On Verification Problems for Systems with Infinite State Spaces

Habilitationsschrift

vorgelegt zur
Erlangung der Lehrbefugnis
für Informatik
an der
Fakultät für Angewandte Wissenschaften
der
Albert-Ludwigs-Universität zu Freiburg i.Br.

von

Richard Mayr

Freiburg, im September 2002

Abstract

This is a collection of my works on verification problems for systems with infinite state spaces. It mostly contains results in the following three areas:

- The decidability and computational complexity of model checking problems for systems with infinite state spaces.
- The relationship between model checking and semantic equivalence checking.
- The decidability and computational complexity of checking semantic equivalences between systems with infinite state spaces.

This collection begins with a short summary titled "On Verification Problems for Systems with Infinite State Spaces", which gives a general overview over the subject. Then follow the papers that contain the details.

Notice: In order to make this thesis self-contained, some material has been included that has already appeared in my dissertation. In detail, the situation is as follows.

The papers number 2 and 6 (see the table of contents) contain material from my dissertation (in slightly extended and modified form). The other papers numbers 3,4,5,7,8,9,10,11 and 12 are new and have nothing in common with my dissertation.

Now we make the same distinction for all theorems that appear in the summary chapter of this thesis. The theorems 1,2 and 10 appeared already (or follow from results) in my dissertation. The other theorems 3–9 and 11–23 are new.

The few papers and theorems that overlap with my dissertation are clearly marked in the table of contents and in chapter 1 (the summary chapter).

Table of Contents

On Verification Problems for Systems with Infinite State Spaces

Richard Mayr

Department of Computer Science,
Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee Geb. 51,
D-79110 Freiburg, Germany.
Phone: +49 761 203 8196, Fax: +49 761 203 8182
mayrri@informatik.uni-freiburg.de
http://www.informatik.uni-freiburg.de/~mayrri

Abstract. This is a short summary of most of my work on verification problems for systems with infinite state spaces. It mostly describes results in the following three areas:

- The decidability and computational complexity of model checking problems for systems with infinite state spaces.
- The relationship between model checking and semantic equivalence checking.
- The decidability and computational complexity of checking semantic equivalences between systems with infinite state spaces.

In order to make it easier to distinguish my own work from that of other authors, all citations of my own work will be in bold face. The three theorems (number 1,2 and 10) that are already contained in my dissertation are explicitly marked.

1 Abstract Models for Infinite-State Systems

Since general programming languages are Turing-powerful, verification problems for them are undecidable. Therefore, abstract system models have been introduced that can model most of the behavior of full programs, but still retain the decidability of most of their verification problems. While some abstractions are finite-state, others still generate systems with infinitely many reachable states. Examples are systems with unbounded creation of new parallel processes, systems which call subroutines (unbounded recursion), systems with unbounded data structures (like counters or buffers), or combinations of all these.

1.1 Bisimulation

The abstract systems models generate (possibly infinite) *labeled transition graphs*, i.e., directed graphs whose nodes represent states and whose arcs represent transitions which are labeled with atomic actions. The underlying semantics of these

systems is given by the following semantic equivalences. We consider the semantic equivalences *weak bisimilarity* and *strong bisimilarity* [Mil89] over transition systems.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over states (of processes) is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them.*

Bisimulation equivalence can also be described by *bisimulation games* [Sti98, Tho93] between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

1.2 Process Rewrite Systems

Many classes of systems that have been studied by other authors and myself can be described in the formalism of *process rewrite systems*, that has been defined in [**May00c**].
Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E ::= \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '.' is a sequential composition and '||' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '||' are associative, '||' is commutative, and 'ϵ' is a unit for '.' and '||'.
A *process rewrite system* (PRS) [**May00c**] is specified by a finite set Δ of *rules* which have the form $E \stackrel{a}{\rightarrow} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '||' is commutative):

$$\frac{(E \xrightarrow{a} F) \in \Delta}{E \xrightarrow{a} F} \qquad \frac{E \xrightarrow{a} E'}{E.F \xrightarrow{a} E'.F} \qquad \frac{E \xrightarrow{a} E'}{E \| F \xrightarrow{a} E' \| F}$$

We extend the notation $E \xrightarrow{a} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \xrightarrow{w} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of the rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. We also use '1' to denote the set of process constants.

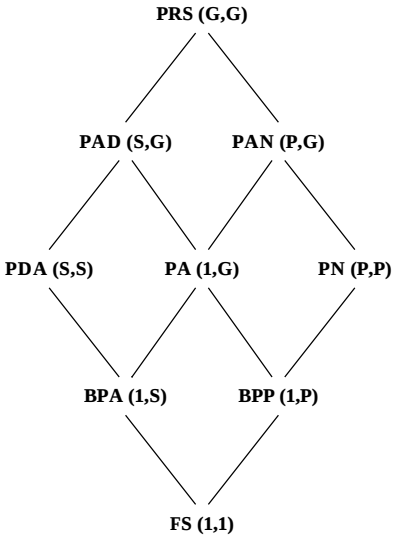


Fig. 1. The PRS-Hierarchy

The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively.

Many of these (A, B)-PRS correspond to widely known models like Petri nets, pushdown processes, context-free processes and others.

1. A $(1, 1)$-PRS is a finite-state system. Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.
2. $(1, S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [BE97, Esp97]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only leftmost derivations are permitted.

3. It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left side of rules). Caucal [Cau92] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labeling of states. Thus (S, S)-PRS are equivalent to pushdown processes, the processes described by pushdown automata.
4. (P, P)-PRS are equivalent to Petri nets. Every constant corresponds to a place in the net and the number of occurrences of a constant in a term corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.
5. $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [BE97, Esp97]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [Chr93].
6. $(1, G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication (see, e.g., [BK85] or [**May01, May97b, May97c**]).
7. (P, G)-PRS are called *PAN-processes* in [**May97a**]. It is a common generalization of Petri nets and PA-processes and it is strictly more general than both of them (e.g., PAN can describe all Chomsky-2 languages while Petri nets cannot).
8. (S, G)-PRS are a common generalization of pushdown processes and PA-processes. They are called *PAD* (PA + PD) in [**May01**].
9. The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [**May00c**]. They subsume all the previously mentioned classes.

It was also shown in [**May00c**] that PRS are equivalent to ground AC rewrite systems, modulo some coding. (Ground rewrite systems are rewrite systems with only ground terms, i.e., terms without free variables. This means rewriting without substitution. Ground AC rewrite systems are ground rewrite systems on terms that can contain an associative and commutative operator.) However, normally ground AC rewrite systems are not used to describe labeled transition systems, since their rewrite rules are normally not labeled with atomic actions. The two main results about PRS in [**May00c**] are the following.

Theorem 1. [**May00c**] *(already appeared in my dissertation* [**May98**]*)*
The PRS-hierarchy is strict with respect to bisimulation equivalence.

This means that all subclasses of PRS in the hierarchy are different from each other w.r.t. bisimulation. If there is an arc from a lower class to a higher one, then the higher one is strictly more expressive (there is a process in the higher class that is not bisimilar to any process in the lower class). Note that this theorem does not carry over to language equivalence, since, e.g., BPA and pushdown automata both describe the class of context-free languages.

Theorem 2. [**May00c**] *(already appeared in my dissertation* [**May98**]*)*
The reachability problem for PRS is decidable.

The algorithm that decides this problem uses a reduction to a nested reachability problem for both Petri nets [May84] and pushdown automata. It uses the facts that the reachability problem was already known to be decidable for these classes of systems.

2 Model Checking VASS and Lossy VASS

One of the most important and most widely known models in the PRS-hierarchy are (P,P)-PRS, which are equivalent to vector addition systems with states (VASS) or Petri nets. In [**BM99**] the decidability of several model checking problems for VASS and some related models has been studied.

An important concept in this context is the notion of *lossiness*. It was first defined to model communication through unreliable channels. The main example are lossy fifo-channel systems, which are systems of finite-state processes that communicate through lossy fifo-channels (buffers) of unbounded length. These lossy fifo-channels are unreliable, because they can spontaneously lose messages. Since normal (non-lossy) fifo-channel systems are Turing-powerful, automatic analysis of them is restricted to special cases [BH97]. However, lossy fifo-channel systems are not Turing-powerful, since reachability and some safety-properties are decidable for them [AJ93, CFI96, ABJ98].

If one abstracts from the order of the messages in the communication channel, but considers only their numbers and types, then one obtains lossy VASS, or lossy counter machines (if one allows tests for zero). Here lossiness means that the value in a Petri net place or counter can spontaneously decrease at any time.

Normal VASS, lossy VASS and lossy counter machines are very different from each other w.r.t. the decidability of model checking problems [**BM99**]. One example is the decidability of model checking with the temporal logic EF (a fragment of computation-tree logic (CTL); see Subsection 3.1).

Theorem 3. [**BM99**]
Model checking with the temporal logic EF is decidable for lossy VASS and lossy counter machines, but undecidable for normal VASS.

However, for different temporal logics the situation can be reversed, as the following theorem shows.

Theorem 4. [**BM99**]
Model checking with the temporal logic LTL is decidable for normal VASS and lossy VASS, but undecidable for lossy counter machines.

The complete results in [**BM99**] are very detailed (and quite technical). They concern much more expressive logics than just EF and LTL. The two theorems

above are just special cases of these results that illustrate the difference between VASS and lossy counter machines.
Some even more general results about the decidability of various problems for lossy counter machines have been achieved in [**May00d**].

Theorem 5. [**May00d**]
Termination is decidable for classic lossy counter machines.

Theorem 6. [**May00d**]
Universal termination (i.e., termination for every input) is undecidable for all types of lossy counter machines.

Theorem 7. [**May00d**]
Boundedness is undecidable for all types of lossy counter machines.

These undecidability results for lossy counter machines have many applications, since the underlying principle is very general. For example, the undecidability of the boundedness problem for reset Petri nets (Petri nets with special reset-arcs that can reset a place to zero, but not test for zero) follows as a corollary from these results. This is because reset Petri nets correspond to a special type of lossy counter machine [**May00d**].
There are other applications in the area of parameterized verification problems. In a parameterized verification problem one tries to show the correctness of a whole (possibly infinite) class of systems, not just a single instance. An example is a communication network $P(n)$ with an arbitrary number n of components (see below). The following theorem (which follows from the results on lossy counter machines) shows that most liveness problems are undecidable for parameterized systems.

Theorem 8. [**May00d**]
A parameterized verification problem is undecidable if it satisfies the following conditions:

1. *It can encode an n-space-bounded lossy counter machine (for some lossiness relation) in such a way that $P(n)$ corresponds to the initial configuration with n in one counter and 0 in the others.*
2. *It can check for the existence of an infinite run.*

The technique of Theorem 8 is used in [**EFM99**] to show the undecidability of the fairness problem for broadcast communication protocols. These are systems of n indistinguishable communicating finite-state processes. The rules for communication are as follows:

1. Two processes can communicate directly by handshake.
2. One process can broadcast a message, which is received (immediately) by all other $n-1$ processes.

Every message sent or received by a process can change its internal state, which in turn defines what actions it can perform and how it reacts to messages. The rules for communication are defined independently from the number n of processes in the system. If one considers processes with k internal states then any configuration of the broadcast protocol with n processes can be described by a tuple $(m_1, m_2, \ldots, m_k)$ where m_i is the number of processes in state i and $\sum_{j=1}^{k} m_j = n$. Every such m_i can be seen as the content of a counter which is bounded by n. A broadcast can cause all processes in a certain state to change to another state. This can be used to reset such a simulated space-bounded counter to zero. Note however, that no test for zero is possible. The problem if such a broadcast protocol terminates (i.e., for every number n of processes the system terminates) is undecidable, because it satisfies the conditions of Theorem 8 (the lossiness relation used here is reset-lossiness). Thus all fairness properties, like those expressible in the temporal logics CTL [CE81, Eme94]) and LTL (linear-time temporal logic [Pnu77]), are undecidable as well.

Theorem 9. [**EFM99**]
All liveness problems for parameterized broadcast communication protocols are undecidable.

In the same way, similar results can be proved for parameterized problems about systems with bounded buffers, stacks, etc.
It should be noted however, that not all verification problems for parameterized systems are undecidable. Many safety problems are still decidable for parameterized systems. For example, it has been shown in [**EFM99**] that some safety problems for parameterized broadcast communication protocols can be solved by a decidable backwards reachability analysis (while a forwards reachability analysis cannot possibly be effective).

3 Model Checking with the Temporal Logic EF and Characteristic Formulae

3.1 Model Checking with EF

The temporal logic EF is a simple, but natural fragment of computation-tree logic [CE81, Eme94]. Instead of the general "until" operator of CTL it has the operator EF, where $EF\,\Phi$ means "there is a reachable state where Φ holds".
It has been shown in [**May01**] that model checking with the temporal logic EF is decidable for many more classes of infinite-state systems than for most other branching-time temporal logics. The most important result of [**May01**] is the following:

Theorem 10. [**May01**] *(already appeared in my dissertation* [**May98**]*)*
The model checking problem for the temporal logic EF and the process class of (S, G)-PRS (also called PAD) is decidable.

Since model checking Petri nets with EF was already known to be undecidable [Esp97], this result establishes the border of decidability of EF model checking in the PRS-hierarchy. Note that model checking with other branching-time temporal logics like CTL or modal μ-calculus is only decidable for purely sequential systems (like pushdown automata), but undecidable for systems that contain parallelism (like BPP, Petri nets, PA or PAD) [Esp97].

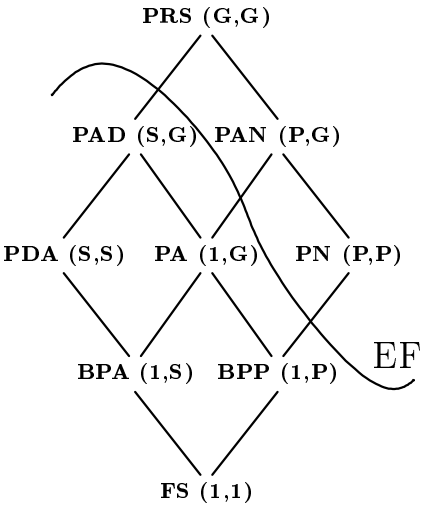


Fig. 2. The border of the decidability of model-checking with the temporal logic EF.

