree). Leaning on the property that a pushdown graph only admits a finite set of isomorphic connected components, we extend the complexity mesure to any prefix recognizable graph: Let $G \in \mathrm{REC}$, $H \in \mathrm{PDG}$ such that $G = \overline{H}$ and $C_1, \dots, C_n$ be the classes of connected components of H. $c_G = \max\{c_{C_i} \mid 1 \leq i \leq n\}$.

3.4 Polynomial Pushdown Graphs

A generalisation of one-counter automaton is the notion of *counter-stack automaton*: A counter-stack automaton is a finite automaton associated with a finite stack of counters each of one may be acceded only when on top of the stack. We define the *alternation languages* A_n by $A_0 = \varepsilon$, $A_{2k+1} = a^*A_{2k}$ and $A_{2k+2} = b^*A_{2k+1}$. The *comb graph* of order n is the graph defined by restriction from the Dyck graph by $\Delta_1^n \sim \{(u, a, v) \in \Delta_2 \mid u, v \in A_n\}$. The comb tree is represented in Figure 2. Figure 7 gives a fractal representation of these graphs.

The family of polynomial pushdown graphs X^n is the principal AFG of generator Δ_1^n.

Property 9 *If* $G \in X^n$ *then* $c_G \preceq k^n$.

This result is a direct consequence of Property 8. We exhibit a strictly increasing chain of AFG in PDG: for any n, $X^{n+1} - X^n \neq \emptyset$.

4 Conclusion

The general model of acceptors used by [11] to define AFA is a finite state machine associated with an auxiliary storage system. The infinite graph formalism is an elegant way to modelize these machines. A language is defined as a trace of an infinite graph. This infinite graph is obtained by simple transformations from a generator which may be considered as an auxiliary storage structure (e.g. Δ_2 is a stack, Δ_1 is a counter...). The intrinsic complexity of the graph and therefore of its traces are inherited from the generator. Figure 8 gives a summary of AFG properties for some graph families (See [14] and [6] for a definition of rational graphs and Turing graphs).

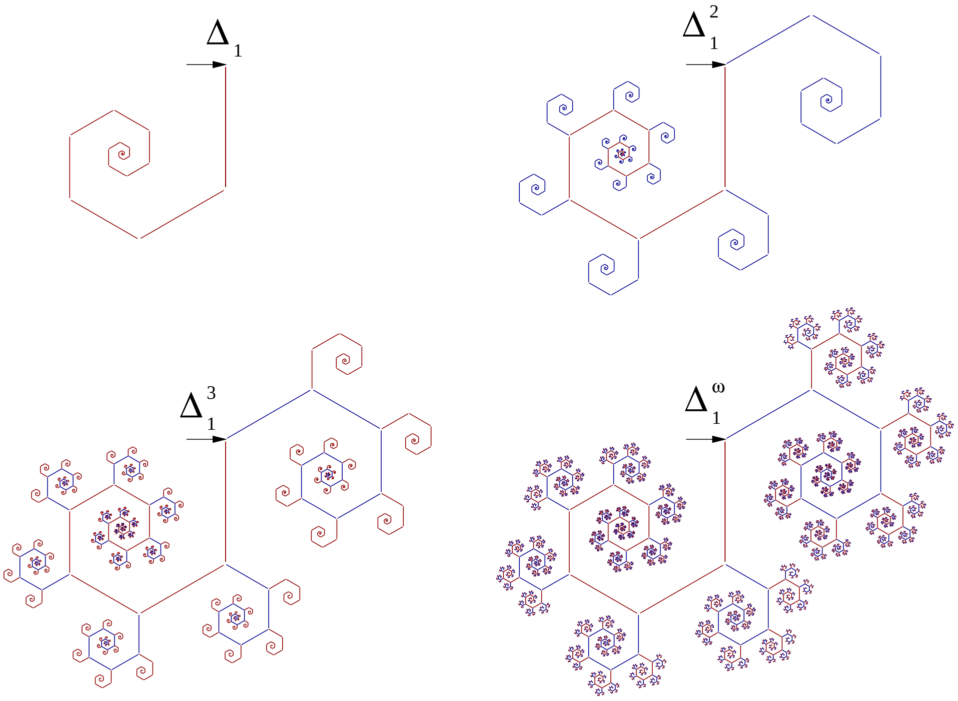


Figure 7: Polynomial Generators: $c_{\Delta_1} \equiv k$, $c_{\Delta_1^2} \equiv k^2$, $c_{\Delta_1^3} \equiv k^3$, $c_{\Delta_1^n} \equiv k^n$.

References

[1] J.-M. Autebert and L. Boasson. *Transductions Rationnelles – Application aux Langages Algébriques*. Masson, Paris, 1988.

[2] J. Berstel. *Transductions and Context-Free-Languages*. B.G. Teubner, Stuttgart, 1979.

[3] A. Blumensath. Prefix-recognizable graphs and monadic second order logic. Technical Report AIB-06-2001, RWTH Aachen, April 2001.

[4] F.J. Brandenburg. On the intersection of stacks and queues. *TCS*, 58:69–80, 1988.

[5] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP 1996*, volume 1099 of *LNCS*, pages 194–205, 1996.

[6] D. Caucal. On the transition graphs of turing machines. In *MCU 2001*, LNCS, pages 177–189, 2001.

[7] N. Chomsky and MP. Schützenberger. *The algebraic theory of context-free languages in computer programming and formal systems*. North Holland, 1963.

	closure / principality	
	AFG	full-AFG
finite graphs	yes / Δ_0^n	yes / Δ_0^n
one reversal pushdown graphs	yes / $\Diamond_2$	no
polynomial graphs	yes / Δ_1^n	no
pushdown graphs	yes / Δ_2	no
prefix recognizable graphs	yes / ?	yes / Δ_2
rational graphs	yes / yes	no
Turing graphs	yes / ?	yes / ?

Figure 8: Some graphs families

[8] B. Courcelle. The monadic second-order logic of graphs ii: infinite graphs of bounded tree width. *Math. Systems Theory*, 21:187–221, 1989.

[9] B. Courcelle. Monadic-second order definable graph transductions : a survey. *TCS*, vol. 126:pp. 53–75, 1994.

[10] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North Holland / American Elsevier, 1975.

[11] S. Ginsburg and S. A. Greibach. Abstract families of languages. *Mem. Am. Math. Soc.*, 87, 1969.

[12] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison Wesley, 1979.

[13] A. Maignan. Sur la complexité des graphes réguliers. rapport de DEA (IFSIC), Rennes, 1994.

[14] C. Morvan and C. Stirling. Rational graphs traces are context-sensitive languages. In *MFCS 2001*, number 2136 in LNCS, pages 436–446, 2001.

[15] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *TCS*, 37:51–75, 1985.

[16] W. Thomas. A short introduction to infinite automata. In *Proceedings of the 5th international conference Developments in Language Theory*. LNCS, 2001.

Undecidability of LTL for Timed Petri Nets

Parosh Aziz Abdulla Aletta Nylén

Department of Information Technology
Uppsala University
Sweden
{parosh, aletta}@it.uu.se

Abstract

We show undecidability of (action based) linear-time temporal logic (LTL) for timed Petri nets. This is to be contrasted with decidability of both the problem of checking safety properties for timed Petri nets, and the problem of checking LTL formulae for (untimed) Petri nets. The undecidability result is shown through a reduction from a similar problem for lossy counter machines [May00].

1 Introduction

In this paper we show undecidability of (action based) linear-time temporal logic (LTL) for *Timed Petri Nets (TPNs)*. The model of TPNs is a generalization of standard Petri nets in the sense that each token is equipped with a real-valued clock representing the age of the token. Furthermore each arc in the net is provided with an interval restricting the ages of tokens allowed to travel through the arc.

In fact, we show that it is already undecidable to check a certain fixed property expressible in LTL for TPNs. More precisely, we show undecidability of whether there exists a computation of the TPN along which a given transition is fired infinitely often. The undecidability result is shown through a reduction from a similar problem for *lossy counter machines* [May00].

Our result should be contrasted with decidability of the following two related problems:

- Checking safety properties for TPNs [AN00, AN01]. More precisely, in [AN01] we show how to characterize the set of markings in a TPN from which a given upward closed set of markings is reachable.

- Checking LTL formulae for standard (untimed) Petri nets [Esp94]. In fact [BM99] shows that there is an effectively constructible representation of the set of markings satisfying a formula.

We recall that (even small fragments) of branching time logics are known to be undecidable for Petri nets [Esp97, May01]. Undecidability holds also for all state based temporal logics [Esp97].

Outline In the next section TPNs and the recurrent place problem for TPNs are introduced. In Section 3 we present LCMs and the recurrent state problem. In Section 4 we prove that LTL is undecidable for TPNs.

2 Timed Petri Nets

Timed Petri Nets (TPNs) are Petri nets where each token is equipped with a real-valued clock representing the age of the token. The firing conditions for transitions include the usual ones for Petri nets. Furthermore, each arc between a place and a transition is labeled with an interval. When a transition is fired, the tokens removed from the input places of the transition and the tokens added to the output places should have ages lying in the intervals of the corresponding arcs.

We let $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ denote the sets of natural numbers and nonnegative reals respectively. A *multiset* B over a set A is a mapping from A to $\mathbb{N}$ such that for $a \in A$, $B(a)$ is the number of occurrences of a in B. We define A^M to be the set of multisets over A. Sometimes we write multisets as lists, so e.g. $(2.4, 5.1, 5.1, 2.4, 2.4)$ represents a multiset B over $\mathbb{R}^{\geq 0}$ where $B(2.4) = 3$, $B(5.1) = 2$ and $B(x) = 0$ for $x \neq 2.4, 5.1$. We may also write B as $\left(2.4^3, 5.1^2\right)$. For multisets B_1 and B_2 over a set A, we say that $B_1 \leq B_2$ if $B_1(a) \leq B_2(a)$ for each $a \in A$. We define $B_1 + B_2$ to be the multiset B where $B(a) = B_1(a) + B_2(a)$, and (assuming $B_1 \leq B_2$) we define $B_2 - B_1$ to be the multiset B where $B(a) = B_2(a) - B_1(a)$, for each $a \in A$. We use $\emptyset$ to denote the empty multiset, i.e., $\emptyset(a) = 0$ for each $a \in A$.

We use a set *Intrv* of *intervals* of the form $[a : b]$, where $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$. For $x \in \mathbb{R}^{\geq 0}$, we write $x \in [a : b]$ to denote that $a \leq x \leq b$.

A *Timed Petri Net (TPN)* is a tuple $N = (P, T, \mathit{In}, \mathit{Out})$ where P is a finite set of *places*, T is a finite set of *transitions* and $\mathit{In}, \mathit{Out} : T \times P \to \mathit{Intrv}^M$ describes the flow relation. If $\mathit{In}(t, p) \neq \emptyset$ ($\mathit{Out}(t, p) \neq \emptyset$) we say that p is an *input (output) place* of t.

A *marking* M of N is a finite multiset over $P \times \mathbb{R}^{\geq 0}$. The marking M defines numbers and ages of the tokens in each place in the net. That is, $M(p, x)$ defines the number of tokens with age x in place p. For example if marking $M = ((p_1, 2.5), (p_1, 1.3), (p_2, 4.7), (p_2, 4.7))$, then there are two tokens in p_1 with ages 2.5 and 1.3 and two tokens each with age 4.7 in the place p_2 in M. For each place p we define $M(p)$ to be the multiset over $\mathbb{R}^{\geq 0}$, where $M(p)(x) = M(p, x)$. Notice that untimed Petri nets are a special case in our model where all intervals are of the form $[0 : \infty]$.

We define two types of transition relations on markings. A *timed transition* increases the age of all tokens by the same real number. Formally $M_1 \longrightarrow_\delta M_2$ if M_1 is of the form $((p_1, x_1), \ldots, (p_n, x_n))$ and there is $\delta \in \mathbb{R}^{\geq 0}$ such that $M_2 = ((p_1, x_1 + \delta), \ldots, (p_n, x_n + \delta))$.

We define the set of *discrete transitions* $\longrightarrow_D$ as $\bigcup_{t \in T} \longrightarrow_t$, where $\longrightarrow_t$ represents the effect of firing the transition t. More precisely, we define $M_1 \longrightarrow_t M_2$ if, for each place p with $\mathit{In}(t, p) = (\mathcal{I}_1, \ldots, \mathcal{I}_m)$ and $\mathit{Out}(t, p) = (\mathcal{J}_1, \ldots, \mathcal{J}_n)$, there are multisets $B_1 = (x_1, \ldots, x_m)$ and $B_2 = (y_1, \ldots, y_n)$ over $\mathbb{R}^{\geq 0}$, such that the following holds.

- $B_1 \leq M_1(p)$.
- $x_i \in \mathcal{I}_i$, for $i : 1 \leq i \leq m$.
- $y_i \in \mathcal{J}_i$, for $i : 1 \leq i \leq n$.
- $M_2(p) = M_1(p) - B_1 + B_2$.

Intuitively, a transition t may be fired only if for each incoming arc to the transition, there is a token with the right age in the corresponding input place. This token will be removed from the input place when the transition is fired. Furthermore, for each outgoing arc a token with an age in the interval will be added to the output place. We define the relation $\longrightarrow$ to be $\longrightarrow_\delta \cup \longrightarrow_D$.

For a marking M_0 of a TPN N, an M_0-*computation* π of N is of the form $M_0, M_1, M_2, \ldots$, where $M_i \longrightarrow M_{i+1}$, for $i \geq 0$. We say that π *visits a place* p *infinitely often* if there are infinitely many i such that $M_i(p) \neq \emptyset$. The *recurrent place problem* for TPNs (RPP-TPN) is defined as follows.

Instance A TPN N, a marking M of N and a place p of N.

Question Is there an M-computation of N visiting p infinitely often?

In Theorem 4.1, we will show that RPP-TPN is undecidable.

3 Lossy Counter Machines

A *lossy counter machine (LCM)* is a tuple $L = (Q, C, \delta)$, where Q is a finite set of *states*, C is a finite set of counters and δ is a finite set of *transitions*. A transition is a triple of the form

$(q_1, \mathit{instr}, q_2)$, where $q_1, q_2 \in Q$ and *instr* is an *instruction*. An instruction is of one of the following three forms

- $c+$ which increases the value of counter c by 1.
- $c-$ which decreases the value of counter c by 1.
- $c?$ which tests whether the value of counter c is equal to 0.

A *configuration* γ of L is of the form (q, Val), where $q \in Q$ and Val is a mapping from the set C of counters to the set $\mathbb{N}$ of natural numbers. We define a transition relation $\longrightarrow$ on the set of configurations such that $(q_1, \mathit{Val}_1) \longrightarrow (q_2, \mathit{Val}_2)$ iff one of the following conditions is satisfied:

1. $(q_1, c+, q_2) \in \delta$, $\mathit{Val}_2(c) = \mathit{Val}_1(c) + 1$ and $\mathit{Val}_2(c') = \mathit{Val}_1(c')$ if $c' \neq c$.
2. $(q_1, c-, q_2) \in \delta$, $\mathit{Val}_1(c) > 0$, $\mathit{Val}_2(c) = \mathit{Val}_1(c) - 1$ and $\mathit{Val}_2(c') = \mathit{Val}_1(c')$ if $c' \neq c$.
3. $(q_1, c?, q_2) \in \delta$, $\mathit{Val}_1(c) = 0$ and $\mathit{Val}_2 = \mathit{Val}_1$.
4. $q_2 = q_1$, $\mathit{Val}_2(c) = \mathit{Val}_1(c) - 1$ for some $c \in C$, and $\mathit{Val}_2(c') = \mathit{Val}_1(c')$ if $c' \neq c$.

For a configuration γ_0, a γ_0-*computation* π of L is of the form $\gamma_0, \gamma_1, \gamma_2, \ldots$, where $\gamma_i \longrightarrow \gamma_{i+1}$, for $i \geq 0$. For a state $q \in Q$, we say that π *visits* q *infinitely often* if there are infinitely many i such that γ_i is of the form (q, Val_i). The *recurrent state problem* for LCMs (RSP-LCM) is defined as follows.

Instance A LCM L, a configuration γ of L and a state q of L.

Question Is there a γ-computation of L visiting q infinitely often?

Theorem 3.1 [May00] RSP-LCM is undecidable.

4 Undecidability of LTL

In this section, we show undecidability of linear-time temporal logic (LTL) for TPNs. We do that by showing that RPP-TPN (see Section 2) is undecidable through a reduction from RSP-LCM (Section 3). It is straightforward to show that RPP-TPN is reducible to the problem of checking whether there is a computation of the TPN along which a given transition is fired infinitely often. It follows that (action based) LTL is also undecidable.

Theorem 4.1 RPP-TPN is undecidable.

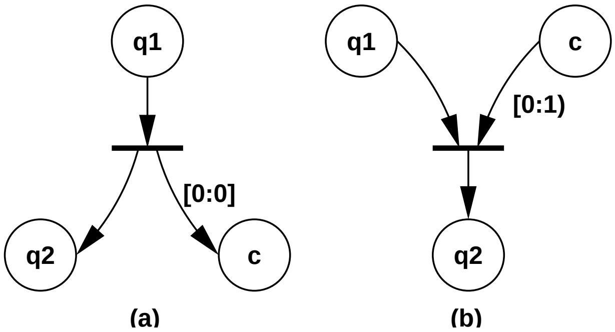


Figure 1: (a) Simulating the operation of increasing the counter. (b) Simulating the operation of decreasing the counter.

We show this through a reduction from RSP-LCM.

Theorem 4.2 RSP-LCM can be reduced to RPP-TPN.

Proof. We say that an instance of RSP-LCM or RPP-TPN is a positive instance if there is a computation such that the state or place is visited infinitely often, that is if the answer to the question in the definition of the problem is yes. Given an instance of RSP-LCM we construct an instance of RPP-TPN such that one of them is a positive instance if and only if the other one is a positive instance. Suppose that we are given an instance of RSP-LCM, i.e., an LCM L, a configuration γ of L and a state q of L. We construct an equivalent instance of RPP-TPN, i.e., we derive a TPN N, a marking M of N, and a place p of N, such that RPP-TPN has a positive answer if and only if RSP-LCM has a positive answer.

Suppose that LCM $L = (Q, C, \delta)$. We construct a corresponding TPN $N = (P, T, \mathit{In}, \mathit{Out})$ as follows. For each state $q \in Q$ there is a place in P which we call place q. We use P_Q to denote to the set of places of N corresponding to states. Also, for each counter $c \in C$ there is a place in P which we call place c. We use P_C to denote to the set of places corresponding to counters. Intuitively, the state of L is defined in N by the element of P_Q which contains a token. (The TPN N satisfies the invariant that there is at most one place in P_Q which contains a token). The value of counter c in L is defined in N by the number of tokens in place c which have ages less than 1 (tokens which have ages more than 1 are considered to have been lost and do not affect the value of the counter). Losses in L are simulated either by making the age of the token at least 1, or by firing a special *loss* transition which can always remove tokens from the places in P_C. The flow relation corresponding to *In* and *Out* reflects these properties and is defined as follows.