The proof of Theorem 10 (see [**May01**]) is done by a tableau construction. If the process satisfies the formula then a finite proof of this is constructed, otherwise the tableau stops and fails after a finite number of steps. The complexity of the tableau-construction is very high (k-times exponential for formulae of nesting depth k), although this model checking problem is only known to be *PSPACE*-hard. For the subclass of (1,P)-PRS (BPP), model checking with EF is known to be *PSPACE*-complete [**May96, May98**]. Pushdown automata ((S,S)-PRS) are also a subclass of PAD ((S,G)-PRS). Model checking pushdown automata with EF is also *PSPACE*-complete [Wal00].

For PA-processes ((1,G)-PRS), another subclass of PAD, a quite different model checking algorithm has later been given by Lugiez and Schnoebelen [LS98]. Their algorithm uses tree-automata to represent infinite sets of reachable configurations of PA-processes. However, the complexity of their algorithm is the same as that of the tableau construction, and, according to Lugiez and Schnoebelen, their algorithm cannot be generalized to the full class of PAD-processes.

3.2 Characteristic Formulae

It has been shown in [**JKM01**] that the results about the decidability of model checking problems with the logic EF can be used to show the decidability of certain bisimulation problems. In [**JKM01**] the decidability of bisimulation-like equivalences between infinite-state processes and finite-state ones was studied. The motivation is that the intended behavior of a process is often easy to specify (by a finite-state system), but a 'real' implementation can contain components which are essentially infinite-state (e.g., counters, buffers). The aim of formal verification is to check if the finite-state specification and the infinite-state implementation are semantically equivalent (i.e., bisimilar).

It has turned out that there is a connection between checking of (strong or weak) bisimulation equivalence and model checking, via the so-called *characteristic formulae*. A characteristic formula for a finite-state system F (w.r.t. strong or weak bisimulation) describes this system completely (modulo strong or weak bisimulation). In the following we only consider weak bisimulation, since strong bisimulation is a special case of it. A characteristic formula Φ_F of a finite-state system F has the property that for any general transition system G (where G can be infinite) we have

$$G \approx F \iff G \models \Phi_F$$

Therefore, one can reduce the equivalence problem $G \approx F$ to a model checking problem. The question was, if (and how) one can construct a characteristic formula for a given finite-state system. It had already been known that one can construct characteristic formulae in the modal μ-calculus [SI94]. The problem with this was that μ-calculus model checking is undecidable for so many classes of infinite-state systems.

In [**JKM01**] it was shown that one can effectively construct characteristic formulae even in the simple temporal logic EF.

Theorem 11. [**JKM01**]
For every finite-state system F one can effectively construct a characteristic formula Φ_F (w.r.t. weak bisimulation), where Φ_F is a formula in the temporal logic EF.

By combining this theorem with Theorem 10, one immediately gets the following result.

Theorem 12. [**JKM01**]
Weak bisimilarity of a PAD-process (an (S,G)-PRS) and a finite-state system is decidable.

Similar theorems follow for other classes of systems with a decidable EF-model checking problem, e.g., lossy counter machines (see Theorem 3).

4 Semantic Equivalence Checking

In the area of formal verification, some of most important semantic equivalences are strong and weak bisimulation equivalence (see Section 1), and simulation equivalence (with is defined indirectly via the notion of simulation preorder).

The three most common problems about semantic equivalence checking with infinite-state systems are the following:

EQUIVALENCE CHECKING OF INFINITE-STATE AND FINITE-STATE SYSTEMS.

Instance: A description of an infinite-state system α and a finite-state system F.

Question: Are α and F strongly bisimilar/weakly bisimilar/simulation equivalent ?

SEMANTIC FINITENESS

Instance: A description of an infinite-state system α.

Question: Does there exist a finite-state system F s.t. α and F are strongly bisimilar/weakly bisimilar/simulation equivalent ?

EQUIVALENCE CHECKING OF TWO INFINITE SYSTEMS

Instance: A description of two infinite-state systems α and β.

Question: Are α and β strongly bisimilar/weakly bisimilar/simulation equivalent ?

4.1 Comparing Infinite-State and Finite-State Systems

As shown in Subsection 3.2, some cases of the first problem (equivalence checking of infinite-state and finite-state systems w.r.t. weak bisimulation) can be reduced to a model checking problem with the logic EF for the infinite-state system. However, the complexity of the algorithms one obtains in this way is often unnecessarily high. Specialized algorithms can give much better complexity bounds. For example, while model checking context-free processes (BPA) with EF is *PSPACE*-complete, weak bisimilarity of BPA and finite-state systems is polynomial [**KM02c**].

Theorem 13. [**KM02c**]
Weak bisimilarity of BPA and finite-state systems is decidable in polynomial time.

A similar theorem for a subclass of BPP has also been shown in [**KM02c**]. Normed BPP are the subclass of BPP, for which from every reachable state there is a terminating computation.

Theorem 14. [**KM02c**]
Weak bisimilarity of normed BPP and finite-state systems is decidable in polynomial time.

For general BPP the exact complexity of this problem is still open. However, it is known to be decidable in polynomial space [**KM02c**].
It has been shown in [**May00b**] that the polynomial algorithms for BPA do not carry over to general pushdown automata.

Theorem 15. [**May00b**]
Weak bisimilarity of pushdown automata with a certain small fixed finite-state system (with only 3 states), is PSPACE-hard.

The same problem for strong bisimilarity is also *PSPACE*-hard, but fixed-parameter tractable (for fixed finite-state systems).

Theorem 16. [**May00b**]
Strong bisimilarity of pushdown automata with finite-state systems is PSPACE-hard. However, the required time is only polynomial in the size of the pushdown automaton and exponential in the size of the finite-state system.

Very recently, a *PSPACE* upper bound for these two problems has been shown in [**KM02a**]. So they are indeed *PSPACE*-complete.

4.2 Semantic Finiteness

Although there are many results about the decidability and complexity of the semantic finiteness problem (for various process classes and semantic equivalences), the picture is only partially clear, i.e., many open questions remain. Here we will only mention some of the most important results. For a more general overview see [**May00a, May00b, KM02b, KM99**]. Jančar and Esparza showed that semantic finiteness of Petri nets is decidable (and *EXPSPACE*-hard) for strong bisimulation, but undecidable for weak bisimulation [JE96]. Burkart, Caucal and Steffen showed that semantic finiteness of BPA is decidable for strong bisimulation [BCS96].
Decidability of semantic finiteness of pushdown automata is an open question, but the following lower bound has been shown in [**May00b**].

Theorem 17. [**May00b**]
Semantic finiteness of pushdown automata w.r.t. strong (or weak) bisimulation is PSPACE-hard.

The known lower bounds for Basic Parallel Processes (BPP) are not as high as those for general Petri nets.

Theorem 18. [**May00a**]
Semantic finiteness of Basic Parallel Processes (BPP) is co-$\mathcal{NP}$-hard w.r.t. strong bisimulation and Π_2^p-hard w.r.t. weak bisimulation.

Very recently, both these lower bounds for Basic Parallel Processes have been improved to *PSPACE* by Jiri Srba in [Srb02]. However, the proofs in [Srb02] use many of the techniques from [**May00a, May00b**].
There are only two results about semantic finiteness w.r.t. simulation equivalence. It is undecidable for Petri nets [JM95] and for PA-processes [**KM99, KM02b**].

Theorem 19. [**KM99, KM02b**]
Semantic finiteness of PA-processes w.r.t. simulation equivalence is undecidable.

4.3 Comparing Infinite-State Systems

While simulation preorder/equivalence is undecidable between all classes of infinite-state systems in the PRS-hierarchy [**KM99, KM02b**], some bisimulation problems are decidable. For example, it is known that strong bisimulation equivalence is decidable for Basic Parallel Processes (BPP) [CHM93], context-free processes (BPA) [BCS95], normed PA-processes [HJ99] and even pushdown automata [Sé98, Sti01]. (However, strong bisimilarity of Petri nets is undecidable [Jan94].)
Almost no upper complexity bounds are known for the decidable strong bisimulation problems, but a few lower bounds are known.

Theorem 20. [**May00a**]
Strong bisimilarity of Basic Parallel Processes (BPP) is co-$\mathcal{NP}$-hard.

This complexity bound has very recently been improved to *PSPACE* by Srba in [Srb02].
Another very recent result in [**KM02a**] improves the already established *PSPACE* lower bound for strong bisimilarity of pushdown automata.

Theorem 21. [**KM02a**]
Strong bisimilarity of pushdown automata is EXPTIME-hard.

Unlike for strong bisimilarity, very little is known about the decidability of weak bisimilarity between infinite-state systems. The known lower bounds for strong bisimilarity carry over, of course, but the decidability of most weak bisimulation problems is open.

5 Parameter-passing Systems

While many known classes of infinite-state systems are in the PRS-hierarchy, there are also some important ones outside it, like, e.g., (lossy) FIFO-channel systems (see [BH97, ABJ98]).
Another process class outside the PRS-hierarchy are the so-called BPA($\mathbb{Z}$)-processes. They have been defined in [**BHM01**] as an extension of BPA. The

intuition for this was as follows. Context-free processes (BPA) have been used for dataflow analysis in recursive procedures with applications in optimizing compilers [EK99]. However, BPA can only model the mutual calling structure of recursive procedures, not how data is passed between them. Therefore, a more refined model called BPA($\mathbb{Z}$) has been introduced, that can model not only recursive dependencies, but also the passing of an integer parameter to a subroutine. Moreover, this parameter can be tested against conditions expressible in Presburger arithmetic. This new and more expressive model can still be analyzed automatically. One can use one-counter machines to describe sets of reachable configurations of BPA($\mathbb{Z}$). The aim was to use this symbolic representation of sets of states to compute the sets of successors and predecessors of given states. This can be used to verify safety-properties.

Theorem 22. [**BHM01**]
The Post* (the set of successors) of a set of BPA($\mathbb{Z}$)-configurations described by a 1-CM can be effectively constructed.

Theorem 23. [**BHM01**]
The Pre* (set of predecessors) of a regular set of BPA($\mathbb{Z}$)-configurations can be effectively constructed. However, the Pre* of a set of BPA($\mathbb{Z}$)-configurations described by a 1-CM cannot be represented by a 1-CM in general and has an undecidable membership problem.

These results have been used in [**BHM01**] to model check BPA($\mathbb{Z}$)-processes with a logic called ISL.

6 Conclusion

In conclusion it can be said that, for the verification of systems with infinite state spaces, some areas are much better understood than others.

- For model checking infinite-state systems, the picture is already relatively clear. The decidability and complexity of most model checking problems for infinite-state systems is known (at least for those classes of processes in the PRS-hierarchy). The remaining open questions here are the following:
  - The exact complexity of the reachability problem for general Petri nets.
  - The exact complexity of the EF-model checking problem for PA-processes.

  Another very interesting open question in this area is, if there exists a polynomial algorithm for model checking finite-state systems with the full modal μ-calculus. However, this is not really a problem about infinite-state systems.
- In the area of semantic equivalence checking, the picture is not so complete.
  - Checking semantic equivalence of an infinite-state and a finite-state system is still the best-understood part of this area. The decidability and complexity of most of these problems is known, due to Theorem 11 and other results.

- For checking semantic finiteness, only partial results are known. In particular, the known upper and lower complexity bounds often do not match.
- For checking semantic equivalences (except weak bisimulation) between two infinite-state systems, the decidability of most problems is known. However, the exact computational complexity of most of these problems is open.
- Very little is yet known about checking weak bisimulation equivalence between two infinite-state systems, except for a few lower bounds. The problem is that this seems to require techniques to cope with infinitely branching graphs. The known methods for strong bisimulation do not carry over to weak bisimulation.

The future of this research area seems to require efforts in two directions. First, one can develop methods to solve the open questions mentioned above. Second, and perhaps even more importantly, one should develop efficient semi-algorithmic techniques for the verification of infinite-state systems, like, e.g., acceleration methods. These do not guarantee a solution in every case, but they can quickly find solutions for most problems that occur.

References

[ABJ98] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *10th Intern. Conf. on Computer Aided Verification (CAV'98)*. LNCS 1427, 1998.

[AJ93] P. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS'93*. IEEE, 1993.

[BCS95] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[BE97] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 5, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. In *Proc. of ICALP'97*, volume 1256 of *LNCS*, 1997.

[BHM01] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. In *Proc. of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *LNCS*. Springer Verlag, 2001.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. of STACS'99*, volume 1563 of *LNCS*. Springer Verlag, 1999.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. volume 131 of *LNCS*, pages 52–71, 1981.

[CFI96] G. Cécé, A. Finkel, and S.P. Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. of LICS'99*. IEEE, 1999.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. of FoSSaCS'99*, volume 1578 of *LNCS*, pages 14–30. Springer Verlag, 1999.

[Eme94] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Volume B, FORMAL MODELS AND SEMANTICS*. Elsevier, 1994.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[HJ99] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.

[Jan94] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proceedings of STACS'94*, volume 775 of *LNCS*. Springer Verlag, 1994.

[JE96] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.

[JKM01] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258:409–433, 2001.

[JM95] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.

[KM99] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.

[KM02a] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *Proc. of MFCS 2002*, volume 2420, pages 433–445. Springer Verlag, 2002.

[KM02b] A. Kučera and R. Mayr. Simulation preorder over simple process algebras. *Information and Computation*, 173(2):184–198, 2002.

[KM02c] A. Kučera and R. Mayr. Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time. *Theoretical Computer Science*, 270:667–700, 2002.

[LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*. Springer Verlag, 1998.

[May84] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460, 1984.

[May96] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In Martin Abadi and Takayasu Ito, editors, *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*. Springer Verlag, 1997.

[May97b] R. Mayr. Model checking PA-processes. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[May97c] R. Mayr. Tableau methods for PA-processes. In D. Galmiche, editor, *Analytic Tableaux and Related Methods (TABLEAUX'97)*, volume 1227 of *LNAI*. Springer Verlag, 1997.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.

[May00a] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP 2000*, volume 1853 of *LNCS*. Springer Verlag, 2000.

[May00b] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proc. of IFIP TCS 2000*, volume 1872 of *LNCS*. Springer Verlag, 2000.

[May00c] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[May00d] R. Mayr. Undecidable problems in unreliable computations. In *Proc. of LATIN 2000*, volume 1776 of *LNCS*. Springer Verlag, 2000. Journal version to appear in TCS.

[May01] R. Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 256:31–62, 2001.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS'77*. IEEE, 1977.

[Sén98] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS'98*. IEEE, 1998.