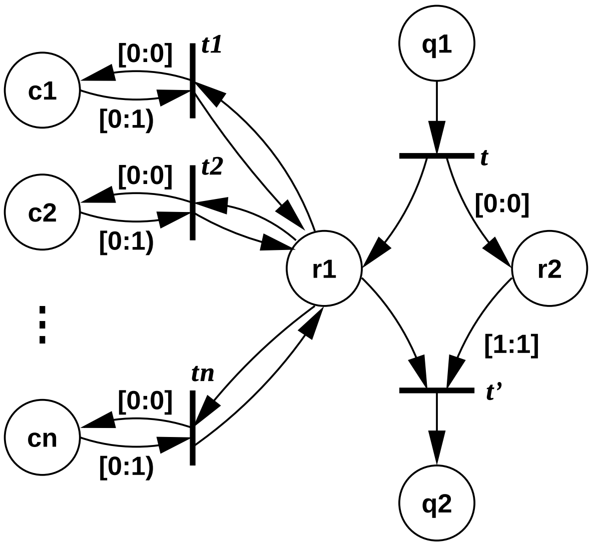


Figure 2: Simulating the operation of testing the value of the counter c.

- A transition $(q_1, c+, q_2)$ in δ is simulated by a transition in T which is of the form in Figure 1(a). The transition moves a token from place q_1 to place q_2 and adds a token to place c.
- A transition $(q_1, c-, q_2)$ in δ is simulated by a transition in T which is of the form in Figure 1(b). The transition moves a token from place q_1 to place q_2 and removes a token from place c.
- Transitions of the form $(q_1, c?, q_2)$ have the most complicated simulations. The construction is shown in Figure 2. We use two intermediate places r_1 and r_2. The transition t is first fired adding a token to each of the places r_1 and r_2. The token in r_2 will either stay in r_2 for exactly on time unit, or it will forever stay in place r_2 after which no tokens will ever reside in any place in P_Q. The idea is that we reset the value of counter c to 0, by making the ages of all tokens in place c at least 1. Observe that we simulate testing for 0 in L by resetting the counter in N. This is possible since L is lossy and therefore it may choose to decrease the counter c to 0 each time c is tested for 0. Furthermore, in order to avoid resetting the values of the rest of the counters, we add, for each $c' \in C - \{c\}$ a new transition. In Figure 2, we assume that $C - \{c\} = \{c_1, \ldots, c_n\}$, and thus we add the transitions $t_1, \ldots, t_n$. The transition is used to refresh the ages of the tokens in the places in $P_{C - \{c\}}$. For instance, if a token in place c_1 is about to become 1 and thus become too old, the transition t_1 can be fired replacing the token by a new fresh token with age 0. When the transition t' is fired, the new control state will be q_2,

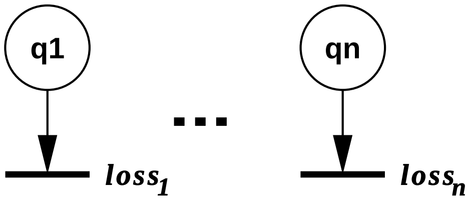


Figure 3: Simulating losses.

and each token in place c will have an age which is at least 1. The resulting marking will therefore correspond to the counter c having the value 0. We also observe that the refreshing process for the rest of the counters will be stopped after firing t', since the token in r_1 will now be removed.

- For each place c in P_C there is a transition which we call loss_c (Figure 3).

Consider a marking M of N and a configuration $\gamma = (q, \mathit{Val})$ of L. We say that M is an *encoding* of γ if M contains one token in place q and the number of tokens with ages less than 1 in place c is equal to $\mathit{Val}(c)$. Furthermore, all other places in M are empty.

We derive N from L as described above. We define M to be the encoding of γ and define p to be q.

Consider a γ-computation $\pi = \gamma_0, \gamma_1, \gamma_2, \ldots$ of L. We show that there is a M-computation $\pi' = M_0, M_1, M_2, \ldots$ of N, such that for each i there is a $j \geq i$ where M_j is an encoding of γ_i. This implies that if π visits q infinitely often then π' visits p infinitely often. We use induction on i. The base case is trivial. We know that $\gamma_i \longrightarrow \gamma_{i+1}$. This means that we can derive γ_{i+1} from γ_i, using one of the four possible types of transitions described above. We explain only the least obvious case, namely when γ_{i+1} is derived from γ_i by testing the value of a counter c for 0. The other cases can be explained in a similar manner. Let $\gamma_i = (q_i, \mathit{Val}_i)$. We know that $\mathit{Val}_i(c) = 0$. By induction hypothesis we know that there is a j such that M_j is a encoding of γ_i. This means that place q_i in M_j contains a token. From the construction described above (Figure 2) we know that we can fire a sequence of transitions, which result in moving the token from place q_i to place q_{i+1}, making the ages of all tokens in place c at least 1 and keeping the number of tokens in $P_C - \{c\}$ which have ages less than one. This means that the new marking will be a encoding of γ_{i+1}.

Suppose that there is an M-computation π of N visiting place q infinitely often. Let π be of the form $M_0, M_1, M_2, \ldots$. Consider the maximal subsequence $\pi' = M'_0, M'_1, M'_2, \ldots$ of π, where each M'_i is an encoding of some configuration of L. The sequence π' exists and is infinite since q is visited infinitely often. Let $\gamma_i = (q_i, \mathit{Val}_i)$ be the configuration which is encoded by M'_i. We show that $\gamma_i \xrightarrow{*} \gamma_{i+1}$. It follows immediately that there is a computation π'' of L visiting q infinitely often.

Since $M'_i \xrightarrow{*} M'_{i+1}$ we know that there are $M_0, M_1, \ldots, M_m$ such that $M_0 = M'_i$, $M_m = M'_{i+1}$ and $M_0 \longrightarrow M_1 \longrightarrow \cdots \longrightarrow M_m$. There are two cases. If $m = 1$, i.e., $M'_i \longrightarrow M'_{i+1}$, we know that M'_{i+1} can be derived from M'_i by firing a transition corresponding to one of those in Figure 1 or Figure 3. In this case the proof is straightforward. If $m > 1$, then M'_{i+1} is obtained from M'_i by firing transitions corresponding to those in Figure 2 (these are the only transitions in N which can make all places in P_Q empty and thus prevent the markings $M_1, \ldots, M_{m-1}$ from being encodings of configurations of L). This means that $(q_i, c?, q_{i+1})$ is a transition in L, for some counter c. From the construction of Figure 2, we know that all tokens in place c will eventually have ages which are at least 1. Furthermore, the ages of some of the tokens in $P_C - \{c\}$ may also exceed 1, since not all tokens need to be refreshed. We can derive γ_{i+1} from γ_i by first performing loss transitions corresponding to tokens which become too old followed by the transition $(q_i, c?, q_{i+1})$. □

Acknowledgement

We would like to thank Richard Mayr for helpful discussions.

References

[AN00] Parosh Aziz Abdulla and Aletta Nylén. Better is better than well: On efficient verification of infinite-state systems. In *Proc. LICS' 00* 16^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 132–140, 2000.

[AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs. In *Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 53 –70, 2001.

[BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Symp. on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333, 1999.

[Esp94] J. Esparza. On the decidability of model checking for several mu-calculi and Petri nets. In *CAAP '94*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer Verlag, 1994.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[May00] R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.

[May01] R. Mayr. Decidability of model checking with the temporal logic ef. *Theoretical Computer Science*, 256:31–62, 2001.

Uniform Solution of Parity Games on Prefix-Recognizable Graphs

Thierry Cachat

Lehrstuhl für Informatik VII, RWTH, D-52056 Aachen
Fax: (49) 241-80-22215, Email: cachat@informatik.rwth-aachen.de

Abstract

Walukiewicz gave in 1996 a solution for parity games on pushdown graphs: he proved the existence of pushdown strategies and determined the winner with an EXPTIME procedure. We give a new presentation and a new algorithmic proof of these results, obtain a uniform solution for parity games (independent of their initial configuration), and extend the results to prefix-recognizable graphs. The winning regions of the players are proved to be effectively regular, and winning strategies are computed.

1 Introduction

Games are an important model of reactive computation and a versatile tool for the analysis of logics like the μ-calculus [4, 5]. Namely we know that the model checking problem of the μ-calculus is polynomialy equivalent to the problem of solving parity games. In recent years, games over infinite graphs have attracted attention as a framework for the verification and synthesis of infinite-state systems [6].

In the present paper we consider two-player parity games on pushdown graphs (transition graphs of pushdown automata) and on prefix-recognizable graphs. It was shown in [9] that one can determine in EXPTIME the winner of a pushdown game, and that winning strategies can be realized also by pushdown automata.

The drawback of these results [6, 9] is a dependency of the analysis on a given initial game position, and a lack of algorithmic description of the (computation of) winning strategies. In this paper we extend the results of [9] to a uniform solution for parity games on prefix-recognizable graphs (independent of initial configuration), and we define explicitly the (computation of a) winning strategy.

In Section 2 we give a new presentation and proof of the results of [9] stressing upon effectivity. Section 3 presents an exemple of pushdown game. Then in Section 4 we extend these results to compute uniformly the winning region of the game (the set of configurations from which Player I can win). It is proved to be effectively regular, and a corresponding winning pushdown strategy is also uniformly defined. In Section 5 we consider parity games on prefix-recognizable graphs, which are an extension of pushdown graphs, where the degree of a vertex can be infinite [2]. We show that any prefix-recognizable game can be "simulated" by a pushdown game, in the sense that under a certain correspondence of game positions, the winner of one game is the same player as the winner of the other game. An exemple is also provided. Applying the uniform solution of Section 4, we get a uniform solution and an effective winning strategy also over prefix-recognizable graphs.

2 Pushdown Games: Walukiewicz's Results

Sections 3 and 4 of [9] are not stated in an effective (*i.e.*, algorithmic) framework, and their results "become" effective only with the help of Section 5 of [9]. We prefer to give first a new presentation of the construction of Section 5 of [9]. Then the most important results can be deduced, including all algorithmic claims.

The idea of [9] is to "reduce" the pushdown game to a parity game on a finite graph. This allows to determine the winner, and also the winning strategy. We assume the positional ("memoryless") determinacy of parity games over finite graphs, see [4].

A *Finite State Parity game* (FSP) is a tuple (S, E, λ) where $S = S_{\mathrm{I}} \uplus S_{\mathrm{II}}$ is the finite set of vertices of the game graph, $E \subseteq S \times S$ is the edge relation, and $\lambda : S \longrightarrow \{0, \cdots, max - 1\}$ is the priority function ($max > 0$). It is assumed in [9] that $E \subseteq (S_{\mathrm{I}} \times S_{\mathrm{II}}) \cup (S_{\mathrm{II}} \times S_{\mathrm{I}})$, but this is not essential. From now on we use the infix notation $\rightarrow$ for the edge relation: $\forall s, s' \in S$, $(s, s') \in E \;\Leftrightarrow\; s \rightarrow s'$.

Starting in a given initial vertex $\pi_0 \in S$, a play in (S, E, λ) proceeds as follows: if $\pi_0 \in S_{\mathrm{I}}$, Player I picks the first transition (move) to π_1, else Player II does, and so on from the new vertex π_1. A play is a (possibly infinite) maximal sequence $\pi_0\pi_1 \cdots$ of successive vertices. For the winning condition we consider the max-parity version: Player I wins the play $\pi_0\pi_1 \cdots$ iff $\limsup_{k\rightarrow\infty} \lambda(\pi_k)$ is odd, *i.e.*, iff the maximal priority seen infinitely often in the play is odd.

A *Pushdown Game System* (PDS) is a tuple (P, Γ, Δ), where $P = P_{\mathrm{I}} \uplus P_{\mathrm{II}}$ is the partitioned set of control locations, Γ the stack alphabet, and Δ a (finite) transition relation $\Delta \subseteq P \times \Gamma \times P \times \Gamma^{\leqslant 2}$, where $\Gamma^{\leqslant 2} = \epsilon \cup \Gamma \cup \Gamma^2$. The set of configurations of the PDS is $V = P\Gamma^*$, partitioned into

$V_I = P_I\Gamma^*, V_{II} = P_{II}\Gamma^*$. The set of transitions, or edge relation, is $\{(p\gamma\nu, p'\mu\nu) \mid (p, \gamma, p', \mu) \in \Delta, \nu \in \Gamma^*\}$. We also have a priority function $\Omega : P \longrightarrow \{0, \cdots, max - 1\}$, extended to V by $\Omega(p\nu) = \Omega(p)$. A play starting from an initial configuration π_0, and the winning condition are defined in the same way as for the FSP, replacing S_I and S_{II} by V_I and V_{II}. Player I wins a play $\pi_0\pi_1 \cdots$ iff $\limsup_{k\to\infty} \Omega(\pi_k)$ is odd. In this section we consider a particular initial configuration $\pi_0 = p_0 \perp$, where $\perp \in \Gamma$, $p_0 \in P$.

A *pushdown strategy* for I in its general form is a deterministic pushdown automaton with input and output. It “reads” the moves of Player II (elements of Δ) and outputs the moves (choices) of Player I , like a pushdown transducer.

Definition 2.1 *Given a game over a PDS* (P, Γ, Δ), *where* Δ_σ *is the set of transition rules in* Δ *departing from Player* σ *configurations, a* pushdown strategy *for Player* I *in this game is a* deterministic *pushdown automaton* $\mathcal{S} = (Q, A, \Pi)$, *with a set* Q *of control states, some stack alphabet* A, *and a finite transition relation* $\Pi \subseteq ((Q \times A \times \Delta_{II}) \times (Q \times A^*)) \cup ((Q \times A) \times (Q \times A^* \times \Delta_I))$.

Theorem 2.2 [9] *Given a Pushdown Game System* $\mathcal{G}$ *with a parity winning condition, one can construct a Finite State Parity game such that:*
1. the winner of the parity game over $\mathcal{G}$ *from the initial configuration* $p_0 \perp$ *is the winner of the FSP from a certain initial vertex denoted* $Check(p_0, \perp, B, \Omega(p_0))$, *where* $B \in (\mathcal{P}(P))^{max}$,
2. a winning pushdown strategy for Player I *in the parity game over* $\mathcal{G}$ *from the initial configuration* $p_0 \perp$ *can be constructed from a winning strategy for Player* I *in the FSP from the initial vertex mentioned above..*

In the sequel we present informally how a play on the PDS is “simulated” in the FSP. We will see later that a configuration $p\gamma\nu$ of the PDS, where $p \in P$, $\gamma \in \Gamma$, and $\nu \in \Gamma^*$, is represented in the FSP by a vertex $Check(p, \gamma, B, m)$, where $m \in \{0, \cdots, max - 1\}$ is a priority, and $B \in (\mathcal{P}(P))^{max}$ “summarizes” information about ν. (the number m is the highest priority seen in a certain part of the game, and B represents the set of control states q such that Player I can win the game from $q\nu$, under certain conditions depending on m). To begin with, consider the initial configuration ($p_0 \perp$), where the symbol $\perp \in \Gamma$ cannot be erased w.r.t. the rules of Δ. The corresponding vertex of the FSP is $Check(p_0, \perp, B, m)$, where in this particular case B and m are not relevant.

From a configuration $p\gamma\nu$, simulated by $Check(p, \gamma, B, m)$, the player whose turn is it is determined by p, in the PDS as well as in the FSP: either $p \in P_I$ or $p \in P_{II}$. Let $\sigma \in \{I, II\}$ such that $p \in P_\sigma$. Different types of moves are possible in the PDS.

If Player σ chooses a transition (p, γ, p', γ'), *i.e.*, if the stack length remains constant, then the FSP proceeds to the vertex $Check(p', \gamma', B, \max(m, \Omega(p')))$. This means that B remains the same, m is updated for later use and represents the maximal priority seen since last initialization of m (see below). The priority of this vertex is $\Omega(p')$ in the FSP, as well as the corresponding configuration in the PDS, and the play goes on like that until some "push" or "pop" operation occurs.

The key point is the treatment of the push operation, because one cannot store in the FSP the whole information contained in the stack. If Player σ chooses a transition $(p, \gamma, p', \gamma'\eta) \in \Delta$, *i.e.*, "pushes" one more symbol onto the stack, then in the FSP the corresponding new vertex is $\mathit{Push}(B, m, p', \gamma'\eta)$. This is an intermediate vertex were Player I (always) has to make a decision. He has to guess what can happen later, and what he can guarantee. Player I chooses a tuple $C \in (\mathcal{P}(P))^{max}$ such that he claims/guesses that whenever the symbol γ' currently at the top of the stack is "popped", then after this pop operation, the PDS will be in a control-state $q \in C_\ell$ such that ℓ is the highest priority seen between the "push" and the "pop" of this γ'. This part of the game is a "subgame" in [9], and this notion is not so far from the idea of "detour" in [7]. More precisely, γ' can be replaced later by another letter, but the condition on C must hold when the length of the stack decreases and symbol η comes at the top of the stack.

So Player I goes to the vertex $\mathit{Claim}(B, m, p', \gamma'\eta, C)$, which is a vertex of Player II . In particular, if $C = (\emptyset, \cdots, \emptyset)$, then Player I is claiming that the stack will never become again shorter. And Player I can claim that the highest priority that can be seen in the subgame is ℓ by choosing C as $(C_1, \cdots, C_\ell, \emptyset, \cdots, \emptyset)$. Player II has to answer the claim of Player I : either he thinks that Player I is bluffing, and he challenges the claim, or he believes that Player I can achieve his claim, and he wants to see what happens after the subgame.

The second case is simple: Player II goes to vertex $\mathit{Jump}(q, \eta, B, m, \ell)$ such that $q \in C_\ell$. This is an intermediate vertex which, as a shortcut, simulates one of the above mentioned subgames: among the propositions of Player I , Player II chooses that the highest priority seen in this subgame was ℓ, and when η appears again at the top of the stack, the new control state is q. The priority of this $\mathit{Jump}()$ vertex is ℓ in the FSP. Then the play goes on to $\mathit{Check}(q, \eta, B, \max(\ell, m, \Omega(q)))$ without any alternative.

In the first case, when Player II challenges the claim, he goes to vertex $\mathit{Check}(p', \gamma', C, \Omega(p'))$. This means that the last component is reset to $\Omega(p')$, and will remember the maximal priority seen in the subgame we just entered. The tuple C is stored, and whenever a "pop" operation occurs later, it is possible to check if the claim of Player I is achieved. If it is, this means immediate win for Player I . If it is not, this means immediate win for Player II (see the proof for details, and above for the update of m). But the play can also stay forever in the Check() vertices, *i.e.*, without "pop". In this case the winner is determined by the parity condition. In fact the claim of Player I after a push operation means also that if no pop occurs later, then he has to win the subgame just with the parity condition.

We restrict ourselves in this paper to the following form of pushdown strategy. We consider a strategy automaton (Q, A, Π) where $Q = P = P_I \uplus P_{II}$, $A = \Gamma \times \Sigma$, Σ is any alphabet, and $\Pi \subseteq ((P_{II} \times A \times \Delta_{II}) \times (P \times A^*)) \cup ((P_I \times A) \times (P \times A^* \times \Delta_I))$. Moreover we have the condition that whenever the game is in a configuration $p\gamma_0 \cdots \gamma_n$, the strategy automaton should be in a configuration $p(\gamma_0, \sigma_0) \cdots (\gamma_n, \sigma_n)$, which means that the strategy has more information in its stack, represented by $\sigma_0 \cdots \sigma_n$, but follows the play. If $p \in P_I$, then $p(\gamma_0, \sigma_0)$ determines the move of Player I w.r.t. Π, and the strategy updates its stack. If $p \in P_{II}$, then for any move of Player II , *i.e.*, for any transition in Δ_{II}, the strategy should update its stack. At the beginning of the play, the strategy has to be initialized properly, according to the initial configuration of the game. Then for each move of the play, the strategy executes a transition.

In our particular form of strategy, there is a redundancy in the transition relation: suppose $p(\gamma_0, \sigma_0)$ is the top of the stack, if $p \in P_I$, then a unique transition is possible in the strategy, and the output of the move $\delta_I \in \Delta_I$ can be deduced from the update of the stack. If $p \in P_{II}$, then a unique transition can follow the choice of Player II and update the stack accordingly, so the input of $\delta_{II} \in \Delta_{II}$ is redundant. From now on we consider $\Pi \subseteq (P \times A) \times (P \times A^{\leqslant 2})$.
Formally, if $p \in P_I$, then $\forall a \in A\ \exists!(p, a, p', w) \in \Pi$. Moreover if $(p, a, p', w) \in \Pi$ and $a = (\gamma, \sigma)$, $w = (\gamma_1, \sigma_1) \cdots (\gamma_k, \sigma_k)$ $(2 \geqslant k \geqslant 0)$, then $(p, \gamma, p', \gamma_1 \cdots \gamma_k) \in \Delta$, that is to say the hint of the strategy is valid. If $p \in P_{II}$, then $\forall (p, \gamma, p', \gamma_1 \cdots \gamma_k) \in \Delta\ \exists!(p, a, p', w) \in \Pi$ such that $a = (\gamma, \sigma)$ and $w = (\gamma_1, \sigma_1) \cdots (\gamma_k, \sigma_k)$ $(2 \geqslant k \geqslant 0)$.

Proof of Theorem 2.2

Definition of the FSP

The PDS is given by $\mathcal{G} = (P, \Gamma, \Delta)$, $P = P_I \uplus P_{II}$, and Ω.
For every $p, p', q \in P$; $\gamma, \gamma', \eta \in \Gamma$; $m, \ell \in \{0, \cdots, max - 1\}$; $B, C \in (\mathcal{P}(P))^{max}$, the FSP has the following vertices:

$$\mathit{Check}(p, \gamma, B, m), \quad \mathit{Push}(B, m, p', \gamma'\eta),$$
$$\mathit{Claim}(B, m, p', \gamma'\eta, C), \quad \mathit{Jump}(p, \gamma, B, m, \ell), \quad \mathit{Win}_I(p), \mathit{Win}_{II}(p),$$

and the following transitions:

$\mathit{Check}(p, \gamma, B, m) \to \mathit{Check}(p', \gamma', B, \max(m, \Omega(p')))$ if $(p, \gamma, p', \gamma') \in \Delta$,
$\mathit{Check}(p, \gamma, B, m) \to \mathit{Win}_I(p')$ if $(p, \gamma, p', \epsilon) \in \Delta$ and $p' \in B_m$,
$\mathit{Check}(p, \gamma, B, m) \to \mathit{Win}_{II}(p')$ if $(p, \gamma, p', \epsilon) \in \Delta$ and $p' \notin B_m$,

$\mathrm{Check}(p,\gamma,B,m) \to \mathrm{Push}(B,m,p',\gamma'\eta)$ if $(p,\gamma,p',\gamma'\eta) \in \Delta$,
$\mathrm{Push}(B,m,p',\gamma'\eta) \to \mathrm{Claim}(B,m,p',\gamma'\eta,C)$,
$\mathrm{Claim}(B,m,p',\gamma'\eta,C) \to \mathrm{Check}(p',\gamma',C,\Omega(p'))$,
$\mathrm{Claim}(B,m,p',\gamma'\eta,C) \to \mathrm{Jump}(q,\eta,B,m,\ell)$ if $q \in C_\ell$, and
$\mathrm{Jump}(q,\eta,B,m,\ell) \to \mathrm{Check}(q,\eta,B,\max(\ell,m,\Omega(q)))$.

One defines in the FSP the player whose turn it is: $\mathrm{Check}(p,\gamma,B,m) \in S_I \Leftrightarrow p \in P_I$, but $\mathrm{Push}(B,m,p',\gamma'\eta) \in S_I$ and $\mathrm{Claim}(B,m,p',\gamma'\eta,C) \in S_{II}$. From other vertices, the players have no alternative: there is a unique successor.

One has the following priorities:
$\lambda(\mathrm{Check}(p,\gamma,B,m)) = \Omega(p)$, $\lambda(\mathrm{Jump}(q,\gamma,B,m,\ell)) = \ell$, and
$\lambda(\mathrm{Push}(B,m,p',\gamma'\eta)) = \lambda(\mathrm{Claim}(B,m,p',\gamma'\eta,C)) = 0$ because the latter are intermediate vertices which should not interfere with the parity condition.

It remains to clarify the situation concerning deadlocks. If the first letter of the stack and the control state do not permit to execute a transition, there is a deadlock in the PDS as in the corresponding vertex of the FSP. We leave to the reader to choose the convention concerning which player wins in that case.

If one needs a bottom stack symbol ($\perp$), that cannot be erased and cannot be pushed, one has to care for this explicitly in Γ and Δ. Otherwise when the stack is empty, no transition is possible in our framework of PDS. We have again to choose a convention for this type of deadlock. It concerns the choice of B in the initial vertex $\mathrm{Check}(p_0,\perp,B,\Omega(p_0))$ of the FSP.

Equivalence between the games: from FSP to PDS

Suppose that Player I has a winning strategy in the FSP from vertex $\mathrm{Check}(p_0,\perp,B,\Omega(p_0))$. Since the game graph is finite, and the strategy can be taken positional [4], it is effectively given as a subset of the set of transitions, and denoted $\overset{str}{\to} \subseteq \to$. We will define from it a winning pushdown strategy in the PDS. This construction will be effective.

The strategy automaton is (P,A,Π), with $A = \Gamma \times \Sigma$. We fix $\Sigma = (\mathcal{P}(P))^{max} \times \{0,\cdots,max-1\}$. For notational convenience, an element $(\gamma,(B,m))$ of A will be written γBm, and a transition $((p,\gamma Bm),(p',\gamma'B'm')) \in \Pi$ will be written as a prefix rewriting rule $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\gamma'B'm'$. Similarly $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\epsilon$, and $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\gamma'B'm'\gamma''B''m''$.

The initial configuration of the PDS is $p_0\perp$, and the one of the FSP is $\mathrm{Check}(p_0,\perp,B,\Omega(p_0))$, where B is chosen according to the convention about empty stack (see above). The initial configuration of the strategy is $p_0 \perp B\Omega(p_0)$. From a configuration $p\,\gamma Bm w$ of the strategy automaton, where $w \in A^*$, the transition in Π is defined as follows:

If $p \in P_I$, then we know that in the FSP Player I chooses the next vertex from $\mathsf{Check}(p,\gamma,B,m)$ according to $\overset{str}{\to}$.

- If $\mathsf{Check}(p,\gamma,B,m) \overset{str}{\to} \mathsf{Check}(p',\gamma',B,\max(m,\Omega(p')))$, then use the transition $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\gamma' B\max(m,\Omega(p'))$.

- If $\mathsf{Check}(p,\gamma,B,m) \overset{str}{\to} \mathsf{Win}_I(p')$, then apply $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\epsilon$. Of course in the PDS there is no immediate win, but the game goes on (cf jump move). Moreover it is necessary, in the new top letter $\gamma' B' m'$ of the stack to update m' according to m and $\Omega(p')$, as follows (details are left to the reader): $p\,\gamma Bm\gamma' B' m' \overset{str}{\hookrightarrow} (p',m)\gamma' B' m' \overset{str}{\hookrightarrow} p'\,\gamma' B' \max(m,m',\Omega(p'))$.

- If $\mathsf{Check}(p,\gamma,B,m) \overset{str}{\to} \mathsf{Push}(B,m,p',\gamma'\eta) \overset{str}{\to} \mathsf{Claim}(B,m,p',\gamma'\eta,C)$, then apply $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\gamma' C\Omega(p')\,\eta Bm$. Of course in the PDS Player II has no opportunity to jump, he must enter the subgame.

If $p \in P_{II}$, then Player II chooses any possible transition in the PDS, and the Strategy automaton updates its stack according to the winning strategy $\overset{str}{\to}$ of the FSP. More precisely, if Player II chooses $(p,\gamma,p',\gamma') \in \Delta$, then the strategy executes $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\gamma' B\max(m,\Omega(p'))$. If II chooses $(p,\gamma,p',\epsilon) \in \Delta$, then the strategy choses $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\epsilon$, followed by an update of m'. If II chooses $(p,\gamma,p',\gamma'\eta) \in \Delta$, then we have to follow $\overset{str}{\to}$ in the FSP, and find C such that $\mathsf{Push}(B,m,p',\gamma'\eta) \overset{str}{\to} \mathsf{Claim}(B,m,p',\gamma'\eta,C)$. Then $p\,\gamma Bm \overset{str}{\hookrightarrow} p'\,\gamma' C\Omega(p')\,\eta Bm$ is applied.

Because $\overset{str}{\to}$ is winning in the FSP, $\overset{str}{\hookrightarrow}$ is also winning in the PDS. Moreover using known algorithms to solve the FSP, we have constructed a pushdown strategy which is winning.

from PDS to FSP

Given a winning strategy in the PDS, we will define a winning strategy in the FSP. Here a strategy in the PDS from initial configuration $p_0 \perp = \pi_0$ is a function Str which associates to the prefix $\pi_0 \cdots \pi_n$ of a play a "next move", *i.e.*, a transition in Δ. We consider a strategy for Player I , so it is defined if $\pi_n \in V_I$. This function is not necessarily computable, so this part is not effective.

As above, a vertex $\mathsf{Check}(p,\gamma,B,m)$ corresponds to a configuration $p\gamma v$ of the PDS. If $p \in P_{II}$, the PDS has to follow the move of the FSP in the usual way, whereas if $p \in P_I$, the strategy Str determines the "good" move of the FSP. The only difficult point is the push operation: from $\mathsf{Push}(B,m,p',\gamma'\eta)$ Player I has to guess a tuple $C \in (\mathcal{P}(P))^{max}$ of sets of possible control states after the next pop. This is well defined if function Str is well defined, although this is a second reason why this part is not effective (even if Str is effective). ∎

Corollary 2.3 *If there is a winning strategy for Player* I *in the parity game over the PDS, then there is effectively a winning pushdown strategy.*

The results in [6, 7] are in some sense stronger. One can deduce from them the winner, and a winning strategy defined by a finite automaton with output. It reads the current configuration and outputs the "next move". This strategy is positional and can also be executed by a pushdown automaton.

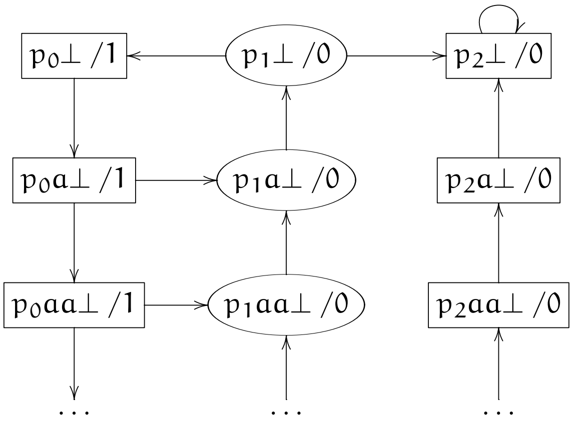
Note that in the above construction, the FSP has the same number max of priorities than the PDS, and the number of vertices is exponential in $|P|$ (more precisely, in $\mathcal{O}(|\Delta|e^{2max|P|})$). So far the best known algorithms to solve finite state parity game are polynomial in the number of vertices and exponential in the number of priorities. Applied here, we get a solution for parity games over pushdown systems (P, Γ, Δ) which is exponential in $max|P|$.

3 Example

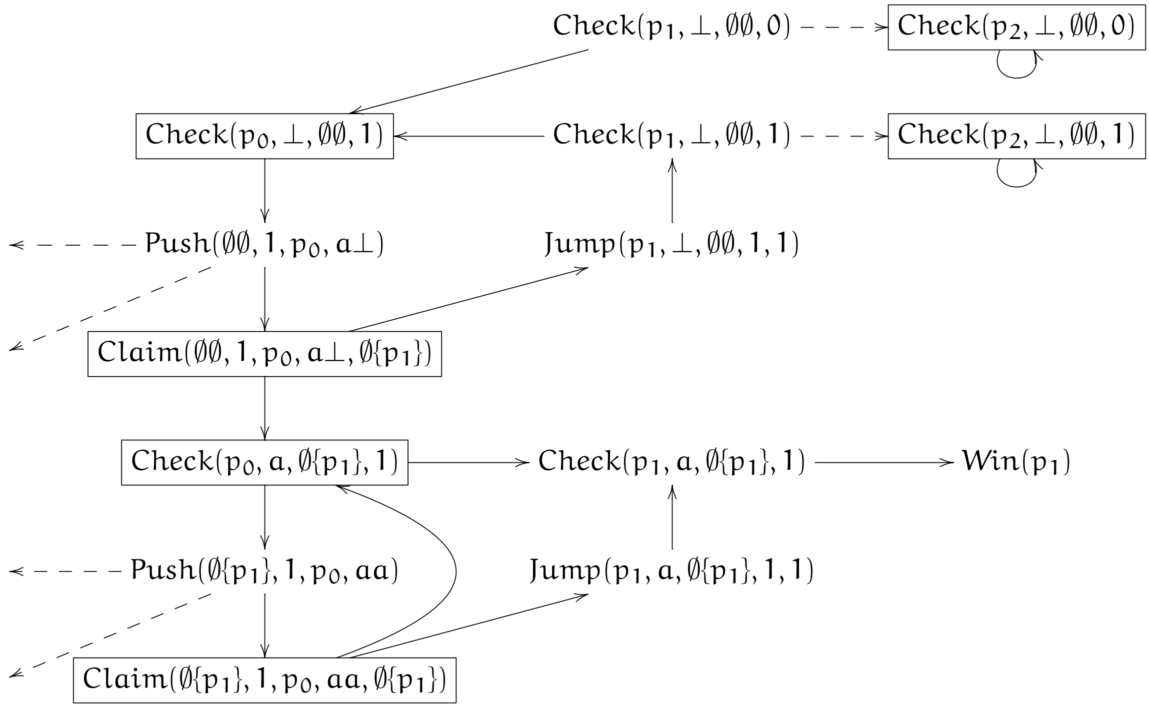
We present here a simple example of pushdown game to illustrate the previous section. Let

$$\begin{aligned}
&\Gamma = \{a, \bot\},\ P_{\mathrm{I}} = \{p_1\},\ P_{\mathrm{II}} = \{p_0, p_2\},\\
&\Delta = \{(p_0, \bot, p_0, a\bot), (p_0, a, p_0, aa), (p_0, a, p_1, a), (p_1, a, p_1, \epsilon),\\
&\qquad (p_1, \bot, p_0, \bot), (p_1, \bot, p_2, \bot), (p_2, \bot, p_2, \bot), (p_2, a, p_2, \epsilon)\},\\
&\Omega(p_0) = 1,\ \Omega(p_1) = \Omega(p_2) = 0,\ max = 2.
\end{aligned}$$

The game graph looks like the following:



We consider the initial configuration $p_1\bot$. We represent below the part of the corresponding FSP that is relevant for Player I . Namely the solid-arrows define a winning strategy for Player I , other arrows are dashed. We will write B_0B_1 for a tuple (B_0, B_1) in $(\mathcal{P}(P))^2$. The symbol $\bot$ cannot be removed from the stack, so the initial value of the tuple $B \in (\mathcal{P}(P))^2$ is not relevant. We set it to $\emptyset\emptyset$ in the initial vertex $\mathrm{Check}(p_1, \bot, \emptyset\emptyset, 0)$.



$\mathrm{Check}(p_1, \bot, \emptyset\emptyset, 0)$
$\mathrm{Check}(p_2, \bot, \emptyset\emptyset, 0)$
$\mathrm{Check}(p_0, \bot, \emptyset\emptyset, 1)$
$\mathrm{Check}(p_1, \bot, \emptyset\emptyset, 1)$
$\mathrm{Check}(p_2, \bot, \emptyset\emptyset, 1)$
$\mathrm{Push}(\emptyset\emptyset, 1, p_0, a\bot)$
$\mathrm{Jump}(p_1, \bot, \emptyset\emptyset, 1, 1)$
$\mathrm{Claim}(\emptyset\emptyset, 1, p_0, a\bot, \emptyset\{p_1\})$
$\mathrm{Check}(p_0, a, \emptyset\{p_1\}, 1)$
$\mathrm{Check}(p_1, a, \emptyset\{p_1\}, 1)$
$\mathrm{Win}(p_1)$
$\mathrm{Push}(\emptyset\{p_1\}, 1, p_0, aa)$
$\mathrm{Jump}(p_1, a, \emptyset\{p_1\}, 1, 1)$
$\mathrm{Claim}(\emptyset\{p_1\}, 1, p_0, aa, \emptyset\{p_1\})$

We see that Player I has a winning strategy from $\mathrm{Check}(p_1, \bot, \emptyset, 0)$, by choosing always $(\emptyset, \{p_1\})$ after a Push node, and, of course, going from $p_1\bot$ to $p_0\bot$.

4 Extension to a Uniform Solution

The deficit of the result of [9] is that the winner is determined only from the initial position $p_0\ \bot$. We give here an algorithm which determines the winner from any position. One need a pre-computation to solve the FSP below, e.g. with the algorithm of [8]. Moreover we get a global or "symbolic" representation of the whole winning region, which will be proved to be regular (configurations are words over the alphabet $P \cup \Gamma$).

We have seen that a configuration $p\gamma v$ of the PDS is represented in the FSP by $\mathrm{Check}(p, \gamma, B, m)$ where B "summarizes" information about v. This can be used more systematically if we know from which configurations qv Player I can win. For all $B \subseteq P$, we write $[B]^{max} = (B, \cdots, B) \in (\mathcal{P}(P))^{max}$.