[SI94] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[Srb02] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proc. of STACS 2002*, volume 2285 of *LNCS*, pages 535–546. Springer Verlag, 2002.

[Sti98] C. Stirling. The joys of bisimulation. In *Proc. of MFCS'98*, volume 1450 of *LNCS*, pages 142–151. Springer Verlag, 1998.

[Sti01] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255:1–31, 2001.

[Tho93] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *Proc. of TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer Verlag, 1993.

[Wal00] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2000)*, volume 1974 of *LNCS*, pages 127–138. Springer Verlag, 2000.

Process Rewrite Systems

Richard Mayr

Institut für Informatik, Technische Universität München,
Arcisstr. 21, D-80290 München, Germany;
e-mail: mayrri@informatik.tu-muenchen.de
fax: +49 (89) 289-28207/28483
Web: http://www7.informatik.tu-muenchen.de/~mayrri

Abstract

Many formal models for infinite-state concurrent systems are equivalent to special classes of rewrite systems. We classify these models by their expressiveness and define a hierarchy of classes of rewrite systems. We show that this hierarchy is strict with respect to bisimulation equivalence.

The most general and most expressive class of systems in this hierarchy is called *Process Rewrite Systems* (PRS). They subsume Petri nets, PA-Processes and pushdown processes and are strictly more expressive than any of these. Intuitively, PRS can be seen as an extension of Petri nets by subroutines that can return a value to their caller. We show that the reachability problem is decidable for PRS. It is even decidable if there is a reachable state that satisfies certain properties that can be encoded in a simple logic. Thus PRS are more expressive than Petri nets, but not Turing-powerful.

1 Introduction

Petri nets and process algebras are two kinds of formalisms used to build abstract models of concurrent systems. These abstract models are used for verification, because they are normally smaller and more easily handled than full programs. Formal models should be simple enough to allow automated verification, or at least computer-assisted verification. On the other hand they should be as expressive as possible, so that most aspects of real programs can be modeled.

Many different formalisms have been proposed for the description of infinite-state concurrent systems. Among the most common are Petri nets, Basic Parallel Processes (BPP), context-free processes (BPA) and pushdown processes. BPP are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset. PA-Processes [BK85, Kuc, May97b] are the smallest common generalization of BPP and BPA. PA-processes, pushdown processes and Petri nets are mutually incomparable.

We present a unified view of all these formalisms by showing that they can be seen as special subclasses of rewrite systems. Such unified representations have already been used by Stirling, Caucal and Moller [Cau92, Mol96], but only for purely sequential or purely parallel systems. Here we generalize this to systems with both sequential and parallel composition.

Basically, the rewriting formalism is first order prefix-rewrite systems on process terms without substitution and modulo commutativity and associativity of parallel composition and associativity of sequential composition. The most general class of these systems will be called *Process Rewrite Systems (PRS)*. All the previously mentioned formalisms can be seen as special cases of PRS, and PRS is strictly more general (see Theorem 4.14). Intuitively, PRS can be seen as an extension of Petri nets by subroutines that can return a value to their caller. As PRS is a very expressive model, model checking with any temporal logic (except Hennessy-Milner logic) is undecidable for it (see Section 7). However, we show that the reachability problem is decidable for PRS. The interesting point here is that PRS is strictly more general than Petri nets, but still not Turing-powerful.

The rest of the paper is structured as follows. In Section 2 we define process terms and the rewriting formalism. We describe a hierarchy of subclasses of it, which we call the *PRS-hierarchy*. Section 3 explains the intuition for the

various classes in the PRS-hierarchy. In Section 4 we show that the PRS-hierarchy is strict with respect to bisimulation. In Section 5 we show that the reachability problem is decidable for PRS. Section 6 generalizes this result to reachability of certain classes of states that are described by state formulae. The paper closes with a section that summarizes the results.

2 Terms and Rewrite Systems

Many classes of concurrent systems can be described by a (possibly infinite) set of process terms, representing the states, and a finite set of rewrite rules describing the dynamics of the system.

Definition 2.1 Let $Act = \{a, b, \ldots\}$ be a countably infinite set of atomic actions and $Const = \{\epsilon, X, Y, Z, \ldots\}$ a countably infinite set of process constants. The process terms that describe the states of the system have the following form:

$$t ::= \epsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2$$

where ϵ is the empty term, $X \in Const$ is a process constant (used as an atomic process in this context), "$\|$" means parallel composition and "$.$" means sequential composition. Parallel composition is associative and commutative. Sequential composition is associative. Let $\mathcal{T}$ be the set of process terms.

Convention 1: We always work with equivalence classes of terms modulo commutativity and associativity of parallel composition and modulo associativity of sequential composition. Also we define that $\epsilon.t = t = t.\epsilon$ and $t \| \epsilon = t$.

Convention 2: We defined that sequential composition is associative. However, when we look at terms we think of it as left-associative. So when we say that a term t has the form $t_1.t_2$, then we mean that t_2 is either a single constant or a parallel composition of process terms.

The *size* of a process term is defined as the number of occurrences of constants in it plus the number of occurrences of operators in it.

$$\begin{aligned} size(\epsilon) &:= 0 \\ size(X) &:= 1 \\ size(t_1.t_2) &:= size(t_1) + size(t_2) + 1 \\ size(t_1 \| t_2) &:= size(t_1) + size(t_2) + 1 \end{aligned}$$

For a term t the set $Const(t)$ is the set of constants that occur in t.

$$\begin{aligned} Const(\epsilon) &:= \emptyset \\ Const(X) &:= \{X\} \\ Const(t_1.t_2) &:= Const(t_1) \cup Const(t_2) \\ Const(t_1 \| t_2) &:= Const(t_1) \cup Const(t_2) \end{aligned}$$

The dynamics of the system is described by a finite set of rules Δ of the form $(t_1 \xrightarrow{a} t_2)$ where t_1 and t_2 are process terms and $a \in Act$ is an atomic action. The finite set of rules Δ induces a (possibly infinite) labeled transition system with relations $\xrightarrow{a}$ with $a \in Act$. For every $a \in Act$, the transition relation $\xrightarrow{a}$ is the smallest relation that satisfies the following inference rules.

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 \| t_2 \xrightarrow{a} t_1' \| t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2}$$

where t_1, t_2, t_1', t_2' are process terms. Note that parallel composition is commutative and thus the inference rule for parallel composition also holds with t_1 and t_2 exchanged.

Since Δ is finite, the generated LTS is finitely branching. (For some classes of systems (e.g. Petri nets) the branching-degree is bounded by a constant that depends on Δ. For other classes (e.g. PA) the branching-degree is finite at every state, but it can get arbitrarily high.) Also every single Δ uses only a finite subset $Const(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} (Const(t_1) \cup Const(t_2))$ of constants and only a finite subset $Act(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} \{a\}$ of atomic actions. Thus for every Δ only finitely many of the generated transition relations $\xrightarrow{a_i}$ for $a_i \in Act$ are nonempty. (Those for which $a_i \in Act(\Delta)$). Still the generated transition system can be infinite. (Consider the analogy: Every labeled Petri net has only finitely many transitions and uses only finitely many different atomic actions, but the state space can be infinite.) The relation $\xrightarrow{a}$ is generalized to sequences of actions in the standard way. Sequences are denoted by σ.

Remark 2.2 *There is no operator "+" for nondeterministic choice in the process terms, because this is encoded in the set of rules Δ! There can be several rules with the same term on the left hand side. It is also possible that several rules are applicable at different places in a term. The rule that is applied and the position where it is applied are chosen nondeterministically.*

Also there is no such thing as action prefixes in the process terms. The atomic actions are introduced by the rules.

Many common models of systems fit into this scheme. In the following we characterize subclasses of rewrite systems. The expressiveness of a class depends on what kind of terms are allowed on the left hand side and right hand side of the rewrite rules in Δ.

Definition 2.3 (Classes of process terms)
We distinguish four classes of process terms:

1 Terms consisting of a single process constant like X.

S Terms consisting of a single constant or a sequential composition of process constants like $X.Y.Z$.

P Terms consisting of a single constant or a parallel composition of process constants like $X \| Y \| Z$.

G General process terms with arbitrary sequential and parallel composition like $(X.(Y \| Z)) \| W$.

Also let $\epsilon \in S, P, G$, but $\epsilon \notin 1$. It is easy to see the relations between these classes of process terms: $1 \subset S$, $1 \subset P$, $S \subset G$ and $P \subset G$. S and P are incomparable and $S \cap P = 1 \cup \{\epsilon\}$.

We characterize classes of process rewrite systems (PRS) by the classes of terms allowed on the left hand sides and the right hand sides of rewrite rules.

Definition 2.4 (PRS)
Let $\alpha, \beta \in \{1, S, P, G\}$. A (α, β)-PRS is a finite set of rules Δ where for every rewrite rule $(l \xrightarrow{a} r) \in \Delta$ the term l is in the class α and $l \neq \epsilon$ and the term r is in the class β (and can be ϵ). The initial state is given as a term $t_0 \in \alpha$. A (G, G)-PRS is simply called PRS.

Remark 2.5 *W.l.o.g. it can be assumed that the initial state t_0 of a PRS is a single constant. There are only finitely many terms $t_1, \ldots, t_n$ s.t. $t_0 \xrightarrow{a_i} t_i$. If t_0 is not a single constant then we can achieve this by introducing a new constant X_0 and new rules $X_0 \xrightarrow{a_i} t_i$ and declaring X_0 to be the initial state.*

(α, β)-PRS where α is more general than β or incomparable to β (for example $\alpha = G$ and $\beta = S$) do not make any sense. This is because the terms that are introduced by the right side of rules must later be matched by the left sides of other rules. So in a (G, S)-PRS the rules that contain parallel composition on the left hand side will never be used (assuming that the initial state is a single constant). Thus one may as well use a (S, S)-PRS. So we restrict our attention to (α, β)-PRS with $\alpha \subseteq \beta$.

Figure 1 shows a graphical description of the hierarchy of (α, β)-PRS.

Many of these (α, β)-PRS correspond to widely known models like Petri nets, pushdown processes, context-free processes and others.

1. A $(1, 1)$-PRS is a finite-state system. Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.

2. $(1, S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [BE97, Esp97]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.

3. It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left side of rules). Caucal [Cau92] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labeling of states. Thus (S, S)-PRS are equivalent to pushdown processes, the processes described by pushdown automata.

4. (P, P)-PRS are equivalent to Petri nets. Every constant corresponds to a place in the net and the number of occurrences of a constant in a term corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.

5. $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [BE97, Esp97]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [Chr93].

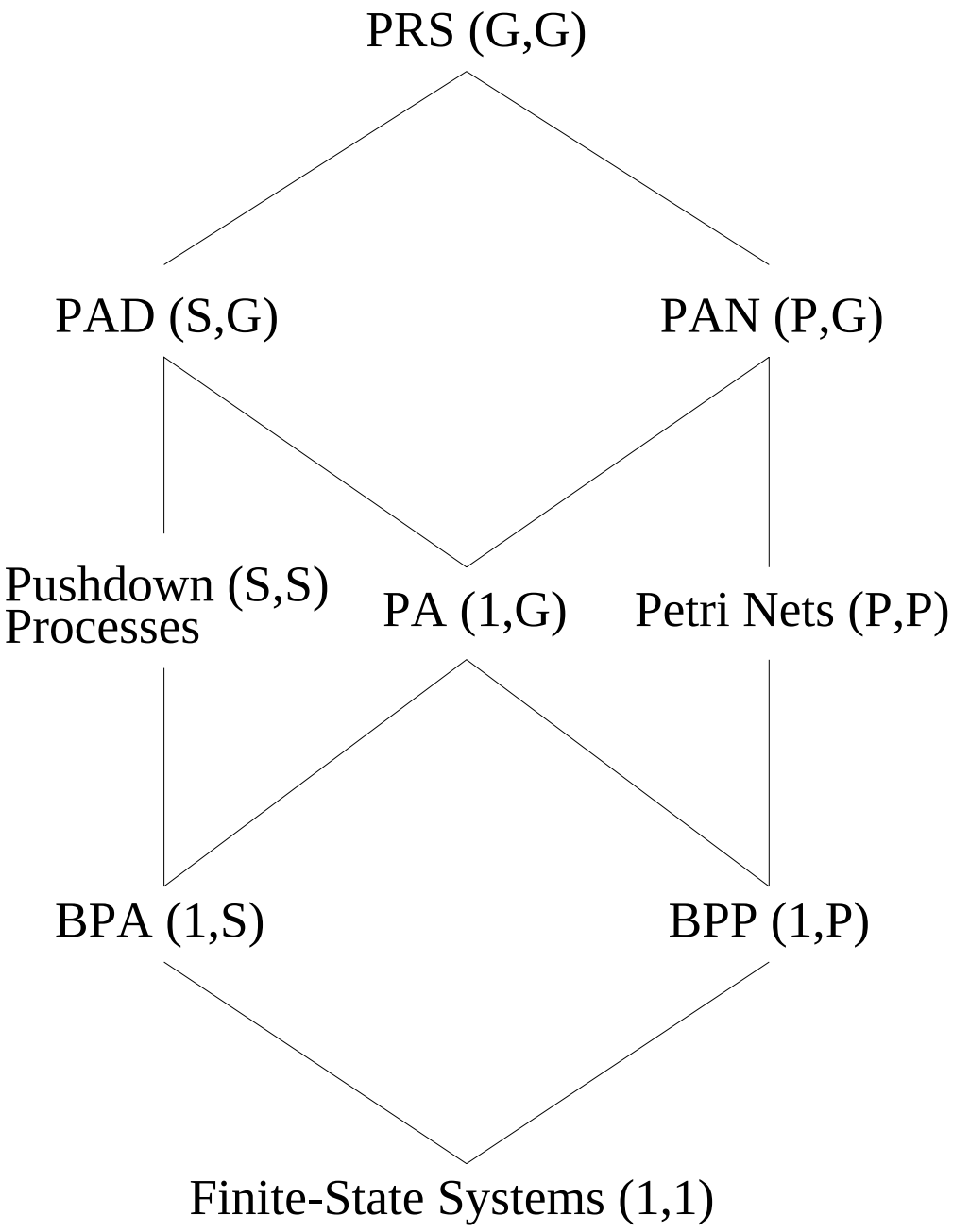


Figure 1: The PRS-hierarchy.

6. $(1, G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication (see [BK85, May97b, Kuc]).