Algorithm 4.1 (uniform solution for parity game on PDS)
Input: *a PDS* (P, Γ, Δ), $P = P_I \uplus P_{II}$, *and a priority function* $\Omega : P \longrightarrow \{0, \cdots, max - 1\}$, *a configuration* $\pi_0 = p\gamma_0\gamma_1 \cdots \gamma_n \in P\Gamma^*$
Output: *a winning strategy from* π_0, *or the answer "*π_0 *is not in the winning region"*

```
Solve first the FSP corresponding to the PDS (see proof of Theorem 2.2). Determine the winning region $W_I$: the set of vertices from which Player I has a winning strategy in the FSP, and compute a positional (and uniform) winning strategy on $W_I$.
$D_{n+1} := \emptyset$
for $i := n$ downto 0 do
  $D_i := \{q \in P \mid Check(q, \gamma_i, [D_{i+1}]^{max}, \Omega(q)) \in W_I\}$
end for
if $p \notin D_0$ then
  answer "$\pi_0$ is not in the winning region"
else
  answer "there is a winning pushdown strategy with initial configuration
  $p\ \gamma_0[D_1]^{max}\Omega(p)\ \gamma_1[D_2]^{max}0\ \cdots\ \gamma_n[D_{n+1}]^{max}0$, and transitions like in the proof of Theorem 2.2."
end if
```

In the initialization of D_{n+1}, $\emptyset$ should be replaced by some $D_{n+1} \subseteq P$ if there is another convention about empty stack. More formally, this iterative computation can be transformed into an alternating automaton reading the word $p\gamma_0\gamma_1 \cdots \gamma_n$, where the transitions are defined depending on which configurations are winning in the FSP. This proves that the winning region of the PDS is regular.

Theorem 4.2 *Given a PDS with a parity winning condition, one can compute uniformly the winning region of Player* I *, which is regular, and a winning pushdown strategy with Algorithm 4.1.*

For the proof we observe that the winning condition concerns only the priorities seen infinitely often, and the result of a play does not depend on a finite prefix of it. The initialization of the strategy, as well as determining the winner, is in linear time in the length of the configuration, and the computation of the "next step" is in constant time.

It remains open how to extend the techniques of [6, 7] also to a uniform solution.

Example

We consider the same example as in section 3. If we solve completely the FSP, we see that the following nodes are in the winning region of Player I :

$$\begin{array}{ccc} \mathrm{Check}(p_0, \perp, B_0B_1, 1) & \text{for all} & (B_0, B_1) \in (\mathcal{P}(P))^2, \\ \mathrm{Check}(p_1, \perp, B_0B_1, 0) & \text{for all} & (B_0, B_1) \in (\mathcal{P}(P))^2, \\ \mathrm{Check}(p_0, a, B_0B_1, 1) & \text{if} & p_1 \in B_1, \\ \mathrm{Check}(p_1, a, B_0B_1, 0) & \text{if} & p_1 \in B_0, \\ \mathrm{Check}(p_2, a, B_0B_1, 0) & \text{if} & p_2 \in B_0. \end{array}$$

Applaying the algorithm to a configuration $\pi_0 = pa \cdots a\perp$, we get $D_{n+1} := \emptyset$ and for all $i \in \{0, n\}$ $D_i = \{p_0, p_1\}$. Then the winning region of Player I in the PDS is $\{p_0, p_1\}a^*\perp$.

5 Parity Games on Prefix-Recognizable Graphs

Among several equivalent definitions of Prefix-Recognizable Graph (class $\mathrm{REC}_{\mathrm{RAT}}$ in [2]) we choose the following. Given a finite alphabet Γ, a graph, or set of edges $G \subseteq \Gamma^* \times \Gamma^*$ is a PRG iff

$$G = \{uw \hookrightarrow vw \mid u \in U_i, v \in V_i, w \in W_i, 1 \geqslant i \geqslant N\},$$

where for all i, $1 \leqslant i \leqslant N$, the U_i, V_i, W_i are regular sets over Γ.

Games over PRG are defined in a natural way. In a configuration $x \in \Gamma^+$, the first letter determines the priority and the player whose turn it is. For technical reasons the priority function is $\Omega : \Gamma \longrightarrow \{2, \cdots, max - 1\}$, extended to Γ^+ by $\Omega(ax) = \Omega(a), \forall a \in \Gamma, x \in \Gamma^*$. And we have $\Gamma = \Gamma_{\mathrm{I}} \uplus \Gamma_{\mathrm{II}}$, $V_{\mathrm{I}} = \Gamma_{\mathrm{I}}\Gamma^*$, and $V_{\mathrm{II}} = \Gamma_{\mathrm{II}}\Gamma^*$, similarly to the PDS. A game starting from $\pi_0 \in \Gamma^+$ is defined in the usual way. Again we consider max-parity: I wins $\pi_0\pi_1 \cdots$ iff $\limsup_{k \to \infty} \Omega(\pi_k)$ is odd.

Reduction to Parity Game on Pushdown Graph

We will define a PDS (P, Γ', Δ) which is equivalent to the PRG in the sense that Player I wins the PDS iff he wins the PRG, and a winning strategy in one game can be effectively constructed from a winning strategy in the other game.

Let $\Gamma' = \Gamma \uplus \{\perp\}$. A vertex $ax \in \Gamma^+$ of the PRG ($a \in \Gamma$) is represented by the configuration $t_k^{\mathrm{I}}ax\perp$ or $t_k^{\mathrm{II}}ax\perp$, if $k = \Omega(a)$ and a is in Γ_{I} or Γ_{II} respectively ($t_k^{\mathrm{I}}, t_k^{\mathrm{II}} \in P$). The idea of the reduction is to decompose the transition $uw \hookrightarrow vw$ of the PRG letter by letter, using intermediate configurations in the PDS.

Theorem 5.1 *Given a PRG with parity condition, one can construct in linear time a PDS which is equivalent to the PRG in the following sense: a play over the PRG is mapped to a play over the PDS*

preserving the winning condition. Consequently:
1. the winner of the parity-PRG from a given configuration is the winner of the parity-PDS from the corresponding configuration,
2. a winning strategy in the parity-PRG can be calculated from a winning strategy in the parity-PDS.

Proof: Each regular set is recognized by a (say deterministic, complete) finite automaton: $\mathcal{B}_i$ for U_i, $\mathcal{C}_i$ for $\tilde{V}_i$, $\mathcal{D}_i$ for W_i. Here $\tilde{V}_i$ is the mirror language of V_i, *i.e.*, $\mathcal{C}_i$ is reading from right to left. We note p_{ij} the states of $\mathcal{B}_i$, q_{ij} and r_{ij} those of $\mathcal{C}_i$ and $\mathcal{D}_i$ respectively. The corresponding initial states are p_{i0}, q_{i0}, and r_{i0}. We note $p_{ij} \xrightarrow{a} p_{ij'}$ a transition in $\mathcal{B}_i$ labeled by the letter a. In the following j is always an integer in the *finite* range $[0, NS]$ where NS is the maximal number of states of the automata $\mathcal{B}_i$, $\mathcal{C}_i$, and $\mathcal{D}_i$.

We define now the PDS that simulates the PRG. The control-states of the PDS have the same names as the states of the automata $\mathcal{B}_i$, $\mathcal{C}_i$, and $\mathcal{D}_i$, with additional superscript I or II (i is ranging over $[1, N]$):

$$\begin{aligned} P_I &= \{p^I_{ij} \mid 0 \leqslant j \leqslant NS\} \cup \{t^I_k \mid 2 \leqslant k < max\} \cup \{s^I_i\} \\ P_{II} &= \{p^{II}_{ij} \mid 0 \leqslant j \leqslant NS\} \cup \{t^{II}_k \mid 2 \leqslant k < max\} \cup \{s^{II}_i\}\,. \end{aligned}$$

Additional control states t^I_k and t^{II}_k are used to mark the configurations of the PDS that correspond to vertices of the PRG. States s^I_i, s^{II}_i are added for technical reasons.

The transitions rules of the PDS are the following: for all $a, b \in \Gamma, c \in \Gamma'$,

$t^I_k c \hookrightarrow p^I_{i0} c$ (Player I chooses to use in the PRG a transition of type i: $u_i w_i \hookrightarrow v_i w_i$ $u_i \in U_i, v_i \in V_i, w_i \in W_i$),

$t^{II}_k c \hookrightarrow p^{II}_{i0} c$ (similarly for Player II).

Then for all $\sigma \in \{I, II\}$,

$p^\sigma_{ij} a \hookrightarrow p^\sigma_{ij'}$ if $p_{ij} \xrightarrow[\mathcal{B}_i]{a} p_{ij'}$ ("reading" of u_i),

$p^\sigma_{ij} c \hookrightarrow s^{\bar{\sigma}}_i c$ if p_{ij} is a final state of $\mathcal{B}_i$ (Player σ decides that the word u_i ends here, and asks the opponent for agreement).

The opponent of σ is denoted $\bar{\sigma}$.

$s^{\bar{\sigma}}_i c \hookrightarrow r^\sigma_{i0} c$ (the opponent wants to verify that the rest of the stack is really in W_i, because he thinks that this is not the case)

$r^\sigma_{ij} a \hookrightarrow r^\sigma_{ij'}$ if $r_{ij} \xrightarrow[\mathcal{D}_i]{a} r_{ij'}$ ("reading" of w_i, then:)

$r^\sigma_{ij} \perp$ is immediate lost for σ if r^σ_{ij} is not final in $\mathcal{D}_i$,
and immediate win for σ if r^σ_{ij} is final in $\mathcal{D}_i$,

$s^{\bar{\sigma}}_i c \hookrightarrow q^\sigma_{i0} c$ (otherwise, the opponent is trusting Player σ, and lets him continue),

$q^\sigma_{ij} c \hookrightarrow q^\sigma_{ij'} bc$ if $q_{ij} \xrightarrow[\mathcal{C}_i]{b} q_{ij'}$ ("writing" of v_i, chosen by Player σ),

$q_{ij}^{\sigma}a \hookrightarrow t_k^{\mathrm{I}}a$ if q_{ij} is a final state of $\mathcal{C}_i$, $a \in \Gamma_{\mathrm{I}}$ and $k = \Omega(a)$ (Player σ chooses that ν_i ends here),
$q_{ij}^{\sigma}a \hookrightarrow t_k^{\mathrm{II}}a$ if q_{ij} is a final state of $\mathcal{C}_i$, $a \in \Gamma_{\mathrm{II}}$ and $k = \Omega(a)$ (similarly).
Note that the control state t_k^{I} is redundant with the first letter.

Of course the priority of t_k^{σ} is $\Omega(t_k^{\sigma}) = k$. Given $x \in \Gamma^+$, it is clear that: $x \hookrightarrow y$ in the PRG ($y \in \Gamma^*$) iff from the corresponding configuration $t_k^{\sigma}x\bot$, Player σ can reach the configuration $t_k^{\sigma'}y\bot$ corresponding to y or wins immediately (if the opponent $\bar{\sigma}$ thinks that σ wants to violate the transition rule).

Now the unique deficit of this construction is that Player σ can stay forever in the states q_{ij}^{σ}, pushing infinitely many new letters onto the stack. To avoid this unfair behavior, which does not correspond to a real transition of the PRG, we define the following priorities: $\Omega(p_{ij}^{\mathrm{I}}) = \Omega(q_{ij}^{\mathrm{I}}) = \Omega(r_{ij}^{\mathrm{I}}) = \Omega(s_i^{\mathrm{I}}) = 0$, $\Omega(p_{ij}^{\mathrm{II}}) = \Omega(q_{ij}^{\mathrm{II}}) = \Omega(r_{ij}^{\mathrm{II}}) = \Omega(s_i^{\mathrm{II}}) = 1$. These priorities are lower than the normal priorities, so they have no influence on the winning condition of the “real” game. One takes 0 for Player I , while Player I wants an odd number; so he can’t win by staying in those intermediate states. Similarly for Player II . ■

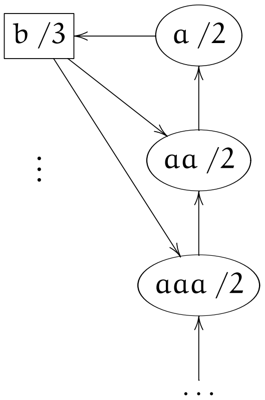
The reduction presented here is not so far from the result of [3] (Proposition 4.2 of the full version), that the prefix recognizable graphs are obtained from the pushdown graphs by ϵ-closure. It should be possible to extend the symbolic solution of [1] for reachability and Büchi games on pushdown graphs to prefix-recognizable graphs (with the new feature of optimal strategy).

Example

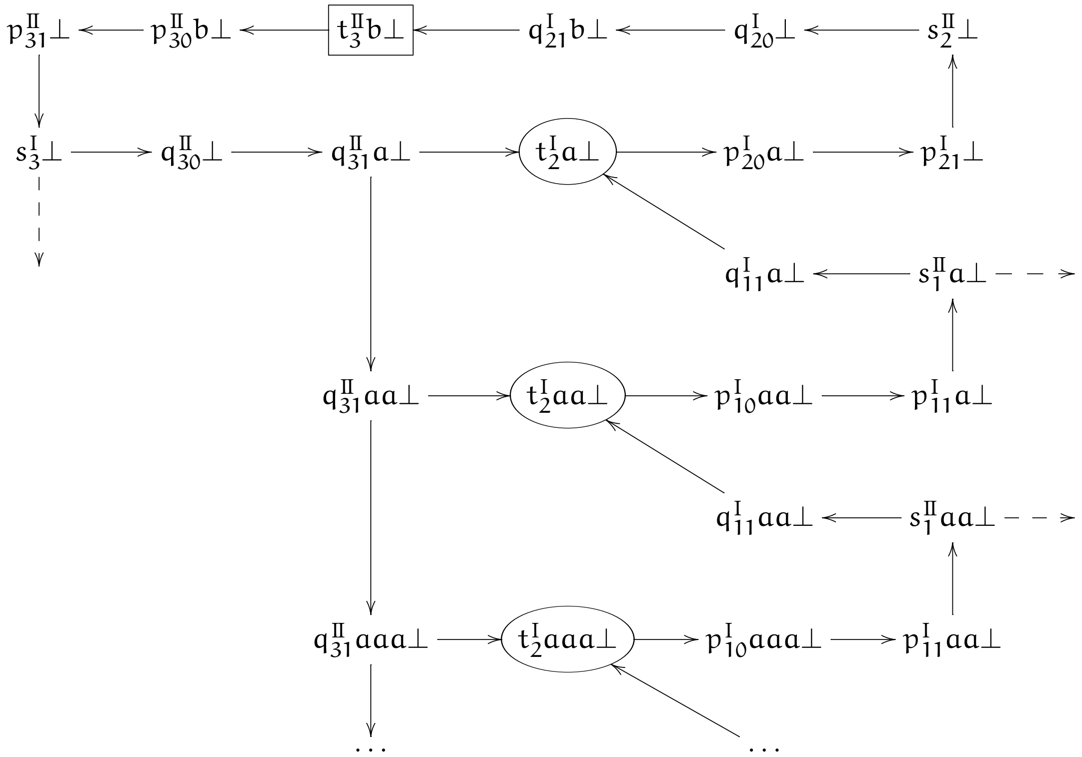
Let $\Gamma = \{a, b\}$, we consider the following PRG:

$$G \quad = \quad (a \hookrightarrow \epsilon)a^+ \ \cup \ (a \hookrightarrow b)\epsilon \ \cup \ (b \hookrightarrow a^+)\epsilon \ ,$$

writen here in the form $(U_1 \hookrightarrow V_1)W_1 \ \cup \ (U_2 \hookrightarrow V_2)W_2 \ \cup \ (U_3 \hookrightarrow V_3)W_3$, with $N = 3$. Let $max = 4$, $\Omega(a) = 2$, $\Omega(b) = 3$, $\Gamma_{\mathrm{I}} = \{a\}$, $\Gamma_{\mathrm{II}} = \{b\}$. The game graph is pictured here:



According to the above construction, a vertex a^i is represented in the PDS by $t_2^I a^i \bot$, and b is represented by $t_3^{II} b \bot$. The automaton recognizing U_i, V_i and W_i are very simple, we do not define them explicitely. We draw a part of the graph of the corresponding PDS:



Player I has a winning strategy from b in the PRG.

Discussion

It remains open how to apply the MSO-definability of a winning region (either for deciding the winner or for extracting a winning strategy). Another question is to develop a theory of game simulation which covers the examples of Sections 2 and 5.

Acknowledgment

Great thanks to Thomas Wilke and the referees of ICALP [1] for suggesting the idea of Section 4, to Igor Walukiewicz for helping me understand his paper, and to Wolfgang Thomas, Christof Löding and the referees of INFINITY for helpful remarks.

References

[1] T. CACHAT, *Symbolic Strategy Synthesis for Games on Pushdown Graphs*, accepted for ICALP'02, to appear in LNCS, available at `http://www-i7.informatik.rwth-aachen.de/~cachat/`.

[2] D. CAUCAL, *On infinite transition graphs having a decidable monadic theory*, ICALP'96, LNCS 1099, pp. 194–205, 1996.

[3] D. CAUCAL, *On the transition graphs of Turing machines*, 3rd MCU, LNCS 2055, pp. 177-189, 2001. Full version to appear in TCS, available at `http://www.irisa.fr/prive/caucal/biblio.html`.

[4] E. A. EMERSON and C. S. JUTLA, *Tree automata, mu-calculus and determinacy*, FoCS'91, IEEE Computer Society Press, pp. 368–377, 1991.

[5] E. A. EMERSON, C. S. JUTLA, and A. P. SISTLA, *On model-checking for fragments of* μ-*calculus*, CAV'93, LNCS 697, pp. 385–396, 1993.

[6] O. KUPFERMAN and M. Y. VARDI, *An Automata-Theoretic Approach to Reasoning about Infinite-State Systems*, CAV'00, LNCS 1855, pp. 36–52, 2000.

[7] M. Y. VARDI, *Reasoning about the past with two-way automata*, ICALP'98, LNCS 1443, pp. 628–641, 1998.

[8] J. VÖGE AND M. JURDZIŃSKI, *A discrete strategy improvement algorithm for solving parity games*, CAV'00, LNCS 1855 pp. 202–215, 2000.

[9] I. WALUKIEWICZ, *Pushdown processes: games and model checking*, CAV'96, LNCS 1102, pp. 62–74, 1996. Full version in Information and Computation 157, 2000.

Monotonic Extensions of Petri Nets : Forward and Backward Search Revisited

A. Finkel[1], J.-F. Raskin[2*†], M. Samuelides[1], L. Van Begin[2‡]
[1] LSV & ENS de Cachan
61 av. du Prsident Wilson, 94235 Cachan cedex,France
finkel@lsv.ens-cachan.fr
[2] Université Libre de Bruxelles
Blvd du Triomphe, 1050 Bruxelles, Belgium
{jraskin,lvbegin}@ulb.ac.be

Abstract

In this paper, we revisite the forward and backward approaches to the verification of extensions of infinite state Petri Nets. As contributions, we propose an efficient data structure to represent infinite downward closed sets of markings and to compute symbolically the minimal coverability set of Petri Nets, we identify a subclass of Transfer Nets for which the forward approach generalizes and we propose a general strategy to use both the forward and the backward approach for the efficient verification of general Transfer Nets.

1 Introduction

Model-checking techniques have proven useful for the verification of finite state abstractions of various concurrent and distributed computer systems. Unfortunately, useful finite state abstractions are often difficult to obtain from the concrete systems to verify. As a consequence, a lot of efforts have been made recently to extend the successful techniques for model-checking of finite state systems to infinite state systems and parametric verification. A lot of interesting theoretical results have been obtained, see for example [3, 6, 13, 16].