7. (P, G)-PRS are called *PAN-processes* in [May97a]. It is the smallest common generalization of Petri nets and PA-processes and it is strictly more general than both of them (e.g. PAN can describe all Chomsky-2 languages while Petri nets cannot).

8. (S, G)-PRS are the smallest common generalization of pushdown processes and PA-processes. They are called *PAD* (PA + PD) in [May98].

9. The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [May97c]. They subsume all the previously mentioned classes.

3 The Intuition

In this section we explain the general intuition for the definition of (α, β)-PRS, i.e. what does it mean that parallel/sequential/arbitrary composition is allowed in terms on the left/right hand sides of rules?

If parallel composition is allowed on the right hand side of rules, then there can be rules of the form $t \xrightarrow{a} t_1 \| t_2$. This means that it is possible to create processes that run in parallel. The rule can be interpreted that, by action a, the process t becomes the process t_1 and spawns off the process t_2 or vice versa.

If sequential composition is allowed on the right hand side of rules, then there are rules of the form $t \xrightarrow{a} t_1.t_2$. The interpretation is that process t calls a subroutine t_1 and becomes process t_2. It resumes its execution when the subroutine t_1 terminates.

If arbitrary sequential and parallel composition is allowed on the right hand side of rules then both parallelism and subroutines are possible.

If parallel composition is allowed on the left hand side of rules, then there are rules of the form $t_1 \| t_2 \xrightarrow{a} t$. This can be interpreted as synchronization/communication of the parallel processes t_1 and t_2. This is because this action can only occur if both t_1 and t_2 change in a certain defined way.

If sequential composition is allowed on the left hand side of rules, then there can be rules of the form $t_1'.t_2 \xrightarrow{a} t'$ and $t_1''.t_2 \xrightarrow{a} t''$. The intuition is that a process t called a subroutine t_1 and became process t_2 by a rule $t \xrightarrow{a} t_1.t_2$. The subroutine may in its computation reach a state t_1' or t_1''. Now one of these rules is applicable. This means that the result of the computation of the subroutine affects the behavior of the caller when it becomes active again, since the caller can become t' or t''. The interpretation is that the subroutine returns a value to the caller when it terminates.

If arbitrary sequential and parallel composition is allowed on the left hand sides of rules then both synchronization and returning of values by subroutines are possible. It will be shown in Section 5 that rules with nested sequential and parallel composition (on the left side or the right side) do not increase the expressiveness. It suffices to have systems of rules where every single rules only contains either sequential or parallel composition.

4 The PRS-Hierarchy is Strict

The question arises if this hierarchy of (α, β)-PRS is strict. For the description of languages this is not the case, because for example context-free processes (BPA) and pushdown processes (PDA) both describe exactly the Chomsky-2 languages. However, the hierarchy is strict with respect to bisimulation equivalence. Bisimulation equivalence [Mil89] is a finer equivalence than language equivalence. It is defined as follows:

Definition 4.1 A binary relation R over the states of a labeled transition system is a *bisimulation* iff

$$\forall (s_1, s_2) \in R \; \forall a \in Act. \quad \begin{array}{l} (s_1 \xrightarrow{a} s_1' \;\Rightarrow \exists s_2 \xrightarrow{a} s_2'.\; s_1' R s_2') \wedge \\ (s_2 \xrightarrow{a} s_2' \;\Rightarrow \exists s_1 \xrightarrow{a} s_1'.\; s_1' R s_2') \end{array}$$

Two states s_1 and s_2 are *bisimilar* iff there is a bisimulation R such that $s_1 R s_2$. This definition can be extended to states in different transition systems by putting them 'side by side' and considering them as a single transition system. It is easy to see that there always exists a largest bisimulation which is an equivalence relation. It is called *bisimulation equivalence* or *bisimilarity* and it is denoted by $\sim$.

Definition 4.2 A class of processes A is more general than a class of processes B with respect to bisimulation iff the following two conditions are satisfied:

1. For every B-process there is a semantically equivalent A-process:
   $\forall t \in B. \exists t' \in A.\ t' \sim t$

2. There is an A-process that is not bisimilar to any B-process:
   $\exists t \in A. \forall t' \in B.\ t \not\sim t'$

It has already been established in [BCS96, Mol96] that the classes of finite-state systems, BPP, BPA, pushdown systems, PA and Petri nets are all different with respect to bisimulation. For PAD, PAN and PRS this remains to be shown.

The proof has two parts: First we show that there is a pushdown process that is not bisimilar to any PAN-process. Then we show that there is a Petri net that is not bisimilar to any PAD-process.

Definition 4.3 Consider the following pushdown system:

$$
\begin{array}{lll}
U.X \xrightarrow{a} U.A.X & U.A \xrightarrow{a} U.A.A & U.A \xrightarrow{b} U.B.A \\
U.X \xrightarrow{b} U.B.X & U.B \xrightarrow{b} U.B.B & U.B \xrightarrow{a} U.A.B \\
U.X \xrightarrow{c} V.X & U.A \xrightarrow{c} V.A & U.B \xrightarrow{c} V.B \\
U.X \xrightarrow{d} W.X & U.A \xrightarrow{d} W.A & U.B \xrightarrow{d} W.B \\
V.A \xrightarrow{a} V & V.B \xrightarrow{b} V & V.X \xrightarrow{e} V \\
W.A \xrightarrow{a} W & W.B \xrightarrow{b} W & W.X \xrightarrow{f} W
\end{array}
$$

with the initial state $U.X$. The execution sequences of this system are as follows: First it does a sequence of actions in $\{a, b\}^*$ and then one of two things:

1. A "c", the sequence in reverse and finally a "e".

2. A "d", the sequence in reverse and finally a "f".

Now we show that this pushdown system is not bisimilar to any PAN-process. First we need several definitions and lemmas.

Definition 4.4 Let t be an arbitrary process and σ a sequence of actions. The runs of t are its computations of maximal length. We define that $only(t, \sigma)$ is true iff the following conditions are satisfied:

- All runs of t are finite.
- All these runs do the sequence of actions σ.

Lemma 4.5 (Dickson's Lemma [Dic13])
Given an infinite sequence of vectors $M_1, M_2, M_3, \ldots$ in $\mathbb{N}^k$ there are $i < j$ s.t. $M_i \leq M_j$ ($\leq$ taken componentwise).

Remember that P is the class of process terms that contain only parallel composition; see Def. 2.3.

Lemma 4.6 *For every PAN Δ there is a sequence $\sigma \in \{a, b\}^*$ s.t. no $\alpha \in P$ satisfies any of the following two conditions:*

Cond1(σ, α)**:** $\exists \alpha_c.\ \alpha \xrightarrow{c} \alpha_c \wedge\ only(\alpha_c, \sigma\, e)$

Cond2(σ, α)**:** $\exists \alpha_d.\ \alpha \xrightarrow{d} \alpha_d \wedge\ only(\alpha_d, \sigma f)$

Proof We assume the contrary and derive a contradiction. So we assume that there is a PAN Δ s.t. for every $\sigma_i := a^i b$ ($i \in \mathbb{N}$) there is an $\alpha^i \in P$ s.t. *Cond1*(σ_i, α^i) or *Cond2*(σ_i, α^i).
There must be an infinite subsequence of $\alpha^1, \alpha^2, \ldots$ where *Cond1*(σ_i, α^i) is always satisfied or an infinite subsequence of $\alpha^1, \alpha^2, \ldots$ where *Cond2*(σ_i, α^i) is always satisfied. W.r. we assume that there is an infinite subsequence where *Cond1*(σ_i, α^i) is always satisfied. Now we only regard this infinite subsequence. Since Δ is finite, there are only finitely many different rules in Δ that are marked with the action c. Let $(t_1 \xrightarrow{c} t'_1), \ldots, (t_n \xrightarrow{c} t'_n)$ be those rules. (Note that $t_i \in P$ for every i, because Δ is a PAN. However, t'_i need not be in P.) It follows that one of these rules must be used infinitely often to obtain α^i_c from α^i. Let this rule be $(t_k \xrightarrow{c} t'_k)$ for some $k \in \{1, \ldots, n\}$. Thus there is an infinite subsequence of the sequence $\alpha^1, \alpha^2, \ldots$ where only this rule is used to obtain α^i_c from α^i. Now we consider only this infinite subsequence.
We regard the sequence α^i of the α that satisfy Cond1. *Const*(Δ) is finite and $\alpha^i \in P$. Moreover, all α_i only contain constants from the finite set

$Const(\Delta)$. Thus we can apply Dickson's Lemma. By Dickson's Lemma there are $j, j' \in \mathbb{N}$ s.t. $j' > j$ and $\alpha^{j'} \geq \alpha^j$ (this means $\alpha^{j'} = \alpha^j \| \beta$ for some $\beta \in P$).

For both α^j and $\alpha^{j'}$ the rule $(t_k \xrightarrow{c} t'_k)$ is used to obtain α_c^j, $\alpha_c^{j'}$. Thus $\alpha^j = t_k \| \gamma$ for some $\gamma \in P$ and $\alpha_c^j = t'_k \| \gamma$. Also we have $\alpha^{j'} = \alpha^j \| \beta = t_k \| \gamma \| \beta$ and $\alpha_c^{j'} = t'_k \| \gamma \| \beta = \alpha_c^j \| \beta$. By Cond1 we have $only(\alpha_c^j, \sigma_j e)$ and $only(\alpha_c^{j'}, \sigma_{j'} e)$. However, $\alpha_c^{j'}$ also enables the sequence $\sigma_j e$. This is a contradiction. ■

Lemma 4.7 *For every PAN Δ there is a sequence $\Sigma \in \{A, B\}^*$ s.t. no process term t (w.r.t. Δ) is bisimilar to the pushdown system $U.\Sigma.X$ of Def. 4.3.*

Proof We assume the contrary and derive a contradiction. Assume that there is a PAN Δ s.t. for every sequence $\Sigma \in \{A, B\}^*$ there is a term $t(\Sigma)$ s.t. $t(\Sigma) \sim U.\Sigma.X$. For every Σ let $t(\Sigma)$ be the smallest term that has this property.

For any sequence $\Sigma \in \{A, B\}^*$ let $\sigma(\Sigma)$ be the sequence of actions a and b that is obtained by converting Σ to lowercase letters.

It follows from the definition of bisimulation that no process that has only finite computations can be bisimilar to a process that has an infinite computation. Thus by Def. 4.3 it follows that for every sequence $\Sigma \in \{A, B\}^*$ and every state $t(\Sigma)$ the following properties hold:

C There is a state $t_c(\Sigma)$ s.t. $t(\Sigma) \xrightarrow{c} t_c(\Sigma)$ and $t_c(\Sigma) \sim V.\Sigma.X$ and thus $only(t_c(\Sigma), \sigma(\Sigma)e)$.

D There is a state $t_d(\Sigma)$ s.t. $t(\Sigma) \xrightarrow{d} t_d(\Sigma)$ and $t_d(\Sigma) \sim W.\Sigma.X$ and thus $only(t_d(\Sigma), \sigma(\Sigma)f)$.

For every $t(\Sigma)$ the action c disables the action d and vice versa. Thus the actions c and d must both occur in the same subterm α of $t(\Sigma)$ and $\alpha \in P$. (This is because in a PAN no single action can change two separate subterms. For example in the term $(t_1.t_2) \| t_3$ (where t_1, t_2 and t_3 are not ϵ) no single action can change both t_1 and t_3.) Let α be the maximal parallel subterm of $t(\Sigma)$ where the actions c or d occur. This means that α is part of a subterm of the form $\alpha.\beta$ or $\alpha \| (\beta.\gamma)$, but not of the form $\alpha \| \beta$ for some $\beta \in P$. It follows that α cannot immediately synchronize with the rest of the term $t(\Sigma)$.

We have that $\alpha \xrightarrow{c} \alpha_c$ and $\alpha \xrightarrow{d} \alpha_d$. Let $t(\Sigma)[\alpha \to \alpha']$ be the term that one gets by replacing this one particular α in $t(\Sigma)$ by α'. (Not every subterm

α is replaced by α' !) This means that $t_c(\Sigma) = t(\Sigma)[\alpha \to \alpha_c]$ and $t_d(\Sigma) = t(\Sigma)[\alpha \to \alpha_d]$.

Without restriction we now assume that Σ begins with A (the other case is symmetric). Then $t(\Sigma)_c$ ($t(\Sigma)_d$) must enable action a, but not action b. We show that the action a must be enabled by a subterm of $t(\Sigma)_c$ ($t(\Sigma)_d$) that is different from α_c (α_d). We assume the contrary and derive a contradiction. In this case the rest of $t(\Sigma)_c$ ($t(\Sigma)_d$), without α_c (α_d), enables neither a nor b. It follows that the rest of $t(\Sigma)_c$ ($t(\Sigma)_d$) cannot do action a or b before α_c (α_d) terminates. If α_c (α_d) does not terminate then by Lemma 4.6 the conditions **C** and **D** cannot be satisfied for some Σ, a contradiction. If α_c (α_d) does terminate then for some suffixes σ', σ'' of $\sigma(\Sigma)$ we get $only(t(\Sigma)[\alpha \to \epsilon], \sigma' e)$ and $only(t(\Sigma)[\alpha \to \epsilon], \sigma'' f)$. Again this is a contradiction.

Thus the action a must be enabled at a subterm of $t(\Sigma)_c$ ($t(\Sigma)_d$) that is different from α_c (α_d). As we have $t(\Sigma) \sim U.\Sigma.X$ and $U.\Sigma.X \stackrel{b}{\to} U.B.\Sigma.X$ there must be a t' s.t. $t \stackrel{b}{\to} t'$ and $t' \sim U.B.\Sigma.X$. This action b must occur in α, because the rest of $t(\Sigma)$ cannot do b. Thus $\alpha \stackrel{b}{\to} \alpha'$ and $t' = t(\Sigma)[\alpha \to \alpha']$. We have $U.B.\Sigma.X \stackrel{c}{\to} V.B.\Sigma.X$. As the rest of t' (without α') cannot do action c we get $\alpha' \stackrel{c}{\to} \alpha''$ and $t(\Sigma)[\alpha \to \alpha''] \sim V.B.\Sigma.X$. However now the rest of the term $t(\Sigma)[\alpha \to \alpha'']$ (without α'') can do action a, but $V.B.\Sigma.X$ cannot. Thus $t(\Sigma)[\alpha \to \alpha''] \not\sim V.B.\Sigma.X$ and we have a contradiction. ■

Lemma 4.8 *The pushdown system $U.X$ of Def. 4.3 is not bisimilar to any PAN Δ with initial state t_0.*

Proof We assume the contrary and derive a contradiction. Assume that there is a PAN Δ with initial state t_0 s.t. $t_0 \sim U.X$. Let Σ be the sequence from Lemma 4.7. (Note that Σ depends on Δ.) The process $U.X$ can reach the state $U.\Sigma.X$. Thus t_0 must be able to reach a state t s.t. $t \sim U.\Sigma.X$. By Lemma 4.7 such a term t does not exist, a contradiction. ■

It follows directly that the pushdown system from Def. 4.3 is not bisimilar to any PA-process either. However, as PAD and PRS subsume pushdown processes, it is a PAD and PRS-process. Thus PAD is strictly more general then PA and PRS is strictly more general than PAN. PAD subsumes BPP and BPP is incomparable to pushdown systems. Thus PAD is also more general than pushdown processes. Now we show that there is a Petri net that is not bisimilar to any PAD-process.

Definition 4.9 Consider the following Petri net (given as a (P, P)-PRS).

$$X \xrightarrow{g} X\|A\|B \qquad X \xrightarrow{c} Y \qquad Y\|A \xrightarrow{a} Y \qquad Y\|B \xrightarrow{b} Y$$
$$X\|A \xrightarrow{d} Z \qquad X\|B \xrightarrow{d} Z \qquad Y\|A \xrightarrow{d} Z \qquad Y\|B \xrightarrow{d} Z$$

The initial state is $X\|A\|B$.

Lemma 4.10 *If there is a PAD-process that is bisimilar to the state $X\|A\|B$ of the Petri net of Def. 4.9, then there is also a pushdown process that is bisimilar to $X\|A\|B$.*

Proof Let Δ be a PAD and Q the initial state s.t. $Q \sim X\|A\|B$. W.r. we can assume that Q is a single constant (see Def. 2.4). We construct a pushdown process (an (S, S)-PRS) Δ' that is also bisimilar to $X\|A\|B$.

First we show that in every reachable state of Δ of the form $(t_1\|t_2).t_3$ (t_3 can be ϵ) t_1 or t_2 must be deadlocked.

Assume that there is a state $(t_1\|t_2).t_3$ that is reachable from Q. Then a state M must be reachable from $X\|A\|B$ s.t. $(t_1\|t_2).t_3 \sim M$. There are two cases:

1. If M is deadlocked then t_1 and t_2 must be deadlocked.
2. If M is not deadlocked then there is an M' s.t. $M \xrightarrow{d} M'$ and M' is deadlocked. By the definition of PAD a single action d can only change t_1 or t_2, but not both. Thus either t_1 or t_2 must be deadlocked.

Thus if parallel composition occurs in a state that is reachable from Q, then all but one part of it must be deadlocked. Since Q is a single constant, parallel composition can only be introduced by PAD-rules. If such a rule has the form $(u \xrightarrow{x} u_1\|u_2) \in \Delta$ for some action x, then u_1 or u_2 must be deadlocked. W.r. let u_1 be deadlocked. However, the term $u_1.t$ for some term t is not necessarily deadlocked. Thus in Δ' we replace the rule $(u \xrightarrow{x} u_1\|u_2)$ by the rule $u \xrightarrow{x} u_2.u_1$. The new system is equivalent up to bisimulation. (We assume w.r. that u_2 cannot influence u_1. This means that there is no rule in Δ' whose left hand side is $v_2.v_1$ where v_2 is a nonempty suffix of u_2 and v_1 is a nonempty prefix of u_1. This can be achieved by renaming of constants in u_2 and Δ' if necessary.)

The other case where parallel composition occurs in a rule in Δ is when a rule has the form $u \xrightarrow{x} u_1.(u_2\|u_3).u_4$, where u_1 or u_4 can be ϵ. There are two cases:

1. If u_1 can terminate then the term $(u_2 \| u_3)$ can become active. Therefore u_2 or u_3 must be deadlocked. W.r. let u_2 be deadlocked. Then in Δ' we replace this rule by the rule $u \xrightarrow{x} u_1.u_3.u_2.u_4$. Note that u_2 is deadlocked, but $u_2.u_4$ is not necessarily deadlocked. (We assume w.r. that u_1 cannot influence u_3 and u_3 cannot influence u_2. This can be achieved by renaming of constants in u_1 and u_3 and Δ' if necessary.)

2. If u_1 cannot terminate then in Δ' we replace this rule by the equivalent rule $u \xrightarrow{x} u_1$.

Thus we get a new system Δ' that is equivalent to Δ up to bisimulation, but Δ' does not contain parallel composition. Thus, if the preconditions are satisfied, the (S, S)-PRS Δ' with initial state Q is bisimilar to $X \| A \| B$. This is the pushdown process that we are looking for. ■

Definition 4.11 Let Δ be a (α, β)-PRS for $\alpha, \beta \in \{1, S, P, G\}$ and t_0 the initial state. The language generated by this system is the set of all sequences σ s.t. $\exists t.\ t_0 \xrightarrow{\sigma} t$ and t is deadlocked.

Lemma 4.12 *If a process t is bisimilar to a pushdown process then the language generated by t is a context-free language.*

Proof Directly from Def. 4.1 and the definition of pushdown processes. ■

Lemma 4.13 *The Petri net of Def. 4.9 is not bisimilar to any PAD-process.*

Proof We assume the contrary and derive a contradiction. If there is a PAD-process that is bisimilar to the Petri net of Def. 4.9, then by Lemma 4.10 there is a pushdown process that is bisimilar to this Petri net. Then by Lemma 4.12 the Petri net of Def. 4.9 generates a context-free language L. By the definition of this Petri net L is

$$\{g^m c\sigma \mid m \geq 0 \wedge \sigma \in \{a,b\}^* \wedge \#_a\sigma = m+1 \wedge \#_b\sigma = m+1\} \cup$$
$$\{g^m d \mid m \geq 0\} \cup$$
$$\left\{ g^m c\sigma d \;\middle|\; \begin{array}{l} m \geq 0 \wedge \sigma \in \{a,b\}^* \wedge \\ \#_a\sigma \leq m+1 \wedge \#_b\sigma \leq m+1 \wedge \#_a\sigma + \#_b\sigma \leq 2m+1 \end{array} \right\}$$

It follows that $L \cap g^*ca^*b^* = \{g^m c a^{m+1} b^{m+1} \mid m \geq 0\}$. By applying the pumping lemma for context-free languages [HU79] it is easy to show that L is not context-free. Thus we have a contradiction. ■

It follows that PAD and PAN are incomparable and PRS is strictly more general than PAD. By combining these results with the other results above we get the following theorem.

Theorem 4.14 *The PRS-hierarchy is strict with respect to bisimulation.*

5 The Reachability Problem

In this section we show that the reachability problem is decidable for PRS. Thus PRS are not Turing-powerful.

REACHABILITY

Instance: A PRS Δ with initial state t_0 and a given state t.

Question: Is the state t reachable from t_0 ? Formally: Is there a sequence of actions σ s.t. $t_0 \xrightarrow{\sigma} t$?

For Petri nets reachability is decidable and *EXPSPACE*-hard [May84, Lip76]. Here we show that reachability is decidable for PRS by reducing the problem to the reachability problem for Petri nets. As the atomic actions are not important for reachability, we'll ignore them for the rest of this section and write just $t_1 \to t_2$ instead of $t_1 \xrightarrow{a} t_2$.

We prove the decidability of reachability in two steps. First we show that it suffices to decide the problem for a special class of PRS, the PRS in transitive normal form (see below). Then we solve the problem for this subclass of PRS.

Definition 5.1 For a PRS Δ and process terms $t, t' \in \mathcal{T}$ we define

$$t \succ^{\Delta} t' \;:\Longleftrightarrow\; \exists \sigma.\ t \xrightarrow{\sigma} t'$$

where σ is a sequence of applications of rules in Δ. If Δ is fixed, then we just write $t \succ t'$.

Δ is in *normal form* iff all rules in Δ are in normal form. A rule is in normal form if it has one of the following two forms:

Par-Rule $X_1 \| X_2 \to Y$ or $X \to Y_1 \| Y_2$ or $X \to Y$.

Seq-Rule $X_1 . X_2 \to Y$ or $X \to Y_1 . Y_2$ or $X \to Y$.

where X, Y, X_i, Y_i are process constants and Y can be ϵ.
The only rules that are both seq-rules and par-rules are of the form $X \to Y$.
The following relations $\succ^{\Delta}_{par}$ and $\succ^{\Delta}_{seq}$ are technicalities used in the proofs.

$$t \succ^{\Delta}_{par} t' \iff \exists\sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are par-rules from } \Delta$$
$$t \succ^{\Delta}_{seq} t' \iff \exists\sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are seq-rules from } \Delta$$

A PRS Δ is in *transitive normal form* iff it is in normal form and for all $X, Y \in Const$

$$X \succ^{\Delta} Y \ \Rightarrow\ (X \to Y) \in \Delta$$

Proposition 5.2 *Let Δ be a PRS in transitive normal form and t_1, t_2 process terms that do not contain the operator for sequential composition. It is decidable if $t_1 \succ^{\Delta}_{par} t_2$.*

Proof This follows directly from the decidability of the reachability problem for Petri nets [May84]. ■

The reachability problem for PRS is reducible to the reachability problem for PRS in normal form.

Lemma 5.3 *Let Δ be a PRS and $t_1, t_2 \in \mathcal{T}$.*
Then a PRS Δ' in normal form and terms t_1' and t_2' can be effectively constructed s.t. Δ', t_1' and t_2' use only constants from the finite set V' (with $Const(\Delta) \subseteq V' \subset Const$) and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$.

Proof For any rule $(u_1 \to u_2)$ in Δ let

$$norm(u_1 \to u_2) := size(u_1) + size(u_2)$$

Let k_i be the number of rules $(u_1 \to u_2)$ in Δ that are not in normal form and $norm(u_1 \to u_2) = i$. Let n be the maximal i s.t. $k_i \neq 0$. (n exists because Δ is finite). We define $Norm(\Delta) := (k_n, k_{n-1}, \ldots, k_1)$. These norms are ordered lexicographically. Δ is in normal form iff $Norm(\Delta) = (0, \ldots, 0)$. Now we describe a procedure that transforms Δ into a new PRS Δ' and terms t_1, t_2 into t_1', t_2' s.t. $Norm(\Delta') <_{lex} Norm(\Delta)$ and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$.

Remember that sequential composition is left-associative. This means that the term $X.Y.Z$ is $(X.Y).Z$. It has the subterms X, Y, Z and $X.Y$, but not $Y.Z$. However, the term $X.(Y\|Z)$ has a subterm $Y\|Z$.

If $Norm(\Delta) \neq (0,\ldots,0)$ then there is a rule in Δ that is not in normal form. Take a non-constant subterm t of this rule and replace every subterm t in Δ and in t_1 and t_2 by a new constant X. Then add two rules $X \to t$ and $t \to X$. This yields a new set of rules Δ' and t_1' and t_2'. By the definition of *Norm* and *size* we get $Norm(\Delta') <_{lex} Norm(\Delta)$. The constant X serves as an abbreviation for the term t. There are only two problems:

1. A rule is applicable to a subterm of t, but not to X. For example $t = Y.Z.W$ and there is a rule $Y.Z \to V$. In this case the rule $X \to t$ must be applied first. So the term X can be rewritten to $V.W$ in two steps.

2. During the rewriting a subterm t is created. However, the rule that contains t as a subterm on the left side is no longer applicable, because the subterm t has been replaced by X. For example let $t = Y\|Z$, the initial state is $(W\|Z).W$ and there are rules $W \to Y$ and $(Y\|Z).W \to V$. By the above algorithm the rule $(Y\|Z).W \to V$ has been transformed into $X.W \to V$ and rules $X \to Y\|Z$ and $Y\|Z \to X$ have been added. The initial state $(W\|Z).W$ can be rewritten to $(Y\|Z).W$, but now the changed rule $X.W \to V$ is not applicable. However, by applying the new rule $Y\|Z \to X$ first we get $X.W$ and can finally rewrite the term to V.

Thus we get

$$t_1 \succ^{\Delta} t_1 \iff t_1' \succ^{\Delta'} t_2'$$

By repeating this algorithm we finally get a set of rules Δ'' and terms t_1'' and t_2'' s.t. $Norm(\Delta'') = (0,\ldots,0)$ and

$$t_1 \succ^{\Delta} t_1 \iff t_1'' \succ^{\Delta''} t_2''$$

Δ'' is in normal form. ■

The following lemma will be used to prove the correctness of the algorithm in Lemma 5.5.

Lemma 5.4 *Let Δ be a PRS in normal form. If there are constants X, Y s.t. $X \succ^{\Delta} Y$ and $(X \to Y) \notin \Delta$, then there are also constants X', Y' with $(X' \to Y') \notin \Delta$ and $X' \succ^{\Delta}_{par} Y'$ or $X' \succ^{\Delta}_{seq} Y'$.*

Proof It follows from the preconditions that we can choose a pair of constants X', Y' s.t. $(X' \to Y') \notin \Delta$ and $X' \xrightarrow{\sigma} Y'$ for a sequence σ of minimal length. More precisely the length of σ is minimal over the choice of X', Y' and σ.

Now we show that $X' \succ^{\Delta}_{par} Y'$ or $X' \succ^{\Delta}_{seq} Y'$. We do this by assuming the contrary and deriving a contradiction. We say that a rule is trivial if it has the form $(X'' \to Y'')$. We assume that σ contains both seq-rules and par-rules that are nontrivial. There are two cases:

1. The last nontrivial rule in σ is a par-rule. If a seq-rule $Z_1 \to Z_2.Z_3$ occurs in σ then there is a subsequence σ' of σ and a constant Z_4 s.t. $Z_2.Z_3 \xrightarrow{\sigma'} Z_4$. This contradicts the minimality of the length of σ.

2. The last nontrivial rule in σ is a seq-rule. This seq-rule must have the form $Z_1.Z_2 \to Z$. The first nontrivial par-rule that occurs in σ must have the form $Z' \to Z'_1 \| Z'_2$. Then there is a subsequence σ' of σ and a constant Z'' s.t. $Z' \xrightarrow{\sigma'} Z''$. This contradicts the minimality of the length of σ.