[*] This author was partially supported by a “Crédit aux chercheurs”, Belgian National Fund for Scientific Research.

[†] This author was partially supported by a FRFC grant 2.4530.02.

[‡] This author was supported by a Walloon Region grant ”First Europe”.

Nevertheless, a lot of work remains to be done to turn those positive theoretical results into practical verification algorithms.

In parametric verification, we want to verify at once an entire family of systems. For example, some mutual exclusion protocols have been designed to work for any number of processes that want to share common resources and the verification of such protocols for a specific number of process is not relevant. In this context, several abstraction have proven to be useful, see for example [1, 4, 17]. The work in this paper is directly connected to the context of the so-called *counting abstraction*. When considering the counting abstraction, the model of (infinite) *Petri Nets* and its extensions, like *Transfer Nets* and *Reset Nets*, are particularly important. In this paper, we will discuss efficient techniques to analyze infinite state Petri Nets and Transfer Nets (note that Reset Nets can be viewed as a subclass of Transfer Nets).

There are two main different approaches for the verification of Petri Nets. The first one is the *forward approach* (that was first defined by Karp and Miller in [20]). This approach starts from the (possibly parametric) set of initial markings and computes an approximation of the closure of the transition relation (often referred as the Post relation) over markings defined by the Petri net. That over-approximation is sufficiently precise to completely answer interesting questions about Petri Nets. One of the most important class of properties we can verify is a subclass of the so-called *safety properties*, i.e. ”can the Petri net ever reach a set of *bad markings*?”, with the restriction that the set of bad markings is *upward closed*. That result allows us *in theory* to automatically answer any *mutual exclusion* property for example. The *backward approach* represents an alternative to the forward approach for the verification of such properties. The backward approach consists in applying iteratively from the set of Bad markings the Pre relation (which is the inverse of the Post relation). If the closure of the Pre relation intersects with the set of initial markings then we know that some Bad markings are reachable. The application of the Pre relation is guaranteed to terminate if the set of Bad markings is upward closed [3, 16]. Unfortunately, in the two cases, a *naive* implementation of the abstract algorithm is not practical. It is not surprising as we know that the *theoretical complexity* of the reachability problem of upward closed sets (also called *coverability problem*) for that class of infinite state models is *very high*, see [22].

So, further research was necessary to obtain more practically useful verification techniques. For the forward approach, we have defined in [15] an heuristic to *minimize* the set of markings to consider when computing the Karp-Miller covering tree. For the backward approach, we have defined in [9] a BDD-like structure that allow us to *compactly* represent the infinite sets that are generated during the iteration of the Pre relation. The resulting algorithm for the backward search is *symbolic* in the sense that (minimal) markings are never enumerated during the computation and all the operations on sets involved in the algorithm are directly computed on the underlying compact structure that represents the sets. Practical evaluation of the algorithm has shown that it is much more effective than the naive *enumerative* approach, see [10] for details.

In this paper, as a first contribution, we show how to turn into a symbolic algorithm the enumerative algorithm that we have defined in [15] to compute the minimal coverability set of an infinite state Petri net. For this we use a variant of the data structure that we have defined in [9]. As we have now two symbolic algorithms (one for the forward search, one for the backward search) , we are able to make a *fair* comparison between the relative practical merits of the two approaches.

A main advantage of the backward search is its *robustness* in the following sense: it is not only applicable to the basic class of infinite Petri Nets but also to all the extensions that preserve *monotonicity* (see [3]). Transfer Nets (and so broadcast protocols that they generalize) and Reset Nets maintain monotonicity for example. Unfortunately, the forward approach does *not* generalize for those models (see [8, 13]) i.e. in those cases the search is not guaranteed to terminate. Indeed, there are negative results [8] that show us that it is not always possible to compute the coverability set for those extensions. Nevertheless, as a second contribution, we show in this paper that we can compute forwardly a weak version of the coverability set for an interesting subclass of Transfer Nets. That weak version allow us to decide the safety properties that can be expressed as upward closed sets of markings. This subclass is of practical interest as it covers all the examples of abstractions of multi-threated JAVA programs that we have analyzed backwardly in [12]. The advantage of the forward approach is that it starts from the set of initial markings and usually generate sets that are *more structured* than those generated with the backward search.

In [10], we have shown that the efficiency of the backward search can be improved substantially by using rough approximations of the forward reachable states in order to guide the search toward initial markings. In our previous works, the over-approximation is obtained automatically by computing the structural invariants of the net ([24]). As a third contribution, we propose here to use the symbolic implementation of the minimal coverability set for Petri nets to compute another over-approximation of the forward reachable states of general Transfer Nets that do not fall in the class of models that we have identified. If the over-approximation is *too large* to give an conclusive answer to the safety verification problem, we propose to use this over-approximation to guide the exact backward search. The information collected during this over-approximation is potentially (and often much) richer than the one computed with the invariants. Those heuristics seems necessary to attack those verification problems that have very high theoretical complexities [23].

Structure of the paper In section 2, we introduce the model of Multi Transfer Nets that contains Petri Nets as a subclass. We also recall some notions about upward closed sets, downward closed sets and covering sets. In section 3, we show how to represent efficiently infinite downward closed sets with a graph based data structure. Section 4 presents a symbolic algorithm to compute the minimal coverability tree of a Petri Net. We have implemented this symbolic algorithm and used our graph based structure to represent the downward closed sets it manipulates. We report in section 5 on the practical behavior of our new symbolic forward algorithm and compare its performances with a symbolic backward algorithm that we have defined and implemented in previous works. In section 6, we identify an interesting subclass of Multi Transfer Nets for which the forward search can be extended, we call this class the Multi Isolated Transfer Nets. In section 7, we suggest the cooperative use of a forward approximation and the backward search for the full class of Multi Transfer Nets.

2 Petri Nets and Multi Transfer Nets

In this section, we define Multi Transfer Nets (MTNs for short), an extension of Petri Nets with fairly general transfers. That extension maintains the monotonicity property of Petri Nets [1].

Definition 1 (Multi Transfer Net) A *Multi Transfer Net* is a pair $\langle \mathcal{P}, \mathcal{B} \rangle$ where: $\mathcal{P} = \{p_1, \ldots, p_n\}$ is a set of places, and $\mathcal{B} = \{M_1, \ldots, M_m\}$ is a set of *multi transfers*. A *multi transfer* M is a tuple $\langle T, \{B_1, \ldots, B_u\} \rangle$ such that

- $T = \langle \mathcal{I}, \mathcal{O} \rangle$ is the *Petri Net transition* part of the *multi transfer*: $\mathcal{I}, \mathcal{O} : \mathcal{P} \to \mathbb{N}$;
- each $B_i = \langle P_i, p_i \rangle$ with $P_i \subseteq \mathcal{P}$ (a set of source places) and $p_i \in \mathcal{P}$ (a target place) is a *transfer*.

In order to avoid cyclic transfers, a multi transfer M with set of transfers $\{B_1, \ldots, B_u\}$ must satisfy the following conditions:

1. for any B_i, we require that $p_i \notin P_i$;

[1] By monotonicity property, we mean that if a transition t can be fired in a marking $\mathbf{m}$, it can also be fired in any markings $\mathbf{m}'$ greater than $\mathbf{m}$.

2. for any *transfers* B_i and B_j with $B_i \neq B_j$, we require that $(P_i \cup \{p_i\}) \cap (P_j \cup \{p_j\}) = \emptyset$.

□

A Petri Net is a MTN where each multi transfer contains an empty set of transfer (then the multi transfer is just a plain Petri Net transition).

A marking $\mathbf{m} : \mathcal{P} \to \mathbb{N}$ is a function which assigns a value $c \in \mathbb{N}$ to each place. That function can equivalently be seen as a vector of size $|\mathcal{P}|$. We define $\preceq$ on markings such that $\mathbf{m} \preceq \mathbf{m}'$ iff $\mathbf{m}(p) \leq \mathbf{m}'(p), \forall p \in \mathcal{P}$. We say that $\mathbf{m}$ is covered by $\mathbf{m}'$ when $\mathbf{m} \preceq \mathbf{m}'$. We denote $\sum_{p \in P} \mathbf{m}(p)$ by $\mathbf{m}(P)$.

Definition 2 (Multi Transfer-Enabling) Let M be a multi-transfer with the Petri Net *transition* part $\langle \mathcal{I}, \mathcal{O} \rangle$. We say that M is *enabled* in $\mathbf{m}$ if $\mathcal{I} \preceq \mathbf{m}$. □

Definition 3 (Multi Transfer-Firing) Let $M = \langle T, \{B_1, \ldots, B_u\} \rangle$ be a multi transfer enabled in $\mathbf{m}$. *Firing* M in $\mathbf{m}$ leads to the marking $\mathbf{m}'$ (written $\mathbf{m} \rightarrowtail_M \mathbf{m}'$). To define $\mathbf{m}'$, we need to define the intermediary markings $\mathbf{m}_1$ and $\mathbf{m}_2$:

- $\mathbf{m}_1 = \mathbf{m} - \mathcal{I}$. That is, we first remove the tokens needed by the Petri Net part of the transition;
- then we define $\mathbf{m}_2$ as follows:
  - for any place p which is target of a transfer, i.e. $p = p_i$ for some $1 \leq i \leq u$, we have $\mathbf{m}_2(p) = \mathbf{m}_1(p) + \mathbf{m}_1(P_i)$. That is, we transfer all the remaining tokens from the sources to the target;
  - for any place p which is a source of a transfer, i.e. $p \in P_i$ for some $1 \leq i \leq u$, we have $\mathbf{m}_2(p) = 0$;
  - for all other places p (which are neither a source nor the target of a transfer), we have $\mathbf{m}_2(p) = \mathbf{m}_1(p)$;
- Finally, $\mathbf{m}'$ is obtained from $\mathbf{m}_2$ by adding the tokens produced by the Petri Net part of the transition, that is: $\mathbf{m}' = \mathbf{m}_2 + \mathcal{O}$.

□

Fig. 1 shows an example of MTN modeling the MESI protocol [18]. Dashed arrows represent transfer arcs and plain arrows represent classical Petri Nets arcs. When transition d is fired, one token is removed from the place invalid, then the tokens of the places shared, modified and exclusive are transferred to the place invalid and finally one token is put in exclusive. We now define the semantics of MTNs.

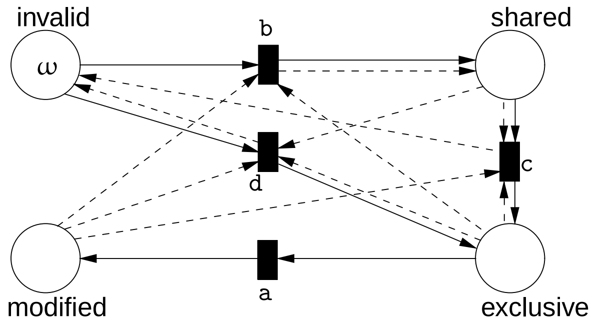


Figure 1: MTN of the MESI protocol.

Definition 4 (Operational Semantics) Let $\mathcal{M} = \langle \mathcal{P}, \mathcal{B} \rangle$ be an MTN. A *run* of $\mathcal{M}$ is a sequence of markings $\mathbf{m}_0\mathbf{m}_1 \ldots \mathbf{m}_n$ such that for any i, $0 \leq i < n$, there exists $M \in \mathcal{B}$ such that $\mathbf{m}_i \rightarrowtail_M \mathbf{m}_{i+1}$, $\mathbf{m}_0$ is the *initial* marking of the run and $\mathbf{m}_n$ the *target* marking of the run. A marking $\mathbf{m}'$ is *reachable* from a marking $\mathbf{m}$, written $\mathbf{m} \rightarrowtail^* \mathbf{m}'$, if and only if there exists a run with initial marking $\mathbf{m}$ and target marking $\mathbf{m}'$. The *set of reachable markings* of $\mathcal{M}$ from a set of markings S_0, written $\mathsf{Reach}(\mathcal{M}, S_0)$, is defined as the set $\{\mathbf{m}' \mid \exists \mathbf{m} \in S_0 : \mathbf{m} \rightarrowtail^* \mathbf{m}'\}$. □

Given a MTN $\mathcal{M}$, a set of initial markings S_0 and a set of Bad markings U, we are interested by the following two decision problems:

1. "Can the MTN $\mathcal{M}$ reach a Bad marking in U starting from an marking in S_0 ?", i.e. $\mathsf{Reach}(\mathcal{M}, S_0) \cap U =^? \emptyset$. This is called the *safety verification problem*.

2. "Is there a bound $c \in \mathbb{N}$ for the place p such that in any reachable marking of $\mathcal{M}$ starting from S_0, the number of tokens in p does not exceed c ?", i.e. $\exists c \in \mathbb{N} : \forall \mathbf{m} \in \mathsf{Reach}(\mathcal{M}, S_0) : \mathbf{m}(p) \leq c?$. [2] This is called the *place boundedness problem*.

Before going further, we need some more notations. We define a special value ω. For any $c \in \mathbb{N}$, we have $c < \omega$, and furthermore we have $\omega + c = \omega - c = \omega, \forall c \in \mathbb{N} \cup \{\omega\}$. We extend markings to ω-markings which assigns a value $c \in \mathbb{N} \cup \{\omega\}$ to each places. The $\preceq$ relation on markings is extended to ω-markings in the obvious way.

Definition 5 (Least Upper Bound) The *least upper bound (lub)* of a (possibly infinite) set of markings $\{\mathbf{m}_1, \mathbf{m}_2, \ldots\}$ is the marking $\mathbf{m}$ such that:

$$\begin{cases} \mathbf{m}(p) = \omega & \textbf{if } \forall c \in \mathbb{N} : \exists i \geq 1 : \mathbf{m}_i(p) > c \\ \mathbf{m}(p) = max(\{\mathbf{m}_1(p), \mathbf{m}_2(p), \ldots\}) & \textbf{otherwise}. \end{cases}$$

[2] Let us note that the weaker question "Is $c \in \mathbb{N}$ a bound for the place p ?" is a particular case of the safety verification problem.

□

We say that a set of markings S is *downward closed* iff we have

$$\forall \mathbf{m} : (\mathbf{m} \in S \rightarrow \forall \mathbf{m}' \preceq \mathbf{m} : \mathbf{m}' \in S).$$

Symmetrically, we say that a set of markings S is *upward closed* iff we have

$$\forall \mathbf{m} : (\mathbf{m} \in S \rightarrow \forall \mathbf{m}' \succeq \mathbf{m} : \mathbf{m}' \in S).$$

Given a upward closed set S, we note $\mathsf{Min}(S)$ the set of minimal elements defined as:

$$\mathsf{Min}(S) = \{\mathbf{m} \in S \mid \neg\exists \mathbf{m}' \in S : \mathbf{m}' \prec \mathbf{m}\}.$$

Given a downward closed set S, we note $\mathsf{Lim}(S)$ the set of its limits elements defined as:

$$\mathsf{Lim}(S) = \{\mathbf{m} \in S^{lim} | \neg\exists \mathbf{m}' \in S^{lim} : \mathbf{m} \prec \mathbf{m}'\} \cup \{\mathbf{m} \in S | \neg\exists \mathbf{m}' \in S : \mathbf{m} \prec \mathbf{m}'\}$$
$$\text{where } S^{lim} = \{\mathbf{m} | \mathbf{m} = lub(\{\ \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n, \ldots\}) \text{ with } \forall i \geq 1 : \mathbf{m}_i \in S \wedge \mathbf{m}_i \prec \mathbf{m}_{i+1}\}.$$

Given a set of ω-markings, we define its downward closure as follows:

Definition 6 (Downward Closure) Let S be a set of ω-markings, the *downward closure* of S, noted $\downarrow S$, is the set of markings $\{\mathbf{m} | \exists \mathbf{m}' \in S \text{ and } \mathbf{m} \preceq \mathbf{m}'\}$ □

In $\mathbb{N}^k$, any upward closed set S is identified by its finite set of minimal elements and any downward closed set S is identified by its finite set of limit elements (that are vectors potentially with ωs). We are now equipped to define the notion of *coverability set* which is an important tool to answer the decision problems that we have mentioned above.

Definition 7 (Coverability Set [15, 14]) A *coverability set* for a MTN $\mathcal{M}$ and a set S_0 of initial markings, noted $\texttt{CS}(\mathcal{M}, S_0)$, is a set of ω-markings such that :

(1) for every $\mathbf{m} \in \texttt{CS}(\mathcal{M}, S_0) \setminus \texttt{Reach}(\mathcal{M}, S_0)$, there exists an infinite sequence $\mathbf{m}_1 \prec \mathbf{m}_2 \prec \mathbf{m}_3 \ldots$ with for all $i \geq 1 : \mathbf{m}_i \in \texttt{Reach}(\mathcal{M}, S_0)$ and such that $\mathbf{m} = lub(\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \ldots\})$.

(2) $\forall \mathbf{m} \in \texttt{Reach}(\mathcal{M}, S_0)$, there exists $\mathbf{m}' \in \texttt{CS}(\mathcal{M}, S_0)$ such that $\mathbf{m} \preceq \mathbf{m}'$.

□

In all the coverability sets, there is an interesting one which is called the minimal coverability set.

Definition 8 (Minimal Coverability Set [15, 14]) The *minimal coverability set* of a Petri Net $\mathcal{M}$ for a set S_0 of initial markings is the intersection of all the finite coverability sets of $\mathcal{M}$ for S_0. □

We have shown in a previous work [15] that the minimal coverability set is unique and can be computed effectively for any Petri Net $\mathcal{M}$ and any finite set of initial ω-markings S_0. Let us now recall some properties of coverability sets, see [15] for details.

1. any $\mathtt{CS}(\mathcal{M}, S_0)$ is an approximation of the reachable markings in the following sense :

$$\mathtt{Reach}(\mathcal{M}, S_0) \subseteq \downarrow \mathtt{CS}(\mathcal{M}, S_0)$$

2. any $\mathtt{CS}(\mathcal{M}, S_0)$ is sufficient to answer any safety verification problem if the set of Bad markings U is upward closed. In fact, we have:

$$\mathtt{Reach}(\mathcal{M}, S_0) \cap U = \emptyset \textbf{ iff } \downarrow \mathtt{CS}(\mathcal{M}, S_0) \cap U = \emptyset$$

3. any $\mathtt{CS}(\mathcal{M}, S_0)$ is sufficient to give an answer to the place boundedness problem. A place p is bounded in $\mathtt{Reach}(\mathcal{M}, S_0)$ **iff** there does not exist a ω-marking $\mathbf{m} \in \mathtt{CS}(\mathcal{M}, S_0)$ such that $\mathbf{m}(p) = \omega$.