Thus σ consists either only of applications of par-rules (and thus $X' \succ^{\Delta}_{par} Y'$) or only of seq-rules (and thus $X' \succ^{\Delta}_{seq} Y'$). ■

Lemma 5.5 *Let Δ be a PRS in normal form. Then a PRS Δ' in transitive normal form can be effectively constructed s.t.*

$$\forall t_1, t_2 \in \mathcal{T}.\ t_1 \succ^{\Delta'} t_2 \iff t_1 \succ^{\Delta} t_2$$

Proof It suffices to find all pairs of constants X, Y s.t. $X \succ^{\Delta} Y$ and to add the rules $(X \to Y)$ to Δ. By Lemma 5.4 it suffices to check $X \succ^{\Delta}_{par} Y$ and $X \succ^{\Delta}_{seq} Y$. This is decidable because of Proposition 5.2 and the decidability of the reachability problem for pushdown processes (see [BEM97]). Lemma 5.4 basically says that while there are new rules to add we can find at least one to add.

The algorithm is as follows:

$\Delta' := \Delta$; flag := true;
While flag **do**
  flag := false;
  For every pair of constants X, Y with $(X \to Y) \notin \Delta'$ **do**
    If $X \succ_{par}^{\Delta'} Y$ or $X \succ_{seq}^{\Delta'} Y$ **then** ($\Delta' := \Delta' \cup (X \to Y)$; flag := true) **fi**;
  od;
od;

■

Theorem 5.6 *The reachability problem is decidable for PRS. The complexity is polynomially equivalent to reachability for Petri nets.*

Proof Let Δ be a PRS and $t_1, t_2 \in \mathcal{T}$. The question is if $t_1 \succ^{\Delta} t_2$.

We construct a new PRS Δ' by adding new constants X_1 and X_2 and rules $X_1 \to t_1$ and $t_2 \to X_2$. It follows that $t_1 \succ^{\Delta} t_2 \Leftrightarrow X_1 \succ^{\Delta'} X_2$. Then we use Lemma 5.3 and transform Δ' into a PRS Δ'' in normal form. Normally the terms X_1, X_2 would also change in this transformation, but since they are single constants they stay the same. This procedure adds at most $2k$ new rules, where k is the number of non-constant strict subterms of rules in Δ. Thus $k = \mathcal{O}(n^2)$ and $size(\Delta')$ is polynomial in $size(\Delta)$. We get $t_1 \succ^{\Delta} t_2 \Leftrightarrow X_1 \succ^{\Delta''} X_2$. Then we use Lemma 5.5 to transform Δ'' into a PRS Δ''' in transitive normal form. It follows that $t_1 \succ^{\Delta} t_2 \Leftrightarrow X_1 \succ^{\Delta'''} X_2$. Since $|Const(\Delta)| = \mathcal{O}(n)$ there are $\mathcal{O}(n^2)$ pairs of constants. Thus the algorithm of Lemma 5.5 uses $\mathcal{O}(n^2)$ instances of the reachability problem for Petri nets and for pushdown processes in every instance of the loop. The loop is done at most $\mathcal{O}(n^2)$ times. Thus it uses at most $\mathcal{O}(n^4)$ instances of the reachability problem for Petri nets and pushdown processes. Since Δ''' is in transitive normal form we have

$$t_1 \succ^{\Delta} t_2 \Leftrightarrow X_1 \succ^{\Delta'''} X_2 \Leftrightarrow (X_1 \to X_2) \in \Delta'''$$

The condition $(X_1 \to X_2) \in \Delta'''$ is trivial to check.

The reachability problem for pushdown processes is polynomial [BEM97]. The algorithm for PRS uses only polynomially many instances of Petri net reachability. Since PRS are more general than Petri nets, it follows that reachability for PRS is polynomially equivalent to Petri net reachability. ■

6 The Reachable Property Problem

In the previous section the problem was if one given state is reachable. Here we consider the question if there is a reachable state that has certain properties. We call this problem the *reachable property problem*. Unlike for reachability, the atomic actions are important for this problem. Properties are described by state formulae that have the following syntax:

$$\Phi := a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2$$

The denotation $[\![\Phi]\!]$ of a state formula Φ is a (possibly infinite) set of process terms.

$$\begin{aligned} [\![a]\!] &:= \{t \mid \exists t'.\, t \xrightarrow{a} t'\} \\ [\![\neg\Phi]\!] &:= \mathcal{T} - [\![\Phi]\!] \\ [\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\ [\![\Phi_1 \vee \Phi_2]\!] &:= [\![\Phi_1]\!] \cup [\![\Phi_2]\!] \end{aligned}$$

To simplify the notation we use sets of actions. Let $A := \{a_1, \ldots, a_k\} \subseteq Act$.

$$\begin{aligned} [\![A]\!] &:= [\![a_1]\!] \cap \ldots \cap [\![a_k]\!] \\ [\![-A]\!] &:= [\![\neg a_1]\!] \cap \ldots \cap [\![\neg a_k]\!] \end{aligned}$$

By transformation to disjunctive normal form every state-formula Φ can be written as $(A_1 \wedge -B_1) \vee \ldots \vee (A_n \wedge -B_n)$, where $A_i, B_i \subseteq Act$.
We consider the question if there is a reachable state that satisfies a given state formula. To express this problem, we define another operator.

$$[\![\Diamond\Phi]\!] := \{t \mid \exists \sigma, t'.\, t \xrightarrow{\sigma} t' \in [\![\Phi]\!]\}$$

Note that state-formulae do not contain the operator $\Diamond$. Let $t \in \mathcal{T}$ be a process term. For $t \in [\![\Phi]\!]$ we also write $t \models \Phi$.

Reachable property problem

Instance: A PRS Δ with initial state t_0 and a state-formula Φ.
Question: $t_0 \models \Diamond\Phi$?

We prove the decidability of the reachable property problem for PRS in two steps. First we show that it suffices to solve the problem for PRS in transitive normal form.

Lemma 6.1 *Let Δ be a PRS that uses only constants from the finite set $Const(\Delta) \subset Const$, and let $t_0 \in \mathcal{T}$ be a process term.*

Then a PRS Δ' in normal form and a term t_0' can be effectively constructed s.t. for every state formula Φ, $t_0 \models \Diamond\Phi$ with respect to Δ iff $t_0' \models \Diamond\Phi$ with respect to Δ'.

Proof We use the same algorithm to transform Δ and t as in Lemma 5.3. The new rules that are added are labeled with the new (silent) action τ, that doesn't occur in Φ.

The only problem that remains is that if a subterm t is replaced by a new constant X, then X does not enable the same actions as t. Thus, for example, the new system might satisfy a formula $\Diamond(\neg a)$, although the original doesn't. The solution is as follows: Compute the set of actions $\{b_1, \ldots, b_m\}$ that are enabled by the term t in w.r.t. Δ. Then add new rules $X \xrightarrow{b_1} X$, ..., $X \xrightarrow{b_m} X$. This must be done after every step where a subterm a replaced by a constant. Then the new system Δ', t_0' satisfies exactly the same formulae $\Diamond\Phi$ as the old one. ■

Lemma 6.2 *Let Δ be a PRS in normal form. Then a PRS Δ' in transitive normal form can be effectively constructed s.t. for every term t and every state-formula Φ, $t \models \Phi$ w.r.t Δ iff $t \models \Phi$ w.r.t Δ'.*

Proof We use the same algorithm as in Lemma 5.5. The only difference is that we label the newly added rules with the special (silent) action τ that does not occur in any state-formula. ■

Remark 6.3 *By Lemma 6.1 and Lemma 6.2 it follows that it suffices to solve the reachable property problem for PRS in transitive normal form. Let there be a PRS Δ in transitive normal form with initial state t_0 and Φ a state-formula. The problem is if $t_0 \models \Diamond\Phi$. As Φ can be transformed into disjunctive normal form and*

$$t \models \Diamond(\Phi_1 \vee \Phi_2) \iff t \models \Diamond(\Phi_1) \vee t \models \Diamond(\Phi_2)$$

it suffices to show decidability for formulae of the form $\Diamond(A \wedge -B)$, where $A, B \subseteq Act$.

The following definition and lemma by Jančar [Jan90] are used to show the effectiveness of the procedures *check* and *check'* that are used to show the decidability of the reachable property problem.

Definition 6.4 For a given Petri net N the set L_N of formulae is defined as follows:

- There is one variable M that stands for a marking of the net.
- A term is either
  - a term $M(p)$ where p is a place, or
  - a constant $c \in \mathbb{N}$, or
  - of the form $t_1 + t_2$.
- A formula is either
  - an atomic formula $t_1 < t_2$ or $t_1 \leq t_2$, where t_1, t_2 are terms, or
  - of the form $f_1 \& f_2$ where f_1, f_2 are formulae.

For a concrete marking M, $f(M)$ denotes the instance of f with this M. The semantics is natural.

Lemma 6.5 *([Jan90])*
For a Petri net N with initial marking M_0 it is decidable if there is a reachable marking M s.t. $f(M)$.

Definition 6.6 Let $C \subset Const$ and $t \in P$. Let h be a function s.t. $h(C,t)$ is true iff t contains only constants from C and false otherwise.
Let Δ be a PRS in transitive normal form, $X \in Const$ and let A, A', B be finite sets of actions. Let j be a mapping $j : 2^A \mapsto 2^{Const}$.
$check(X, j, A', B)$ iff there exists a $t \in P$ s.t. $X \succ^{\Delta}_{par} t$ and

$$(t = t' \| \|_{(C \in 2^A)} t_c) \ \wedge \ t' \models (A' \wedge -B) \ \wedge \bigwedge_{C \in 2^A} h(j(C), t_c)$$

and the additional constraint that $t \in Const \Rightarrow t' = \epsilon$.
$check'(X, A, B)$ iff there exists a $t \in P$ s.t. $X \succ^{\Delta}_{par} t$ and $t \notin Const$ and

$$t \models (A \wedge -B)$$

Lemma 6.7 *The functions check and check′ are decidable.*

Proof Directly from Lemma 6.5, because par-rules correspond to Petri net transitions. ■

Definition 6.8 The function *snd* returns the nesting-depth of sequential composition in a process term.

$$\begin{aligned} snd(\epsilon) &:= 0 \\ snd(X) &:= 0 \\ snd(t_1 \| t_2) &:= max(snd(t_1), snd(t_2)) \\ snd(t_1.t_2) &:= max(snd(t_1)+1, snd(t_2)) \end{aligned}$$

Definition 6.9 Let Δ be a PRS in transitive normal form, $X \in Const$, $n \in \mathbb{N}$ and A, B finite sets of actions.
Let $reach(X, n, A, B)$ be true iff there exists a term t s.t. $t \notin Const$, $X \succ^{\Delta} t$, $t \models (A \wedge -B)$ and the nesting-depth of sequential composition in t is at most n, i.e. $snd(t) \leq n$.
The function *reachseq* is defined like *reach*, except that the first rule applied to X must be a seq-rule of the form $X \stackrel{a}{\rightarrow} Y.Z$ and Z is never changed afterwards. (This implies that *reachseq* is only defined for $n \geq 1$.)

Now we describe recursive algorithms for *reach* and *reachseq*.

1 $reach(X, n, A, B)$
2 case $n = 0$:
3 return$(check'(X, A, B))$;
4 case $n > 0$:
5 for every mapping $j : 2^A \longmapsto 2^{Const}$ and every $A' \subseteq A$
6 if $A' \cup \bigcup_{C \in 2^A \wedge j(C) \neq \emptyset} C = A$ then
7 if $check(X, j, A', B)$ then
8 if $\bigwedge_{C \in 2^A} \left(\bigwedge_{X' \in j(C)} reachseq(X', n, C, B) \right)$ then return(true);
9 return(false);

The function *reachseq* is only defined for arguments $n \geq 1$.

```
1  reachseq(X, n, A, B)
2      for every X → Y.Z
3          if reach(Y, n − 1, A, B) then return(true);
4          for every Y ≻ W
5              if W.Z ⊨ (A ∧ −B) then return(true);
6      return(false);
```

Remark: Note that seq-rules of the form $X.Y \to Z$ (sequential composition on the left side) are almost never used in the algorithm. The only exception is in line 5 of the function *reachseq* where they might be needed to enable some action in A. The reason why they are not used anywhere else is because they are not needed since Δ is in transitive normal form.

Lemma 6.10 *The above algorithms are correct and effective implementations of the functions reach and reachseq.*

Proof By induction on n.

Base case: For *reach* the base case is $n = 0$. The correctness follows immediately from the definition of the function *check'* and Lemma 6.7. Note that no seq-rules are used, because Δ is in transitive normal form.

For *reachseq* the base case is $n = 1$. By definition the first rule application must have the form $X \to Y.Z$ (as in line 2). Line 3 deals with the case that Y alone develops into some term $t \in P$ that satisfies $A \wedge -B$ and Z does not play a role. However, t must not be a single constant, because otherwise it might be able to interact with Z via a seq-rule. The function *reach* is called with argument $n = 0$ and just calls the function *check'* which guarantees that t is not a single constant. In line 4,5 we consider the case that Y is rewritten to a single constant W (possibly Y itself) s.t. $W.Z \models (A \wedge -B)$. Since Δ is in transitive normal norm the condition in line 4 is trivial to check: either $W = Y$ or $(Y \to W) \in \Delta$. Line 5 is there to deal with the case that some seq-rule of the form $W.Z \xrightarrow{a} Z'$ is needed to enable some action a in A.

Step: In the function *reach* we split the set of actions A into subsets. The special subset A' are the actions that should become enabled after applying only par-rules to X. The other subsets of actions are assigned

sets of constants by the function j. These constants require further application of seq-rules. By the function *check* we test for the reachability of a state t with

$$(t = t'|||_{(C\in 2^A)} t_c) \ \wedge \ t' \models (A' \wedge -B) \ \wedge \bigwedge_{C\in 2^A} h(j(C), t_c)$$

and the additional constraint that $t \in Const \Rightarrow t' = \epsilon$.

The constants in the terms t_c require further applications of seq-rules. The additional constraint ensures that at least one t_c is not ϵ or t' is not a constant. This ensures that the reachable state that finally satisfies $(A \wedge -B)$ is not a constant. (The applications of seq-rules in *reachseq* always yield non-constant terms.)

Now for the correctness of *reachseq*: By definition the first rule application must have the form $X \to Y.Z$ (as in line 2). Line 3 deals with the case that Y alone develops into some term $t \in P$ that satisfies $A \wedge -B$ and Z does not play a role. However, t must not be a single constant, because otherwise it might be able to interact with Z via a seq-rule. The function *reach* guarantees this and by induction hypothesis the correctness follows. In line 4,5 we consider the case that Y is rewritten to a single constant W (possibly Y itself) s.t. $W.Z \models (A \wedge -B)$. Since Δ is in transitive normal norm the condition in line 4 is trivial to check: either $W = Y$ or $(Y \to W) \in \Delta$. Line 5 is there is deal with the case that some seq-rule of the form $W.Z \xrightarrow{a} Z'$ is needed to enable some action a in A. ■

Now we show that it suffices to consider terms with bounded nesting-depth of sequential composition.

Lemma 6.11 *Let Δ be a PRS in transitive normal form, $X \in Const(\Delta)$ and $A, B \subseteq Act(\Delta)$.*
Then $X \models \Diamond(A \wedge -B)$ iff there is a term t s.t. $X \succ^\Delta t$, $t \models (A \wedge -B)$ and $snd(t) \leq |A| * |Const(\Delta)|$.

Proof X is transformed into t by applying rewrite rules from Δ. The nesting-depth of sequential composition is only increased when a seq-rule of the form $Z \to Z'.Z''$ is applied to some constant Z which is a subterm of an intermediate term. In the end Z should be rewritten to a subterm t'

of of t that satisfies a part of the formula $(A \wedge -B)$. Thus this subterm Z is required to satisfy $\Diamond(A' \wedge -B)$ for some $A' \subseteq A$. Let a chain be a sequence of applications of rewrite rules s.t. every rule rewrites at least part of the term which was introduced by the previous one. Consider a chain and the sequence of constants Z_i in it to which seq-rules are applied and the sequence of formulae $\Diamond(A'_i \wedge -B)$ that the Z_i are required to satisfy. These subsets A'_i can never get bigger in a chain. Furthermore, if they get smaller they must be subsets of previous ones. Therefore in any chain at most $|A|$ different formulae $\Diamond(A'_i \wedge -B)$ must be satisfied by constants Z_i to which seq-rules are applied. We can assume that in any chain no constant (to which a seq-rule is applied) appears twice with the same formula, because this means that a constant has been rewritten to a term containing this constant without making any progress in the formula. It follows that any chain contains at most $|A| * |Const(\Delta)|$ applications of seq-rules, because there are only $|Const(\Delta)|$ different constants. Thus we get $snd(t) \leq |A| * |Const(\Delta)|$. ■

Theorem 6.12 *The reachable property problem is decidable for PRS.*

Proof An instance is given by a PRS Δ, an initial state t_0 and a state-formula Φ. The question is if $t_0 \models \Diamond\Phi$. Without restriction we can assume that t_0 is a single constant X. (Otherwise just add a rule $X \xrightarrow{\tau} t_0$.) By Lemma 6.1 and Lemma 6.2 the problem can be reduced to a problem for PRS in transitive normal form. By Remark 6.3 the problem can be reduced to problems for formulae of the form $\Diamond(A \wedge -B)$. If $X \models \Diamond(A \wedge -B)$ then there are two cases:

1. X can reach a term $Y \in Const$ s.t. $Y \models (A \wedge -B)$. This can be easily checked, because Δ is in transitive normal form. For every constant $Y \in Const$ check if $(X \rightarrow Y) \in \Delta$ and $Y \models (A \wedge -B)$. Also check if $X \models (A \wedge -B)$.

2. X can reach a term $t \notin Const$ s.t. $t \models (A \wedge -B)$. By Lemma 6.11 there is such a t with $snd(t) \leq |A| * |Const(\Delta)|$. Thus the condition can be checked by computing $reach(X, |A| * |Const(\Delta)|, A, B)$. By Lemma 6.10 this can be done with the algorithms given above.

$X \models \Diamond(A \wedge -B)$ iff one of those checks yields a positive answer. ■

Remark 6.13 *This result can also be used to decide deadlock-freedom. Let* Δ *be a PRS with initial state* t_0 *and* $Act(\Delta)$ *the (finite!) set of actions used in* Δ*. A deadlock is reachable iff* $t_0 \models \Diamond(-Act(\Delta))$*. Thus the system is deadlock-free iff* $t_0 \not\models \Diamond(-Act(\Delta))$*.*

7 Conclusion

The algorithms for the reachability problem and the reachable property problem for PRS rely on the reachability problem for Petri nets, which has a high complexity (*EXPSPACE*-hard [May84, Lip76]). So it might seem that they are not applicable in practice because of their very high complexity. However, there are three arguments in their favor:

1. In many examples the system is not very large and the structure of the Petri nets that are contained in them is often simple.

2. In a large PRS there may be many Petri nets as substructures, but often each of these Petri nets is quite small. These Petri nets are either not connected with each other at all, or their influence on each other is very limited. Thus they yield small subproblems that can be solved in acceptable time.

3. Finally, the reachability problem for Petri nets has been studied for many years and ways of dealing with it have been developed. There are semi-decision procedures that give yes/no/don't know answers in acceptable time [CH78, Mur89, ME96]. These algorithms mostly use constraints to represent sets of states and approximate the behavior of the system.

Therefore the algorithms of Section 5 and Section 6 can still be useful in practice to verify systems that are modeled with PRS.

Process Rewrite Systems (PRS) is a very expressive model of infinite-state concurrent systems that subsumes PAN, PAD, Petri nets, PA-processes, pushdown processes, BPP and BPA. PRS extends Petri nets by introducing an operator for sequential composition. This can be seen as the possibility to call subroutines. The calling of subroutines is already possible in PAN-processes. However, there is a major difference: In PAN subroutines that terminate have no effect on their caller, while in PRS subroutines can return

a value to the caller when they terminate. This is an important aspect in modeling real programs. Thus PRS-processes can be used to model systems that exceed the bounds of the expressiveness of Petri nets and PAN.

PRS is a very general model for concurrent systems. Thus model checking with many temporal logics (EF, CTL, LTL, linear time μ-calculus, modal μ-calculus) is undecidable for it. This is because EF is undecidable for Petri nets [Esp97, BE97], CTL is undecidable for BPP [EK95] and LTL and the linear time μ-calculus are undecidable for PA-processes [BH96]. However, PRS is not Turing powerful, since reachability is still decidable.

Finally, it should be noted that PRS are (roughly) equivalent to ground AC rewrite systems (i.e. rewrite systems without substitution, but with an associative and commutative operator). The general idea is that e.g. a ground AC term $Z(X + Y)$ (where '+' is the associative and commutative operator) corresponds to a PRS-term $(X \| Y).Z$ and vice versa.

Acknowledgments: I thank Michaël Rusinowitch and Javier Esparza for helpful discussions and three anonymous referees for their detailed comments.

References

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[BE97] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 5, 1997.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages*. ACM-Press, 1978.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[Dic13] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[Jan90] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Comuter Science*, 74:71–93, 1990.

[Kuc] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[Lip76] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[May84] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460, 1984.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In Martin Abadi and Takayasu Ito, editors, *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*. Springer Verlag, 1997.

[May97b] R. Mayr. Model checking PA-processes. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[May97c] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 7, 1997. Proceedings of Expressiveness in Concurrency (EXPRESS'97).

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.

[ME96] S. Melzer and J. Esparza. Checking system properties via integer programming. In H.R. Nielson, editor, *Proc. of ESOP'96*, volume 1058 of *Lecture Notes in Computer Science*, pages 250–264. Springer Verlag, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mol96] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[Mur89] T. Murata. Petri nets: Properties, analysis und applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

Model Checking Lossy Vector Addition Systems

Ahmed Bouajjani * *** Richard Mayr** ***

Abstract. Lossy VASS (vector addition systems with states) are defined as a subclass of VASS in analogy to lossy FIFO-channel systems. They can be used to model concurrent systems with unreliable communication. We analyze the decidability of model checking problems for lossy systems and several branching-time and linear-time temporal logics. We present an almost complete picture of the decidability of model checking for normal VASS, lossy VASS and lossy VASS with test for zero.

1 Introduction

VASS's (vector addition systems with states) can model communicating systems through unbounded unordered buffers, and hence they can be seen as abstractions of fifo-channels systems, when the ordering between messages in the channels is not relevant but only their number. Communicating systems are often analyzed under the assumption that they communicate through unreliable channels. Hence, we consider *lossy* models of communicating systems, i.e. models where messages can be lost. Recent works are about lossy unbounded fifo-channels systems [AJ93, AJ96, CFI96]. The reachability problem is decidable for these models, which implies the decidability of the verification problem for safety properties. However, liveness properties cannot be checked for lossy fifo-channel systems, unless for very special ones like single eventualities. In particular, it is impossible to model check lossy channel systems under fairness conditions. Here we study verification problems for VASS and VASS with inhibitor arcs (counter machines) under the assumption of lossiness, i.e. the contents of a place/counter can spontaneously get lower at any time.

Using the approach introduced in [CFI96, ACJT96], it can be shown very easily that the set $pre^*(S)$ of predecessors of any set of configurations S is effectively constructible for lossy VASS even with inhibitor arcs, and that this set can be represented by simple linear constraints (SC for short), where integer variables can be compared only with constants. Moreover, for lossy VASS, the set $post^*(S)$ of successors is SC definable and effectively constructible, but interestingly, for lossy VASS with inhibitor arcs these sets are not constructible although they are SC definable.

Local model checking, or simply *model checking*, consists in deciding whether a given configuration of a system satisfies a given formula of a temporal logic, and *global model checking* consists in constructing the set of all configurations that satisfy a given formula. We address these problems for a variety of linear-time and branching-time properties. We express these properties in a temporal logic, called AL (Automata Logic), which is based on automata on finite and infinite sequences to specify path properties (in the spirit of ETL), and the use of path

* Verimag, Centre Equation, 2 avenue de Vignate, 38610 Gières, France.
** Institut für Informatik, TU-München, Arcisstr. 21, D-80290 München, Germany.
*** Ahmed.Bouajjani@imag.fr, mayrri@informatik.tu-muenchen.de

quantifiers to express branching-time properties (like in ECTL* [Tho89]). The basic state predicates in this logic are SC constraints.
Our main positive result is that for lossy VASS, the global model checking is decidable for the logic $\exists$AL with only upward closed constraints, and dually for $\forall$AL with downward closed constraints ($\forall$AL and $\exists$AL are the universal and existential positive fragments of AL. They subsume respectively the corresponding well-known fragments $\forall$CTL* and $\exists$CTL* [GL94] of the logic CTL*). When only infinite paths are considered our decidability result also holds for normal VASS. A corollary is that linear-time properties on finite and infinite paths (on infinite paths only) are decidable for lossy VASS (normal VASS). We can even construct the set of all the configurations satisfying these properties. This generalizes the result in [Esp97] where only model checking is considered. Notice also that $\forall$AL is strictly more expressive than all linear-time temporal logics.
These decidability results break down if we relax any of the restrictions: model checking becomes undecidable if we consider $\forall$AL or $\exists$AL formulae with both downward and upward closed constraints, or if we consider lossy VASS with inhibitor arcs. Also, even if we use only propositional constraints in the logic (i.e., only constraints on control locations) the use of negation must be restricted: model checking is undecidable for CTL and lossy VASS. However, it is decidable for the fragments EF and EG of CTL even for lossy VASS with inhibitor arcs, but surprisingly, global model checking is undecidable for EG and lossy VASS (while it is decidable for EF and lossy VASS with inhibitor arcs). As a side effect we obtain that normal VASS (Petri nets) and lossy VASS with inhibitor arcs (lossy counter machines) are incomparable.
The missing proofs can be found in the full version of the paper.

2 Vector Addition Systems with States

Definition 1. A n-dim VASS $\mathcal{S}$ is a tuple $(\Sigma, \mathcal{X}, Q, \delta)$ where Σ is a set of action labels, $\mathcal{X}$ is a set of variables such that $|\mathcal{X}| = n$, Q is a finite set of control states, δ is a finite set of transitions of the form (q_1, a, Δ, q_2) where $a \in \Sigma$, $\Delta \in \mathbb{Z}^n$.

A *configuration* of $\mathcal{S}$ is a pair $\langle q, \vec{u}\rangle$ where $q \in Q$ and $\vec{u} \in \mathbb{N}^n$. Let $\mathcal{C}(\mathcal{S})$ be the set of configurations of $\mathcal{S}$. Given a configuration $s = \langle q, \vec{u}\rangle$, we let $State(s) = q$ and $Val(s) = \vec{u}$.
We define a *transition relation* $\longrightarrow$ on configurations as follows: $\langle q_1, \vec{u}_1\rangle \xrightarrow{a} \langle q_2, \vec{u}_2\rangle$ iff $\exists \tau = (q_1, a, \Delta, q_2) \in \delta$, $\vec{u}_2 = \vec{u}_1 + \Delta$. Let $post_\tau(\langle q_1, \vec{u}_1\rangle)$ (resp. $pre_\tau(\langle q_2, \vec{u}_2\rangle)$) denote the configuration $\langle q_2, \vec{u}_2\rangle$ (resp. $\langle q_1, \vec{u}_1\rangle$), i.e., the immediate successor (resp. predecessor) of $\langle q_1, \vec{u}_1\rangle$ (resp. $\langle q_2, \vec{u}_2\rangle$) by the transition τ. Then, we let *post* (resp. *pre*) denote the union of the $post_\tau$'s (resp. pre_τ's) for all the transitions $\tau \in \delta$. In other words, $post(\langle q, \vec{u}\rangle) = \{\langle q', \vec{u}'\rangle \ : \ \exists a \in \Sigma.\ \langle q, \vec{u}\rangle \xrightarrow{a} \langle q', \vec{u}'\rangle\}$, and $pre(\langle q, \vec{u}\rangle) = \{\langle q', \vec{u}'\rangle \ : \ \exists a \in \Sigma.\ \langle q', \vec{u}'\rangle \xrightarrow{a} \langle q, \vec{u}\rangle\}$. Let $post^*$ and pre^* be the reflexive-transitive closures of *post* and *pre*.
Given a configuration s, a *run* of the system $\mathcal{S}$ starting from s is a finite or infinite sequence $s_0 a_0 s_1 a_1 \ldots s_n$ such that $s = s_0$ and, for every $i \geq 0$, $s_i \xrightarrow{a_i} s_{i+1}$. We denote by $Run_f(s, \mathcal{S})$ (resp. $Run_\omega(s, \mathcal{S})$) the set of finite (resp. infinite) runs of $\mathcal{S}$ starting from s.