3 Downward Closed Covering Sharing Trees

The motivation of this section is to define a way to compactly represent (possibly infinite) downward closed sets of markings. We start from Sharing Trees (STs) that are data structures introduced in [25] to efficiently store tuples of integers. A sharing tree $\mathcal{S}$ is a rooted acyclic graph with nodes partitioned in k-*layers* such that: all nodes of layer i have successors in the layer $i+1$; a node cannot have two successors with the same label; finally, two nodes with the same label in the same layer do not have the same set of successors. Formally, $\mathcal{S}$ is a tuple $(N, V, root, end, val, succ)$, where $N = \{root\} \cup N_1 \cup \ldots \cup N_k \cup \{end\}$ is the finite set of *nodes* (N_i is the set of nodes of *layer* i and, by convention, $N_0 = \{root\}$ and $N_{k+1} = \{end\}$), $V = \{x_1, x_2, \ldots, x_k\}$ is a set of variables. Intuitively, N_i is associated to x_i. $val : N \to \mathbb{N} \cup \{\top, \bot\}$ is a labeling function for the nodes, and $succ : N \to 2^N$ defines the successors of a node. Furthermore, (1) $val(n) = \top$ iff $n = root$, $succ(root){=}N_1$, $val(n) = \bot$ iff $n = end$, $succ(end){=}\emptyset$; (2) for $i : 0, \ldots, k$, $\forall n \in N_i$, $succ(n) \subseteq N_{i+1}$ and $succ(n) \neq \emptyset$; (3) $\forall n \in N$, $\forall\ n_1, n_2 \in succ(n)$, if $n_1 \neq n_2$ then $val(n_1) \neq val(n_2)$. (4) $\forall i, 0 \leq i \leq k$, $\forall n_1, n_2 \in N_i$, $n_1 \neq n_2$, if $val(n_1) = val(n_2)$ then $succ(n_1) \neq succ(n_2)$. A path of a k-sharing tree is a sequence of nodes $\langle \top, n_1, \ldots, n_k, \bot \rangle$ such that $n_{i+1} \in succ(n_i)$ for $i = 1, \ldots, k$-1. Paths represent *tuples of size* k of natural numbers. We use *elem*$(\mathcal{S})$ to denote the *flat denotation* of a k-sharing tree $\mathcal{S}$:

$$\mathit{elem}(\mathcal{S}) = \{\ \langle val(n_1), \ldots, val(n_k) \rangle \mid \langle \top, n_1, \ldots, n_k, \bot \rangle \text{ is a path of } \mathcal{S}\ \}.$$

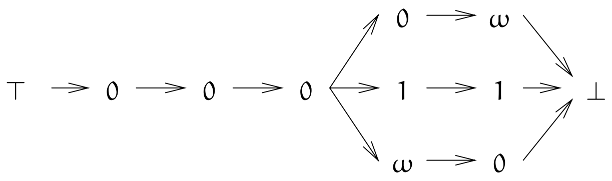


Figure 2: Example of a dcCST that represents an infinite downward closed set.

Conditions (3) and (4) ensure the maximal sharing of prefixes and suffixes among the tuples of the flat denotation of a sharing tree. The *size* of a sharing tree is the number of its *nodes* and *edges*. The number of tuples in $elem(\mathcal{S})$ can be exponentially larger than the size of $\mathcal{S}$. As shown in [25], given a set of tuples $\mathcal{A}$ of size k, there exists a unique sharing tree such that $elem(\mathcal{S}_{\mathcal{A}}) = \mathcal{A}$ (modulo isomorphisms of graphs). Given two finite sets of tuples of integers represented as two STs $\mathcal{S}_1$ and $\mathcal{S}_2$, there are polynomial time algorithms in the size of the two STs to compute $\mathcal{S}_3$ such that $\mathcal{S}_3 = \mathcal{S}_1 \cap \mathcal{S}_2$, $\mathcal{S}_3 = \mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_3 = \mathcal{S}_1 \setminus \mathcal{S}_2$ and a polynomial algorithm to decide if $\mathcal{S}_1 \subseteq \mathcal{S}_2$. But STs can only represent finite sets. In [11], we have proposed to use an extension of that data-structure to represent infinite upward closed sets of markings. We adapt here this idea in order to represent (potentially infinite) downward closed sets of markings. We know that a (potentially infinite) downward closed set of markings is identified by its finite set of limit elements. This set is a finite set of ω-markings. We will use ST to represent this finite set of ω-markings. We define downward closed covering ST as ST with $val : N \to \mathbb{N} \cup \{\top, \bot\} \cup \{\omega\}$. A downward closed covering ST $\mathcal{S}$ has the following semantics :

$$[\![\mathcal{S}]\!] = \{\mathbf{m} \in \mathbb{N}^k | \exists \mathbf{m}' \in elem(\mathcal{S}) : \mathbf{m} \preceq \mathbf{m}'\}$$

So, $[\![\mathcal{S}]\!]$ is the downward closure of $elem(\mathcal{S})$, i.e. $[\![\mathcal{S}]\!] = \downarrow elem(\mathcal{S})$. We will say that a downward closed covering ST (dcCST) $\mathcal{S}$ is irredundant if $\neg\exists \mathbf{m}_1, \mathbf{m}_2 \in elem(\mathcal{S})$ with $\mathbf{m}_1 \prec \mathbf{m}_2$. In our experience, it is often better to keep $\mathcal{S}$ irredundant. An example of dcCST is given in Fig. 2. It represents the infinite downward closed set $\{\mathbf{m} \in \mathbb{N}^5 | \mathbf{m} \preceq \langle 0,0,0,0,\omega\rangle \vee \mathbf{m} \preceq \langle 0,0,0,1,1\rangle \vee \mathbf{m} \preceq \langle 0,0,0,\omega,0\rangle\}$. Note that the dcCST encodes efficiently this infinite downward closed set as its limits elements share large prefixes. It is easy to show with straightforward adaptations of proofs from [11] (where we define CST to represent infinite upward-closed sets) that (i) there is no polynomial time algorithm (unless P = NP) to decide $[\![\mathcal{S}_1]\!] \subseteq [\![\mathcal{S}_2]\!]$ where $\mathcal{S}_1, \mathcal{S}_2$ are two dcCSTs, (ii) there is no polynomial time algorithm (unless P = NP) to compute from $\mathcal{S}_1$ an irredundant dcCST $\mathcal{S}_2$ such that $[\![\mathcal{S}_1]\!] = [\![\mathcal{S}_2]\!]$. We will show that those negative theoretical results seems not to be a practical obstacle to the use of dcCST in a symbolic algorithm computing the minimal coverability set. The algorithm using dcCSTs is given in the next section. Practical evaluation of the algorithm is given in section 5.

4 Symbolic Computation of the Minimal Covering Set for PN

To compute a coverability set of a Petri Net, we generally construct what we call a *coverability tree* (see [20]). This is mainly a tree where the nodes are labeled by the elements of a coverability set and the edges are an approximation of the successor relation between the ω-markings labeling the nodes. Unfortunately, the procedure presented in [20] computes unmanageable trees, even for small Petri Nets. An efficient heuristic is presented in [15]. This algorithm construct the minimal coverability tree, which is a tree where the values of the nodes correspond to the limit elements of the minimal coverability set of the Petri Net. The main idea of the algorithms of [20, 15] is to use an acceleration function f^a when a marking $\mathbf{m}$ is accessible from a markings $\mathbf{m}'$ in the tree with $\mathbf{m}' \prec \mathbf{m}$. The definition of f^a is as follows :
$f^a(\mathbf{m}', \mathbf{m}) = \mathbf{m}''$ such that

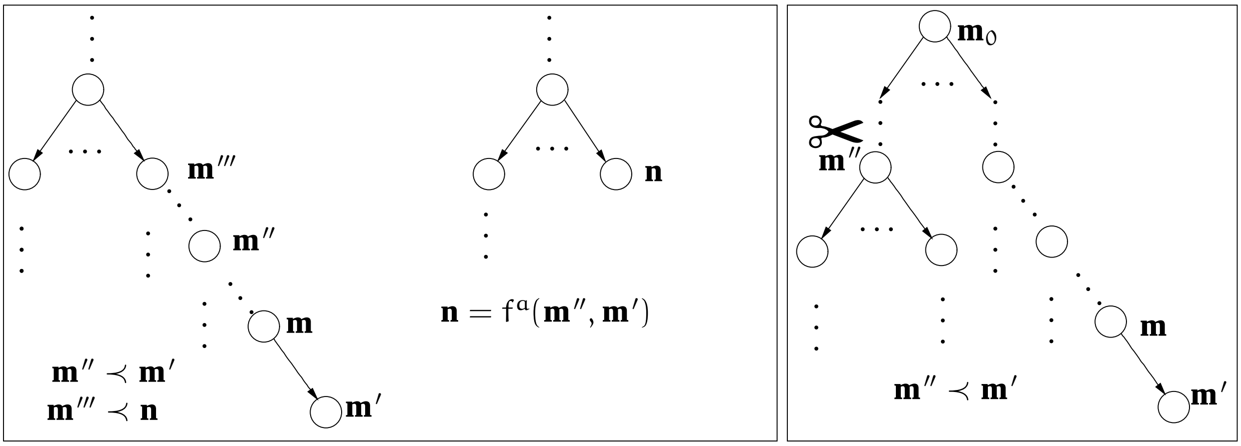
$$\begin{cases} \mathbf{m}''(p) = \mathbf{m}'(p) & \textbf{if } \mathbf{m}(p) = \mathbf{m}'(p) \\ \mathbf{m}''(p) = \omega & \textbf{if } \mathbf{m}(p) > \mathbf{m}'(p) \end{cases}$$

More precisely, the algorithm of [15] works as follows. At each step of the construction of the tree, an untreated node annotated with the ω-marking $\mathbf{m}$ is developed by computing its successors (at the beginning, untreated nodes correspond to initial markings). For each successors $\mathbf{m}'$ of $\mathbf{m}$, we have three cases :

(1) there is an already computed ω-marking $\mathbf{m}''$ such that $\mathbf{m}' \preceq \mathbf{m}''$, then $\mathbf{m}'$ is forgotten.

(2) there is at least one path in the tree from a ω-marking $\mathbf{m}''$ to $\mathbf{m}$ such that $\mathbf{m}'' \prec \mathbf{m}'$, then we take the largest such path and constructs $\mathbf{n} = f^a(\mathbf{m}'', \mathbf{m}')$. Finally, we take the farest predecessor $\mathbf{m}'''$ of $\mathbf{m}$ with $\mathbf{m}''' \prec \mathbf{n}$ (possibly different from $\mathbf{m}''$) and replace the subtree rooted by $\mathbf{m}'''$ by $\mathbf{n}$ (see Fig. 3(a)). We finally remove all the subtrees rooted by $\mathbf{m}''$ with $\mathbf{m}'' \prec \mathbf{n}$ (see Fig. 3(b)).

(3) In the other cases, a new node connected to the node of $\mathbf{m}$ and annoted with $\mathbf{m}'$ is added to the tree. As in the previous case, all the subtrees rooted by a marking $\mathbf{m}''$ such that $\mathbf{m}'' \prec \mathbf{m}'$ are removed.


$\mathbf{m}'''$
$\mathbf{m}''$
$\mathbf{m}$
$\mathbf{m}'$
$\mathbf{m}'' \prec \mathbf{m}'$
$\mathbf{m}''' \prec \mathbf{n}$
$\mathbf{n}$
$\mathbf{n} = f^a(\mathbf{m}'', \mathbf{m}')$
$\mathbf{m}_0$
$\mathbf{m}''$
$\mathbf{m}$
$\mathbf{m}'$
$\mathbf{m}'' \prec \mathbf{m}'$

(a) (b)

Figure 3: operations on trees

An improvement of this algorithm is to symbolically treat the sets of ω-markings instead of enumerating them. Symbolically means that the computations on sets are done by making some global computations on the data structures used to represent them. To make this possible, we first define a set-based version of the algorithm of [15]. Fig. 4 shows such an high-level algorithm manipulating sets. In [15], the minimal coverability tree is constructed to manage case (2) and (3). But note that, for those cases, we only need the reachability relation between ω-markings rather than the direct successor relation of the tree. For this reason, we do not compute a coverability tree in the symbolic algorithm: we do not maintain the successor relation but only the closure of that relation. It is easy to see that this is sufficient for the case (2) and case (3) of the previous algorithm when we want to answer reachability of an upward closed set of markings or place boundness[3].

All the operations of the enumerative algorithm are replaced by operations on downward closed sets. The symbolic algorithm computing the minimal coverability set of a Petri net is given in figure 4 and works as follows. The algorithm maintains a set F of ω-markings that are untreated (this is the frontier of the search), a set S of ω-markings that contains the nodes already treated, and R^+ is a set of pairs of ω-markings $\langle \mathbf{m}, \mathbf{m}' \rangle$ such that $\mathbf{m}, \mathbf{m}' \in S \cup F$ and $\mathbf{m} \rightarrowtail^* \mathbf{m}'$. All the sets of ω-markings or pairs of ω-markings are represented using dcCSTs and all the operations of the algorithm are symbolic in the sense that they all works

[3] Note that the construction of the coverability graph allow us to answer to the *regularity problem* (is the language of the net regular). As the elements of a coverability set are the nodes of a coverability graph, such a graph is constructed by adding edges between the elements of the set by simply applying the Post operation on each of them.

```
1:  function MinimalCoverabilitySet ($\langle \mathcal{P}, \mathcal{B} \rangle; S_0$) return $S$
2:      $F \longleftarrow S_0$
3:      $S \longleftarrow S_0$
4:      $R^+ \longleftarrow \emptyset$
5:      while $F \neq \emptyset$ do
6:          $Succ \longleftarrow \{\mathbf{m}' | \exists \mathbf{m} \in F, M \in \mathcal{B} : \mathbf{m} \rightarrowtail_M \mathbf{m}'\}$
7:          $Succ \longleftarrow max(Succ) \setminus \{\mathbf{m} | \exists \mathbf{m}' \in S : \mathbf{m} \preceq \mathbf{m}'\}$
8:          $R^+ \longleftarrow R^+ \cup \{\langle \mathbf{m}, \mathbf{m}' \rangle | \mathbf{m} \in F \wedge \mathbf{m}' \in Succ \wedge \exists M \in \mathcal{B} : \mathbf{m} \rightarrowtail_M \mathbf{m}'\}$
9:          $R^+ \longleftarrow R^+ \cup \{\langle \mathbf{m}, \mathbf{m}' \rangle | \mathbf{m}' \in Succ \wedge \exists \mathbf{m}'' \in F, M \in \mathcal{B} : \mathbf{m}'' \rightarrowtail_M \mathbf{m}' \wedge \langle \mathbf{m}, \mathbf{m}'' \rangle \in R^+\}$
10:         $Succ', R^{+\prime} \longleftarrow$ Acc($S, Succ, R^+$)
11:         $S \longleftarrow S \cup Succ'$
12:         $S \longleftarrow max(S)$
13:         $F \longleftarrow max(Succ')$
14:         $R^{+\prime} \longleftarrow R^+ \cap (S \times S)$
15:     endwhile
```

Figure 4: Symbolic Algorithm that computes the minimal coverability set for a Petri net $\langle \mathcal{P}, \mathcal{B} \rangle$ and a set of initial ω-marking S_0.

directly on the structure of the dcCSTs representing the sets. All the algorithms on dcCSTs are easy adaptations of algorithms on CSTs that we have defined in [11].

At each iteration of the loop (line 5), the following operations are performed. In line (6), the set of the new reachable ω-markings ($Succ$) is computed and only the maximal elements of this set that are not covered by an ω-marking computed in previous iterations are kept (line 7). Lines 8 and 9 update R^+ for those markings. In line 10, we compute the accelerations (see description of the function Acc). We then add the successor ω-markings of the frontier that are obtained after acceleration to the set S of ω-markings computed so far (line 11). In line 12, we suppress from S all the ω-markings that are not maximal. As the new frontier we only consider the maximal elements computed during the current iteration (possibly accelerated) (line 13). After the minimization of S, the relation R^+ is updated.

The acceleration function is given in Fig. 5 and works as follows. It takes as arguments the set of new reachable ω-markings ($Succ$), the set of reachable ω-markings computed in the previous iterations (S) and the accessibility relations on all those ω-markings (R^+). First, the set of pairs that have to be accelerated is computed (line 2). The first component of those pairs are called source of the acceleration, the second one is called the target of the acceleration and the result of the acceleration is called the accelerated ω-marking. The arcs between the source of the acceleration and the accelerated ω-marking are computed (line

3), the arcs between a predecessors of a source of an acceleration and the corresponding accelerated ω-marking are computed (line 4) and finally, the arcs between the successors of a source of an acceleration that are predecessors of the target of the acceleration are linked to the accelerated ω-marking (line 5). R^+ is adjusted by adding all those arcs and the set of successors is adjusted by adding all the accelerated elements (lines 7-8).

```
1:  function Acc ($S$;$Succ$;$R^+$) return $Succ'$,$R^{+\prime}$
2:      $G \longleftarrow \{\langle \mathbf{m}, \mathbf{m}' \rangle \in R^+ : \mathbf{m} \prec \mathbf{m}'\}$
3:      $H_1 \longleftarrow \{\langle \mathbf{m}, \mathbf{m}' \rangle | \exists \mathbf{m}'' : \langle \mathbf{m}, \mathbf{m}'' \rangle \in G \wedge \mathbf{m}' = f^a(\mathbf{m}', \mathbf{m}'')\}$
4:      $H_2 \longleftarrow \{\langle \mathbf{m}, \mathbf{m}' \rangle | \exists \langle \mathbf{m}''', \mathbf{m}'' \rangle \in G : \mathbf{m}' = f^a(\mathbf{m}''', \mathbf{m}'') \wedge \langle \mathbf{m}, \mathbf{m}''' \rangle \in R^+\}$
5:      $H_3 \longleftarrow \{\langle \mathbf{m}, \mathbf{m}' \rangle | \exists \langle \mathbf{m}''', \mathbf{m}'' \rangle \in G : \mathbf{m}' = f^a(\mathbf{m}''', \mathbf{m}'') \wedge \langle \mathbf{m}''', \mathbf{m} \rangle \in R^+ \wedge \langle \mathbf{m}, \mathbf{m}'' \rangle \in R^+\}$
6:      $H \longleftarrow H_1 \cup H_2 \cup H_3$
7:      $R^{+\prime} \longleftarrow R^+ \cup H$
8:      $Succ' \longleftarrow Succ \cup \{\mathbf{m} | \exists \mathbf{m}' : \langle \mathbf{m}', \mathbf{m} \rangle \in H\}$
```

Figure 5: Function that accelerates all the accelerable cycles.

As previously explained, the size of dcCST can be logarithmic in the size of the set of limit elements it represents. In this way, we can potentially have an exponential gain both in memory usage and execution time using dcCST to represent sets of ω-markings and the closure of the transition relation.

5 Comparison With Backward Approach

5.1 Conceptual Comparison

In this section, we recall some facts about the forward and backward approach for the verification of infinite states Petri Nets and their monotonic extensions.

Forward	**Backward**
Downward closed Sets	Upward closed Sets
Over-approximation of successors	Exact set of predecessors
Acceleration	No Acceleration
Not Robust	Robust
Safety, Place boundedness	Safety
Depends on initial markings	Depends on bad markings

Figure 6: Conceptual differences between forward and backward search.

Sets. Starting from an upward closed set, the application of Pre preserves the upward closure of the set. So, the backward search manipulates upward closed sets. As we have seen, a coverability set is the set of limit elements of a downward closed set of markings that over-approximates the set of reachable markings. So, forward approach manipulates downward closed sets.

Approximation of the computation. The backward approach computes the exact set of predecessors of an upward closed set of bad markings and the forward approach computes an approximation of the reachable markings that is still precise enough to verify upward closed safety properties.

Techniques to guarantee termination. By applying the Pre operation, it is guaranteed to reach a fixpoint after a finite number of iterations. Forward approach needs an acceleration function to reach a fixpoint when the net is unbounded.

Robustness. Backward approach is robust for extensions of Petri Nets that maintain monotonicity of the model, on the other hand forward approach cannot always be extended for those natural extensions of Petri Nets.

Properties. Only covering properties can be decided with the backward algorithm but in addition *place boundness* can be solved with the forward approach.

Dependence of the search. When computing the coverability tree, we start the construction of the tree from the set of initial markings and the tree does not depend on the property that we want verify. On the other hand, with the backward approach, the computation depends on the property to verify. Note that if $\mathsf{Pre}^*(U)$ does not intersect with the set of initial markings then no set computed during the fixpoint computation contains any reachable marking.

5.2 Practical Comparison

We have applied our symbolic forward algorithm on a set of parametrized Petri Nets [4]. The results of the experiments are shown in Fig. 7 and compared with the results obtained using our symbolic backward algorithm defined in [10]. We have run the backward algorithm with

[4] see the web page `http://www.ulb.ac.be/di/ssd/lvbegin/CST/index.html` for a detailed description of the examples.

and without the invariant heuristic of [10]. The invariant heuristic computes *structural invariants* of the Petri net and uses them to prune the backward search: every minimal element defining an upward closed set that does not intersect with the set of solutions of the structural invariants is suppressed. This heuristic is safe as the set of solutions of the structural invariants over-approximates the reachable markings of the Petri net. Our set of examples is composed by some concurrent and production systems as the multipoll ([21]), the mesh2x2 ([2]) and its extension to 3x2 case, the flexible manufacturing system (FMS) of [7], the central server model (CSM) of [2] and the PNCSA protocol analysed in [5, 15].

As we can see from the figures, the forward search is always faster than the backward search. This is due to the fact that the behavior of the Petri net is much more regular when executed from its initial markings than when executed backwardly from a possible non reachable set of markings. As we can see for the PNCSA$_1$ and PNCSA$_2$ examples where we verify two different properties, efficiency of the backward search can depend a lot on the property that we want to verify. The invariant heuristic is very useful to obtain reasonable execution times in the backward search, this confirms the observation that getting some information about (an over-approximation of) the reachable markings is important.

Case Study	P	T	NN_{Post}	NE_{Post}	I_{Post}	EX_{Post}	M_{Post}	NN_{Pre}	NE_{Pre}	I_{Pre}	EX^1_{Pre}	M_{Pre}	EX^2_{Pre}
PNCSA$_1$	31	36	100	80	56	1s	2336kb	176	89	17	16.42s	3720kb	2.5s
PNCSA$_2$	31	36	100	80	56	1s	2336kb	368	132	38	>4h	48620kb	43.79s
CSM	14	13	35	16	10	0.02s	1472kb	109	152	11	0.1s	2116kb	0.07s
FMS	22	20	36	24	16	0.08s	1712kb	881	1695	46	6.13s	5680kb	5.98s
mesh2x2	32	32	78	256	8	0.13s	1784kb	344	427	15	0.8s	2600kb	0.8s
mesh3x2	52	54	106	6400	10	0.48s	2176kb	987	2224	21	6.5s	4992kb	6.5s
multipool	18	21	104	220	11	0.14s	1676kb	196	5641	18	2.1s	2516kb	-

Figure 7: Benchmarks on an AMD Athlon 900Mhz 500Mbytes : P=No. of places; T = No. of transitions; NN_{Post} (NN_{Pre}) = nodes of the sharing tree describing the minimal coverability set (the backward fixpoint); NE_{Post} (NE_{Pre}) = cardinality the minimal coverability set (the backward fixpoint); I_{Post} (I_{Pre}) =No. of iterations of the forward (backward) algorithm to reach the fixpoint; EX_{Post}(EX^1_{Pre}) = Execution time to reach the forward (backward) fixpoint ; EX^2_{Pre} = Execution time to reach the backward fixpoint using structural invariants;

6 A Weaker Notion of the Minimal Coverability Set

As we have seen in the last section, there are conceptual advantages to use the backward approach and practical evidences that plead for the forward approach. Unfortunately, we know that it is not always possible to compute a coverability set for natural extensions of

Petri Nets [8]. In this section, we identify a subclass of MTNs for which we can compute a weaker notion of coverability set. We call this class Multi Isolated Transfer Nets as the restriction is that any two transfers do not share places. This class is of practical importance as it covers all the examples of abstraction of JAVA programs that we have analyzed with the backward approach in [12].

This section is organized as follows. We first formally define the subclass of Multi Isolated Transfer Nets. Then we define the weaker notion of coverability set that we can construct for this class of systems. We then define an algorithm to compute this weak coverability set and illustrate its behavior on a simple example.

6.1 Multi Isolated Transfer Nets

Given a multi transfer $M = \langle T, \{B_1, B_2, \ldots, B_u\}\rangle$, we use the notation $\mathsf{Transfer}(M)$ to designate its set of transfers $\{B_1, B_2, \ldots, B_u\}$. Remember that each B_i is a pair $\langle P_i, p_i\rangle$, where P_i is the set of sources and p_i is the target of the transfer B_i.

Definition 9 A *Multi Isolated Transfer Net* $\mathcal{M} = \langle \mathcal{P}, \mathcal{B}\rangle$ is a MTN satisfying the following additional conditions, expressed informally as :

(i) a place cannot be a source of two different transfers;

(ii) a place cannot be the target of one transfer and source in another one;

(iii) a place source of a transfer cannot be source of the Petri Net part of a multi transfer.

and formally as follows:

(i) $\forall p \in \mathcal{P} : \neg\exists M_i, M_j \in \mathcal{B}$ with $\langle P_i, p_i\rangle \in \mathsf{Transfer}(M_i)$, $\langle P_j, p_j\rangle \in \mathsf{Transfer}(M_j)$, $(\langle P_i, p_i\rangle \neq \langle P_j, p_j\rangle)$, $p \in P_i$ and $p \in P_j$.

(ii) $\neg\exists p \in \mathcal{P}$ such that $\exists M_i, M_j \in \mathcal{B}$ with $\langle P_i, p_i\rangle \in \mathsf{Transfer}(M_i)$, $\langle P_j, p_j\rangle \in \mathsf{Transfer}(M_j)$ and $p \in P_i$ and $p = p_j$.

(iii) $\forall M_i \in \mathcal{B}, \forall\langle P_i, p_i\rangle \in \mathsf{Transfer}(M_i), \forall p \in P_i : \neg\exists M_j \in \mathcal{B}$ with $M_j = \langle\langle \mathcal{I}, \mathcal{O}\rangle, B\rangle$ and $\mathcal{I}(p) > 0$.

□

As an illustration of Multi Isolated Transfer Net, consider Fig. 8. We now define a weak notion of coverability set that is parametrized by a upward closed set U.

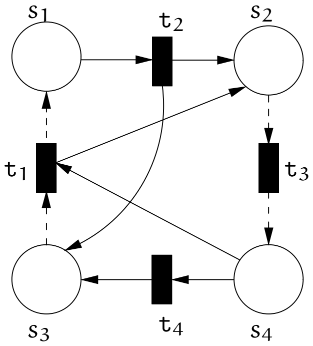


Figure 8: An example of Multi Isolated Transfer Net.

Definition 10 (Weak Coverability Set) Given a MTN $\mathcal{M}$, a set of initial ω-markings S_0 and an upward closed set U defined by an unique minimal marking $\mathbf{m}_U$ [5], a weak coverability set of $\mathcal{M}$ according to S_0 and $\mathbf{m}_U$ (noted $\mathtt{WCS}(\mathcal{M}, S_0, \mathbf{m}_U)$) is a set of ω-markings such that the two following conditions holds:

(1) $\forall \mathbf{m} \in \mathtt{WCS}(\mathcal{M}, S_0, \mathbf{m}_U) \setminus \mathtt{Reach}(\mathcal{M}, S_0)$, there exists an infinite sequence $\mathbf{m}_1 \preceq \mathbf{m}_2 \preceq \ldots \preceq \mathbf{m}_n \preceq \ldots$ with $\mathbf{m}_i \in \mathtt{Reach}(\mathcal{M}, S_0)$ for any $i \geq 1$ and such that:

- if $\mathbf{m}(p) = \omega$ then either $\mathbf{m}_i(p) > \mathbf{m}_{i-1}(p)$, for all $i > 1$, or $\mathbf{m}_i(p) \geq \mathbf{m}_U(p)$, for all $i \geq 1$.
- if $\mathbf{m}(p) \neq \omega$, we have for any $i \geq 0 : \mathbf{m}_i(p) = \mathbf{m}_{i+1}(p)$ and $\mathbf{m}(p) = \mathbf{m}_0(p)$.

(2) $\forall \mathbf{m} \in \mathtt{Reach}(\mathcal{M}, S_0)$, there exists $\mathbf{m}' \in \mathtt{WCS}(\mathcal{M}, S_0, \mathbf{m}_U)$ with $\mathbf{m} \preceq \mathbf{m}'$.

□

A weak coverability set over-approximates the set of reachable states and all the coverability sets. Nevertheless, it is still sufficiently precise to verify upward closed safety properties. On the contrary, it cannot always be used to decide place boundedness. This is formally expressed by the following proposition:

Proposition 1 *Let $\mathcal{M}$ be a MTN, an initial marking S_0 and an upward-closed set U with the minimal marking $\mathbf{m}_U$, the following holds :*

$$\mathtt{Reach}(\mathcal{M}, S_0) \subseteq\downarrow \mathtt{CS}(\mathcal{M}, S_0) \subseteq\downarrow \mathtt{WCS}(\mathcal{M}, S_0, \mathbf{m}_U)$$
$$\downarrow \mathtt{WCS}(\mathcal{M}, S_0, \mathbf{m}_U) \cap U \neq \emptyset \textbf{ iff } \mathtt{Reach}(\mathcal{M}, S_0) \cap U \neq \emptyset$$

□

[5] This restriction is to simplify the presentation, the extension to a finite set of minimal markings (and so to any upward closed set) is not difficult.

6.2 Algorithm to Compute a `WCS`

In this section, we give an algorithm that computes a weak coverability set for a Multi Isolated Transfer Net $\mathcal{M}$, a set of initial markings and an upward closed set U, defined by $\mathbf{m}_U$. The algorithm is presented as an extension of the enumerative algorithm given in section 4. At each step of the construction of the tree, an untreated node annoted with the ω-marking $\mathbf{m}$ is developed by computing its successors (at the beginning, untreated nodes corresponds to initial markings). For each successors $\mathbf{m}'$ of $\mathbf{m}$ by firing M, we have three cases as in the algorithm of section 4 for computing the minimal coverability set of a Petri Net. Cases (1), (3) are identical to the enumerative algorithm of section 4, we only detail case (2) that define the following acceleration (that we call `AccIsolated`) adapted to our subclass of Transfer Nets.

(2) if there is a sequence of transitions σ in the tree from a marking $\mathbf{m}''$ to $\mathbf{m}$ such that $\mathbf{m}'' \prec \mathbf{m}'$ (so, we have $\mathbf{m}'' \rightarrowtail_\sigma \mathbf{m} \rightarrowtail_M \mathbf{m}'$), then we construct $\mathbf{n}$ as follows :
we have $\mathbf{n}(p) = \omega$ **if** $\exists$ a path from $\mathbf{m}''$ to $\mathbf{m}$ corresponding to the sequence of transition σ with $\mathbf{m}'' \prec \mathbf{m}'$ and

(1) p is not the source or the target of a transition in $\sigma \cdot M$ and the marking strictly increases from $\mathbf{m}''$ to $\mathbf{m}$. More formally:

$$\forall \langle P, p_t\rangle \in \mathsf{Transfer}(\sigma \cdot M) : p \notin (\{p_t\} \cup P) \text{ and } \mathbf{m}''(p) < \mathbf{m}'(p)$$

(2) p is the target of a transfer in $\sigma \cdot M$ and either the marking of p increases strictly from $\mathbf{m}''$ to $\mathbf{m}'$ or there is a source of that transfer that increases strictly from $\mathbf{m}''$ to $\mathbf{m}'$. More formally:

$$\exists \langle P, p_t\rangle \in \mathsf{Transfer}(\sigma \cdot M) \text{ with } (p = p_t) \text{ and}$$

$$(\mathbf{m}''(p) < \mathbf{m}'(p) \vee \exists p' \in P : \mathbf{m}''(p') < \mathbf{m}'(p'))$$

(3) p is a source of a transfer in $\sigma \cdot M$ and the marking of p increases strictly from $\mathbf{m}''$ to $\mathbf{m}'$, and furthermore the marking of p is greater or equal to $\mathbf{m}_U(p)$. More formally:

$$\exists \langle P, p_t\rangle \in \mathsf{Transfer}(\sigma \cdot M) \text{ with } p \in P \text{ and } \mathbf{m}''(p) < \mathbf{m}'(p) \text{ and } \mathbf{m}(p) \geq \mathbf{m}_U(p)$$

otherwise, $\mathbf{n}(p) = \mathbf{m}'(p)$.
Finally, we take the farest predecessor $\mathbf{m}'''$ of $\mathbf{m}$ with $\mathbf{m}''' \prec \mathbf{n}$ (possibly different from $\mathbf{m}''$) and replace the subtree rooted by $\mathbf{m}'''$ by $\mathbf{n}$.

Note that as we consider Multi Isolated Transfer Net, those three cases are mutually exclusive. The following theorem states the correction of the algorithm defined with the acceleration above :

Theorem 1 *If $\mathcal{M}$ is a Multi Isolated Transfer Net, S_0 a finite set of initial ω-marking, $\mathbf{m}_U$ a marking defining a non-empty upward closed set U, then the algorithm defined in section 4 with the acceleration* `AccIsolated`*, terminates and computes a weak coverability set.*

Due to space limitation, we omit the formal proof of this statement and give the main idea that underlies the proof of correctness. The only particularity of the acceleration defined above is for places involved in a transfer. Let us take the case of a target place t. In the definition of the accelaration, we put ω in two cases : (1) when t has strictly increased, (2) if one of its sources has strictly increased. Case (1) is classically justified by monotonicity of the model, i.e. repeating $\sigma \cdot M$ will add at least the same number of tokens in t each time (remember that t is not the source of any transfer). Case (2) is justified as follows: t is not the source of any transfer, so the sequence $\sigma \cdot M$ takes out of t a constant number of tokens each time $\sigma \cdot M$ is fired. Furthermore, as a source of t strictly increased between $\mathbf{m}''$ and $\mathbf{m}'$ and this place is not the source of another transfer, when repeating $\sigma \cdot M$, from $\mathbf{m}'$, we know that the target will increase strictly and then we simply apply the justification of case (1). For the sources, the justification is as follows. When a source increases strictly between $\mathbf{m}''$ and $\mathbf{m}'$, we are not sure that it will increase beyond any bound, but we know that it will not decrease when repeating $\sigma \cdot M$. So if its value is greater or equal to $\mathbf{m}_U(s)$, we know that it will stay so. By putting ω in s, we do not take a too rough approximation for the following two reasons. First, putting ω in s is safe w.r.t. the intersection with U, that is $\mathbf{n}$ has an intersection with U iff $\mathbf{m}'$ and all the marking reachable from $\mathbf{m}'$ by iterating $\sigma \cdot M$ has an intersection with U. Second, it is easy to see that if we put ω in a source s, then there is also a ω in the target t of the transfer, and this ω will never leave the place t as we know that t is not the source of a transfer. As a consequence, the ω in s is guaranteed not to "propage" unsafely in the net.

We are planning to extend our symbolic implementation of the algorithm of section 4 and test it in the near future.

6.3 An Examplative Run of the Algorithm

We have seen in the previous subsection that our acceleration for Multi Isolated Transfer Nets does not always put a ω in a source that is increasing. Note that this is the only difference with the algorithm for Petri Net. Applying the usual algorithm for Petri net to a MTN may result in an over-approximation. We illustrate this phenomenon on an example.

Fig. 9(a) shows a simple example of Multi Isolated Transfer Net. We consider $\langle 1,0,0,\omega\rangle$ as initial ω-marking. Fig. 9(b-d) shows the computations of algorithms presented in the previous sections. Fig. 9(b) presents the tree of the enumerative algorithm for Petri Net after two iterations. At this step, it detects that the successor for the transition b is greater than the initial ω-marking $\langle 1,0,0,\omega\rangle$. Thus, case (2) is applied (so, here we forget that s_3 is a source of a transfer in a) : ω is added to the initial ω-marking in s_3, the initial marking is then replaced and the procedure continue from this new unique node. Finally, the algorithm ends after computing the tree shown in Fig. 9(c).

According to the upward closed set $s_2 \geq 1 \wedge s_3 \geq 1$, the algorithm specialized for Multi Isolated Transfer net computes the same tree. Thus, the computed weak coverability set coincides with the over-approximation computed by the algorithm for Petri net and the two algorithms allow us to conclude that there is a mutual exclusion between place s_2 and s_3. But if the safety property to be verified is "no reachable markings satisfies $s_3 \geq 2$", only our algorithm computing the weak coverability set returns the right answer. At the second step, as shown by Fig. 9(b) the algorithm also detects that the successor for transition b is greater than the initial marking but doesn't put ω for the place s_3 because s_3 is the source of a transfer and the number of tokens is not enough to intersect with the upward closed set $s_3 \geq 2$. Fig. 9(d) shows the final tree computed by the new algorithm.

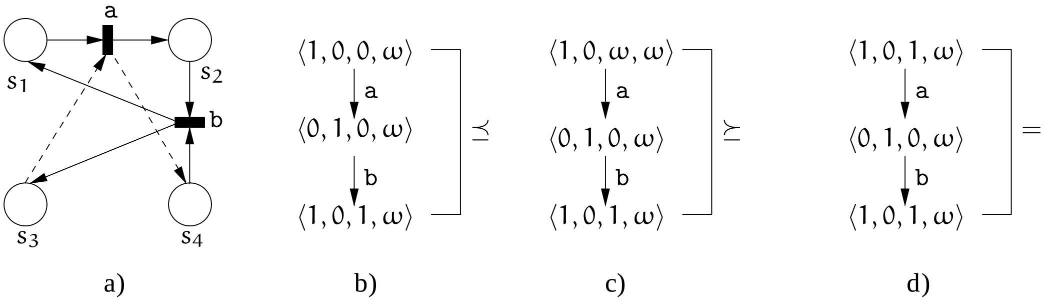


Figure 9: An example of Isolated Multi-transfer Net.

7 The General Case: Combination of Forward and Backward Search

As we already recalled the forward approach cannot be extended to the full class of MTNs [8]. But as shown in Fig. 7, backward approach seems to be more *explosion prone* and seems to be useful only when some information about potential reachable markings can be used to *guide* the search. Structural invariants have been used to prune the backward search space in [10, 12]. Unfortunately, results of Fig. 7 show that this technique does not always speed up the search.