A *lossy VASS* is defined as a VASS with a *weak transition relation* $\Longrightarrow$ on configurations. We define the relation $\Longrightarrow$ as follows: $\langle q_1, \vec{u}_1\rangle \stackrel{a}{\Longrightarrow} \langle q_2, \vec{u}_2\rangle$ iff $\exists \vec{u}'_1, \vec{u}'_2 \in \mathbb{N}^n$, $\vec{u}_1 \geq \vec{u}'_1$, $\langle q_1, \vec{u}'_1\rangle \stackrel{a}{\longrightarrow} \langle q_2, \vec{u}'_2\rangle$, and $\vec{u}'_2 \geq \vec{u}_2$.
The weak transition relation induces corresponding notions of runs, successor and predecessor functions defined by considering the weak transition relation $\Longrightarrow$ instead of $\longrightarrow$.

Definition 2. We order vectors of natural numbers by $(u_1, \ldots, u_n) \leq (v_1, \ldots, v_n)$ iff $\forall i \in \{1, \ldots, n\}.\ u_i \leq v_i$.
Given a set $S \subseteq \mathbb{N}^n$, we denote by $min(S)$ the set of minimal elements of S w.r.t. the relation $\leq$.
Let $S \subseteq \mathbb{N}^n$. Then, S is *upward* (resp. *downward*) *closed* iff $\forall \vec{u} \in \mathbb{N}^n.\ \vec{u} \in S \Rightarrow (\forall \vec{v} \in \mathbb{N}^n.\ \vec{v} \geq \vec{u}$ (resp. $\vec{v} \leq \vec{u}) \Rightarrow \vec{v} \in S)$. Given a set $S \subseteq \mathbb{N}^n$, we denote by $S{\uparrow}$ (resp. $S{\downarrow}$) the upward (resp. downward) closure of S, i.e., the smallest upward (resp. downward) closed set which contains S.

Lemma 3. *Every set $S \subseteq \mathbb{N}^n$ has a finite number of minimal elements. A set is upward closed if and only if $S = min(S){\uparrow}$. The union and the intersection of two upward (resp. downward) closed sets is an upward (resp. downward) closed set. The complement of an upward closed set is downward closed and vice-versa.*

Definition 4 Simple constraints, upward/downward closed constraints. Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables ranging over $\mathbb{N}$.

1. A *simple constraint* over $\mathcal{X}$, SC for short, is any boolean combination of constraints of the form $x \geq c$ where $x \in \mathcal{X}$ and $c \in \mathbb{N} \cup \{\infty\}$.
2. An *upward closed* (resp. *downward closed*) constraint over $\mathcal{X}$, UC (resp. DC) for short, is any positive boolean combination of constraints of the form $x \geq c$ (resp. $x < c$) where $x \in \mathcal{X}$ and $c \in \mathbb{N} \cup \{\infty\}$.

Constraints are interpreted in the standard way as a subset of $\mathbb{N}^n$ ($\leq$ is the usual ordering and $<$ is the strict inequality). Given a simple constraint ξ, we let $[\![\xi]\!]$ denote the set of vectors in $\mathbb{N}^n$ satisfying ξ. Notice that the constraints $x < 0$ and $x \geq \infty$ correspond to $\emptyset$ and that $x \geq 0$ and $x < \infty$ correspond to $\mathbb{N}$.

Definition 5. A set S is SC (resp. UC, DC) definable if there exists an SC (resp. UC, DC) ξ such that $S = [\![\xi]\!]$.

Definition 6 Normal forms.

1. A *canonical product* is a constraint of the form $\vec{\ell} \leq \vec{x} \leq \vec{u}$,
2. A *canonical upward closed product* is a constraint of the form $\vec{\ell} \leq \vec{x}$,
3. A *canonical downward closed product* is a constraint of the form $\vec{x} \leq \vec{u}$,

where $\vec{\ell} \in \mathbb{N}^n$ and $\vec{u} \in (\mathbb{N} \cup \{\infty\})^n$.
A SC (resp. UC, DC) in *normal form* is either $\emptyset$, or a finite disjunction of canonical (resp. canonical upward closed, canonical downward closed) products.

Lemma 7. *Every SC (resp. UC, DC) is equivalent to a SC (UC, DC) in normal form.*

Proposition 8. *SC definable sets are closed under boolean operations, and UC definable sets as well as DC definable sets are closed under union and intersection. The complement of a UC definable set is a DC definable set and vice-versa. A subset of $\mathbb{N}^n$ is UC definable (resp. DC definable) if and only if it is an upward (resp. downward) closed set. A set is SC definable if and only if it is a boolean combination of upward closed sets.*

Let $\mathcal{S} = (\Sigma, \mathcal{X}, Q, \delta)$ be a n-dim VASS with $Q = \{q_1, \ldots, q_m\}$. Then, every set of configurations of $\mathcal{S}$ is defined as a union $C = \{q_1\} \times S_1 \cup \cdots \cup \{q_m\} \times S_m$ where the S_i's are sets of n-dim vectors of natural numbers. The set of configurations C is SC (resp. UC, DC) definable if all the S_i's are SC (resp. UC, DC) definable. We represent SC definable sets by simple constraints in normal form coupled with control states. From now on, we consider a canonical product to be a pair of the form $\langle q, \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle$ where $q \in Q$. A simple constraint is either $\emptyset$ or a finite disjunction of canonical products. We use $\mathrm{SC}(Q, \mathcal{X})$ (resp. $\mathrm{UC}(Q, \mathcal{X})$, $\mathrm{DC}(Q, \mathcal{X})$) to denote the set of simple constraints (resp. upward closed, downward closed constraints). We omit the parameters Q and $\mathcal{X}$ when they are known from the context.

3 Computing Successors and Predecessors

Lemma 9. *The class SC is effectively closed under the operations post and pre for any lossy VASS's.*

Proof. These operations are distributive w.r.t. union. Hence, it suffices to consider separately each transition $\tau = (q, a, \Delta, q')$ and perform them on canonical products:

1. $post_\tau(\langle q, \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle) = \langle q', \vec{x} \leq \vec{u} + \Delta \rangle$.
2. $pre_\tau(\langle q', \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle) = \langle q, (\vec{\ell} - \Delta) \sqcap \vec{0} \leq \vec{x} \rangle$,

where $\forall \vec{u}, \vec{v} \in \mathbb{N}^n$, $\vec{u} \sqcap \vec{v}$ is the vector such that $\forall i \in \{1, \ldots, n\}$. $(\vec{u} \sqcap \vec{v})_i = max(u_i, v_i)$. □

Notice that for lossy VASS's, the *pre* image of any set of configurations is upward closed and its *post* image is downward closed. This also holds for pre^* and $post^*$.

Theorem 10. *For every n-dim lossy VASS $\mathcal{S}$, and every n-dim SC set S, the set $pre^*(S)$ is UC definable and effectively constructible.*

Proof. Since the set $pre^*(S)$ is upward closed, by Proposition 8 we deduce that it is UC definable. The construction of this set is similar to the one given in [CFI96, ACJT96] for lossy channel systems. □

Theorem 11. *For every n-dim lossy VASS $\mathcal{S}$, and every n-dim SC set S, the set $post^*(S)$ is DC definable and effectively constructible.*

Proof. Since $post^*(S)$ is downward closed, by Proposition 8 we deduce that $post^*(S)$ is DC definable. This set can be constructed using the Karp-Miller algorithm for the construction of the coverability graph [KM69]. □

4 Automata and Automata Logic

We use finite automata to express properties of computations. These automata are labeled on states and edges as well. State labels are associated with predicates on the configurations of a given system and edge labels are associated with the actions of the system.

Definition 12. Let Λ and Σ be two finite alphabets. A labeled transition graph over (Λ, Σ) is a tuple $\mathcal{G} = (Q, q_{init}, \Pi, \delta)$ where Q is a finite set of states, q_{init} is the initial state, $\Pi : Q \to \Lambda$ is a state labeling function, $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of labeled transitions. We write $q \xrightarrow{a} q'$ when $(q, a, q') \in \delta$.
Given a state q, a run of $\mathcal{G}$ starting from q is a finite or infinite sequence $q_0 a_0 q_1 a_1 q_2 \ldots$ such that $q_0 = q$ and $\forall i \geq 0.\ q_i \xrightarrow{a_i} q_{i+1}$.

Definition 13 Automata on finite sequences. A finite-state automaton over (Λ, Σ) on finite sequences is a tuple $\mathcal{A}_f = (Q, q_{init}, \Pi, \delta, F)$ where $(Q, q_{init}, \Pi, \delta)$ is a labeled transition graph over (Λ, Σ), and $F \subseteq Q$ is a set of final states. A finite sequence $\lambda_0 a_0 \lambda_1 a_1 \ldots \lambda_n \in \Lambda(\Sigma\Lambda)^*$ is *accepted* by $\mathcal{A}_f$ if there is a run $q_0 a_0 q_1 a_1 \ldots q_n$ of $\mathcal{A}_f$ starting from q_{init} such that $\forall i \in \{0, \ldots, n\}$. $\Pi(q_i) = \lambda_i$, and $q_n \in F$. Let $L(\mathcal{A}_f)$ be the set of sequences in $\Lambda(\Sigma\Lambda)^*$ accepted by $\mathcal{A}_f$.

Definition 14 Büchi ω-automata. A finite-state Büchi automaton over (Λ, Σ) is a tuple $\mathcal{A}_\omega = (Q, q_{init}, \Pi, \delta, F)$ where $(Q, q_{init}, \Pi, \delta)$ is a labeled transition graph over (Λ, Σ), and $F \subseteq Q$ is a set of repeating states. An infinite sequence $\lambda_0 a_0 \lambda_1 a_1 \ldots \lambda_n \in (\Lambda\Sigma)^\omega$ is *accepted* by $\mathcal{A}_\omega$ if there is a run $q_0 a_0 q_1 a_1 \ldots$ of $\mathcal{A}_\omega$ starting from q_{init} such that $\forall i \geq 0.\ \Pi(q_i) = \lambda_i$, and $\overset{\infty}{\exists}\ i \geq 0.\ q_i \in F$. We denote by $L(\mathcal{A}_\omega)$ the set of sequences in $(\Lambda\Sigma)^\omega$ accepted by $\mathcal{A}_\omega$.

Definition 15 Closed ω-automata. A closed ω-automaton is a Büchi automaton $\mathcal{A}_{\omega c} = (Q, q_{init}, \Pi, \delta, F)$ such that $F = Q$.

Remark. [Tho90] Büchi automata define ω-regular sets of infinite sequences. They are closed under boolean operations. Closed ω-automata define closed ω-regular sets in the Cantor topology (the class F in the Borel hierarchy). They correspond to the class of ω-regular safety properties. Closed ω-automata are closed under intersection and union, but not under complementation.

We introduce an automata-based branching-time temporal logic called AL (Automata Logic). This logic is defined in the spirit of the extended temporal logic ETL and is an extension of ECTL* [Tho89]. The logic AL is more expressive than CTL and CTL*, and allows to express all ∞-regular linear-time properties on finite and infinite computations.

Definition 16 Automata Logic. Given a set of control states Q and a set of variables $\mathcal{X}$, we let $\mathcal{F}$ denote a subset of $\mathrm{SC}(Q, \mathcal{X})$, and we let π range over elements of $\mathcal{F}$. Then, the set of $\mathrm{AL}(\mathcal{F})$ formulae is defined by the following grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists\mathcal{A}_f(\varphi_1, \ldots, \varphi_m) \mid \forall\mathcal{A}_f(\varphi_1, \ldots, \varphi_m) \mid \exists\mathcal{A}_\omega(\varphi_1, \ldots, \varphi_m) \mid \forall\mathcal{A}_\omega(\varphi_1, \ldots, \varphi_m)$$

where $\mathcal{A}_f$ (resp. $\mathcal{A}_\omega$) is a finite-state automaton on finite (resp. infinite) sequences over $(\Lambda = \{\lambda_1, \ldots, \lambda_m\}, \Sigma)$. We consider standard abbreviations like $\Rightarrow$.

Definition 17. We use $\star$ to denote f or ω. Let $\mathcal{S} = (\Sigma, \mathcal{X}, Q, \delta)$ be a n-dim (lossy) VASS, We define a satisfaction relation between configurations of $\mathcal{S}$ and AL($\mathcal{F}$) as follows:

$$
\begin{aligned}
s \models (q, \xi) &\text{ iff } State(s) = q \text{ and } Val(s) \in [\![\xi]\!] \\
s \models \neg\varphi &\text{ iff } s \not\models \varphi \\
s \models \varphi_1 \vee \varphi_2 &\text{ iff } s \models \varphi_1 \text{ or } s \models \varphi_2 \\
s \models \varphi_1 \wedge \varphi_2 &\text{ iff } s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models \exists\mathcal{A}_\star(\varphi_1, \ldots, \varphi_m) &\text{ iff } \exists \rho = s_0 a_0 \ldots \in Run_\star(s, \mathcal{S}).\ \exists \sigma = \lambda_{i_0} a_0 \ldots \in L(\mathcal{A}_\star). \\
&\qquad |\sigma| = |\rho| \text{ and } \forall j.\ 0 \le j < |\rho|.\ s_j \models \varphi_{i_j} \\
s \models \forall\mathcal{A}_\star(\varphi_1, \ldots, \varphi_m) &\text{ iff } \forall \rho = s_0 a_0 \ldots \in Run_\star(s, \mathcal{S}).\ \exists \sigma = \lambda_{i_0} a_0 \ldots \in L(\mathcal{A}_\star) \\
&\qquad |\sigma| = |\rho| \text{ and } \forall j.\ 0 \le j < |\rho|.\ s_j \models \varphi_{i_j}
\end{aligned}
$$

For every formula φ, let $[\![\varphi]\!]_{\mathcal{S}} := \{s \in \mathcal{S} \mid s \models \varphi\}$.

Definition 18 Fragments of AL. $\exists$AL($\mathcal{F}$) is the fragment of AL that uses only constraints from $\mathcal{F}$, conjunction, disjunction and existential path quantification. $\forall$AL($\mathcal{F}$) is the fragment of AL that uses only constraints from $\mathcal{F}$, conjunction, disjunction and universal path quantification. Let X be (some fragment of) the logic AL. Then X_f (resp. X_ω, $\mathrm{X}_{\omega c}$) denote the fragment of X where only automata on finite sequences