Fig. 10 shows results on some MTNs corresponding to abstractions of Java programs analysed with the symbolic backward algorithm of [12]. Without information to prune the search space, the backward algorithm can take a lot of time, about one hour for the $\mathrm{P/C_{bug}}$ example and for the corrected version of the model. In those particular examples, the over-approximation computed by the structural invariants is very effective, compare the columns in the table of Fig. 10 giving the time for the computation of Pre^* without and with the invariant heuristic. But this is not always the case.

On the other hand, the forward symbolic algorithm of section 4 has always very short execution times on those examples. Unfortunately, that algorithm only computes an over-approximation of the coverability set for that class of system. For the $\mathrm{P/C^1_{correct}}$ and $\mathrm{P/C^2}$ examples, the algorithm of section 4 established that the bad markings were not reachable. In the other cases, that is for $\mathrm{P/C^1_{bug}}$ and the I/D systems, the forward over-approximation cannot be conclusive as bad markings are reachable. Note that bad markings are really reachable but we can not decide it with the over-approximation given by this algorithm. So in those cases, the backward algorithm has to be applied [6].

Case Study	**P**	**T**	$\mathbf{NN}_{\mathsf{Post}}$	$\mathbf{NE}_{\mathsf{Post}}$	$\mathbf{I}_{\mathsf{Post}}$	$\mathbf{EX}_{\mathsf{Post}}$	$\mathbf{M}_{\mathsf{Post}}$	$\mathbf{NN}_{\mathsf{Pre}}$	$\mathbf{NE}_{\mathsf{Pre}}$	$\mathbf{I}_{\mathsf{Pre}}$	$\mathbf{EX}^1_{\mathsf{Pre}}$	$\mathbf{N}_{\mathsf{Pre}}$	$\mathbf{EX}^2_{\mathsf{Pre}}$
I/D	32	28	218	40	73	3.16s	2728kb	800	3073	30	20.11s	6008kb	3.3s
$\mathrm{P/C^1_{bug}}$	44	37	688	93	78	37.4s	4772kb	8418	12234	29	1h12m	33884kb	9.21s
$\mathrm{P/C^1_{correct}}$	44	37	306	25	18	0.63s	2248kb	12479	8396	29	55m50s	31884kb	0.02s
$\mathrm{P/C^2}$	20	16	107	14	33	0.23s	1732kb	566	810	19	3.1s	3012kb	0.04s

Figure 10: verification of MTNson an AMD Athlon 900Mhz 500Mbytes

Let us now show that the forward approximation computed by the algorithm of section 4 can give more information than the structural invariants. The parametrized MTN of the MESI protocol (see [18]) has only one structural invariant which says that the number of tokens in all the places is constant in all the forward reachable markings. As the place $\mathsf{invalid}$ can contain any number of tokens (it contains ω tokens in the initial ω-markings), this invariant does not give any information to prune the backward search (see [10] for more details). But by applying the symbolic algorithm presented in section 4, we obtain that the reachable markings are covered by the ω-markings $\{\langle\omega,\omega,\omega,0\rangle,\langle\omega,0,\omega,\omega\rangle\}$ (where markings are encoded as $\langle\mathsf{invalid},\mathsf{shared},\mathsf{modified},\mathsf{exclusive}\rangle$). This over-approximation allows us to verify

[6] We could have applied the algorithm of section 6 to compute a WCS that is sufficient to verify the safety properties but our remarks in this section are more general and apply to the full class of MTNs and we do not have currently an efficient symbolic implementation of that algorithm

the mutual exclusion property between shared and exclusive. Other interesting properties cannot be verified with this over-approximation. As an example, consider the mutual exclusion property that asks that at most one token can be in exclusive. The backward approach can be applied efficiently to answer this question by eliminating during the search all the upward closed sets that do not intersect with the over-approximation computed forwardly. This over-approximation contains more information than the structural invariants and so is potentially more effective than the structural invariant heuristic. So, instead of opposing the forward and backward approaches, we propose to use them together.

First, if we have to verify that a Multi Transfer Net that does not fall in the class of Multi Isolated Transfer Net cannot reach an upward closed set of Bad markings U, we first compute an over-approximation of the minimal coverability set with the symbolic procedure defined in section 4. Let us note $\mathcal{O}$ this set of ω-markings. Then, we check if $\downarrow\mathcal{O} \cap U \neq \emptyset$. If this is the case, then we are done, we know that the MTN cannot reach U. Otherwise, we can use $\mathcal{O}$ in conjunction with the backward search as follows. Instead of computing $\mu X \cdot U \cup \mathsf{Pre}(X)$ we compute $\mu X \cdot ((U \sqcap \downarrow\mathcal{O}) \cup (\mathsf{Pre}(X) \sqcap \downarrow\mathcal{O}))$ where $\sqcap$ is defined as :

$$\sqcap : \text{ upward closed set } \times \text{ downward closed set } \to \text{upward closed set}$$

$$S \sqcap T = \{\mathbf{m} | \exists \mathbf{m}' \in S : \mathbf{m}' \preceq \mathbf{m} \text{ and } \mathbf{m}' \in T\}$$

So, we remove from S any marking whose upward closure does not intersect with T. This is mainly the same idea that we use in the invariant heuristic defined in [10]. But the information computed using the forward search is always at least as precise as the one computed using the structural invariants and often much more precise. We will experiment with this heuristic in the near future.

References

[1] P. Abdulla, L. Boasson, A. Bouajjani. Effective Lossy Queue Languages. In *Proc. 28th Intern. Coll. on Automata, Languages and Programming (ICALP'01)*, LNCS, Crete (Greece), July 2001.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Series in Parallel Computing. John Wiley & Sons, 1995.

[3] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS'96*, pages 313–321, 1996.

[4] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Prameterized Networks. In the proceedings of the 6th International Conference, TACAS 2000, LNCS 1785, Springer-Verlag, 2000.

[5] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmeti In *Proc. CONCUR'99*, LNCS 1664, pages 178-193, 1999.

[6] A. Bouajjani and R. Mayr. Model Checking Lossy Vector Addition Systems. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of LNCS, pages 323–333. Springer, 1999.

[7] G. Ciardo, A.S. Miner. Storage Alternatives for Large Structured State Space. In *Proc. Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 1245, pages 44-57, 1997.

[8] C. Dufourd, A. Finkel, Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In *Proc. ICALP'98*, LNCS 1443, pages 103-115,Springer, 1998.

[9] G. Delzanno, and J. F. Raskin. Symbolic Representation of Upward-closed Sets. In *Proc. TACAS 2000*, LNCS 1785, pages 426-440, 2000.

[10] G. Delzanno, J.-F. Raskin, and L. Van Begin. Attacking Symbolic State Explosion. In *Proc. CAV'01*, LNCS 2102, pages 298-310, 2001.

[11] G. Delzanno, J-F. Raskin and L. Van Begin. Covering Sharing Trees: Efficient Data Structures for the Automated Verification of Parameterized Systems. *http://www.ulb.ac.be/di/ssd/jfr/CST.ps*, 2002.

[12] G. Delzanno, J-F. Raskin and L. Van Begin. Towards the Automated Verification of Multithreaded Java Programs. *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, Grenoble, Frane, 2002.

[13] J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. LICS'99*, pages 352–359, 1999.

[14] A. Finkel. Reduction and covering of infinite reachability trees. Information and Computation, 89(2):144-179, December 1990.

[15] A. Finkel. The minimal coverability graph for Petri nets. In In *Advances in Petri Nets '93*, LNCS 674, pages 210-243. Springer, 1993.

[16] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63-92, 2001.

[17] S. M. German, A. P. Sistla. Reasoning about Systems with Many Processes. *JACM* 39(3): 675–735 (1992)

[18] J. Handy. The Cache Memory Book. Academic Press, 1993.

[19] D. S. Johnson. A Catalog of Complexity Classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A, Algorithm and Complexity*, Elsevier, 1990.

[20] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences,* 3, pages 147-195, 1969.

[21] P. Marenzoni, S. Caselli, G. Conte. Analysis of Large GSPN Models : A Distributed Solution Tool. In *Proc. Int. Work. on Petri Nets and Performance*, 1997.

[22] C. Rackoff. The Covering and Boundness Problem for Vector Addition Systems. *Theoretical Computer Science* 6, 223, 1978.

[23] Ph. Schnoebelen. Verifying Lossy Channel Systems Has Nonprimitive Recursive Complexity. LSV Technical Report, Ecole Normale Superieure de Cachan, France, 2002.

[24] M. Silva, E. Teruel, and J. M. Colom. Linear Algebraic and Linear Programming Techniques for Analysis of Place/Transition Net Systems. In W. Reisig and G. Rozenberg, editors. Lectures on Petri Nets I: Basic Models. *Advances in Petri Nets,* LNCS 1491, pages 308-309. Springer, 1998.

[25] D. Zampuniéris, and B. Le Charlier. Efficient Handling of Large Sets of Tuples with Sharing Trees. In *Proc. DCC'95*, 1995.

Model Checking Birth and Death

DINO DISTEFANO, AREND RENSINK, JOOST-PIETER KATOEN

Department of Computer Science, University of Twente

P.O. Box 217, 7500 AE Enschede, The Netherlands

E-mail: {ddino, rensink, katoen}@cs.utwente.nl

Abstract

One of the aspects of computation that state-of-the-art model checking does not deal with very well is that of dynamic *allocation* and *deallocation* (birth and death) of entities. This is especially true if the number of entities is not known beforehand, or even unbounded (causing the state space to become *infinite*). Nevertheless, allocation and deallocation are fundamental concepts in many fields of computer science. For example, in object-orientation, systems are composed by dynamic objects (*entities*) that can be created (*allocated*) in an arbitrary number during the computation. Moreover objects can be destroyed (*deallocated*) e.g., by a garbage collector. Yet another significant field of application is certainly security where properties of systems related with *fresh* secure resources as keys are mostly investigated.

Though there are now calculi (such as the π-calculus [7]) that can express the generation of fresh names, as well as models (such as history-dependent automata [8]) that can describe both the birth and the death of entities, what has been missing so far is a logic where these concepts are captured as primitives; a logic that should be as fundamental to reasoning about dynamic allocation as standard propositional logic is to reasoning about a fixed state space. A formalism that, in a natural way, would allow the specification of properties like *a fresh entity will always eventually be allocated* or *before a particular entity is deallocated, two new entities will be allocated.* Returning to the example of object-oriented systems, we would like to express properties like *every object in the current state will be eventually deallocated* or *the number of running objects will never be less than two*, etc. For security: *an authentication server never provides already used secret session keys* (in general, authentication servers are trusted to generate new keys in a proper manner, however, it is easy to imagine a naive or, even faulty, implementation of such servers).

An attempt to formulate such a logic will be presented in this talk (based on [4]). Called *allocational temporal logic* ($\mathcal{A}\ell\ell$TL), it has the following features: (i) Entity variables x, y, interpreted by a mapping to the entities existing (i.e., *alive*) in a given state. The interpretation is *partial*: a variable not mapped onto an existing entity stands for an entity that has *died*. (ii) Entity equations $x = y$ (where x, y are entity variables), asserting that x and y refer to the same entity. This cannot hold if either x or y has died; (iii) Entity quantification $\exists x.\phi$, which holds in a given state if ϕ holds for some interpretation of x, provided that x is alive. (iv) A predicate x new to express that the entity referred to by x is *fresh*, i.e., newly born. In addition, $\mathcal{A}\ell\ell$TL has the standard LTL temporal operators.

The logic is interpreted over *high-level allocational Büchi automata* (HABA), which extend *history-dependent automata* [8] with a predicate for the *unboundedness* of (the number of entities in) a state, and with a (generalized) Büchi acceptance condition. As for history-dependent automata, a crucial point is that entity identity is *local* to a state.

HABA can be used as finite-state abstraction of certain kind of infinite-state systems. As an example, we define a small imperative language whose main features are the allocation and deallocation of entities. Although the number of entities allocated by a program in this language can be unbounded, the operational semantics yields a finite HABA (a significant condition for the application of model checking).

Together with the logic $\mathcal{A}\ell\ell$TL, the main contribution of this work is that the model-checking problem for $\mathcal{A}\ell\ell$TL is shown to be decidable on HABA. In particular, we present a tableau-based model-checking algorithm that decides whether a given $\mathcal{A}\ell\ell$TL-formula holds for a given HABA. Our algorithm extends the tableau-based algorithm for LTL [6]. To the best of our knowledge, this yields the first approach to effectively model-check models with an unbounded number of entities. This is of particular interest to e.g. the verification of object-oriented systems in which the number of objects is typically not known in advance and may be even unbounded. Currently, in tools for model checking object-oriented systems (such as [3, 5]) dynamic creation of objects is only supported to a limited extent (the number of created objects must be bounded). Furthermore, reasoning about (de)allocation of fresh entities is relevant also in relation to *privacy* and *locality* as discussed in, e.g., [1, 2, 7].

References

[1] L. Caires and L. Cardelli. A spatial logic for concurrency (part I). In *TACS'01*, LNCS 2255, pp. 1–37, Springer, 2001.

[2] L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *TLCA'01*, LNCS 2044, pp. 46–60, Springer, 2001.

[3] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *ICSE'00*, pp. 439–448, IEEE CS Press, 2000.

[4] D. Distefano, A. Rensink and J.-P. Katoen. Model checking birth and death. To appear in 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002).

[5] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *Int. J. on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

[6] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL'85*, pp. 97–107, ACM Press, 1985.

[7] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (part I and II). *Inf. & Comp.*, 100(1): 1-77, 1992.

[8] U. Montanari and M. Pistore. An introduction to history-dependent automata. *Electr. Notes in Th. Comp. Sci.*, 10, 1998.

On the power of nested X and U modalities in the logic LTL[*]

Antonín Kučera Jan Strejček

Faculty of Informatics
Masaryk University
Botanická 68a, 60200 Brno
Czech Republic
{tony,strejcek}@fi.muni.cz

Linear temporal logic (LTL) [Pnu77] is a popular formalism for specifying properties of (concurrent) programs. It extends propositional logic by two modal connectives—the unary X ('next') operator and the binary U ('until') operator. A natural question is how the nesting-depth of these modalities influences the expressive power of LTL. To answer this (and related) questions, we formulate and prove a general pumping lemma admitted by LTL languages. Thus, we obtain a powerful tool which allows to demonstrate the semantical strictness of three hierarchies of LTL formulae, which are parametrized either by the nesting depth of just one of the two modalities, or by both of them. This part of our presentation is closely related to the work of Etessami and Wilke [EW00] (see also [Wil99] for an overview of related results).

The designed pumping lemma provides a necessary condition for the membership to a given class in a given hierarchy, which is, at least in one case, also sufficient. We discuss this issue in greater detail and presenting some recent results.

Finally, we shall also discuss a potential applicability of our results to the problem of state-space explosion in the context of model-checking with LTL [CGP99].

[*] Some of the presented results will appear in Proc. of CSL'02.

References

[CGP99] E.M. Clark, O. Grumberg, and D.A. Peled. *Model Checking.* The MIT Press, 1999.

[EW00] K. Etessami and T. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160:88–108, 2000.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Wil99] T. Wilke. Classifying discrete temporal properties. In *Proceedings of STACS'99*, volume 1563 of *LNCS*, pages 32–46. Springer, 1999.

Applications of the Existential Quantification Technique

Jiří Srba[*]

BRICS[†]
Department of Computer Science,
University of Aarhus,
Ny Munkegade bld. 540,
8000 Aarhus C, Denmark
`srba@brics.dk`

August 14, 2002

Infinity'02 Presentation

(Abstract)

Bisimilarity checking of infinite-state systems has been an active and fascinating area of research for the last decade [1] and a number of original techniques were invented in order to explore the decidability barriers of strong and weak bisimilarity. Typical representatives of systems with unboundedly many reachable states are for example Pushdown Automata (PDA), Petri Nets (PN), Basic Process Algebra (BPA) and Basic Parallel Processes (BPP).

Recently, new techniques were developed to better classify the decidability and complexity aspects of bisimilarity checking for infinite-state systems. One of them is called the *existential quantification technique*. This technique was first used by Jančar [2] in the context of high undecidability of weak bisimilarity for PN and explicitly formulated by Srba in [3, 4]. The existential quantification technique gives the defender (in the usual bisimulation game) the possibility to make independent choices in case of nondeterministic branching.

The technique was beneficial for showing that strong bisimilarity of BPP [3] and later also of BPA [4] are PSPACE-hard problems. These proofs were achieved by polynomial

[*] The author is supported in part by the GACR, grant No. 201/00/0400.

[†] **B**asic **R**esearch in **C**omputer **S**cience,
Danish National Research Foundation.

time reductions from the problem of quantified boolean formula (QBF) and even though the classes BPA and BPP differ substantially in their capabilities, the proof strategies are similar. The reductions can be divided into two phases.

First, a quantified assignment of boolean variables is generated and the clauses of QBF satisfied by this assignment are remembered in the current states. Second, the condition that all clauses are satisfied is checked. It is the first (and the main) phase of assignment generation where the existential quantification technique is used. Moreover, this assignment generation is general enough to be uniformly described both for BPA and BPP. The only place where the proofs for BPA and BPP necessarily differ is in the second phase where the satisfied clauses are checked.

In this presentation we aim at providing a general meta-theorem which formally describes the sufficient conditions for process algebras to be capable of the assignment generation. We also present four applications in order to demonstrate the usefulness of the meta-theorem. The first two applications briefly repeat the PSPACE-hardness of strong bisimilarity for BPA and BPP. The third application deals with strong bisimilarity of normed PDA[1]. The last application proves a new result, namely PSPACE-hardness of weak bisimilarity for normed BPA.

References

[1] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[2] P. Jančar. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*, volume 915 of *LNCS*, pages 349–363. Springer-Verlag, 1995.

[3] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proceedings of 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *LNCS*, pages 535–546. Springer-Verlag, 2002.

[4] J. Srba. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard. In *Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 716–727. Springer-Verlag, 2002.

[1] This problem was very recently improved to EXPTIME-hardness (Kučera and Mayr, MFCS'02) — also by using the existential quantification technique.

New Results for Bisimilarity on Basic Parallel Processes

Petr Jančar

Dept. of Computer Science
FEI, Technical University of Ostrava
17. listopadu 15, CZ-708 33 Ostrava
Czech Republic
Petr.Jancar@vsb.cz

An effective construction of a finite set of linear equations characterizing bisimulation equivalence on a given BPP (Basic Parallel Processes) system will be presented. The description has exponential size but (a variant of it) can be 'traversed' in polynomial space.

This clarifies various problems related to bisimilarity on BPP. E.g., it can be shown that the bisimilarity problem for BPP is in PSPACE; combined with PSPACE-hardness which has been shown recently by J. Srba, PSPACE-completeness of the problem is thus established.

Possibilities to use the introduced method in the case of weak bisimilarity will be briefly discussed.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

```
http://www.fi.muni.cz/informatics/reports/
ftp  ftp.fi.muni.cz (cd pub/reports)
```

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic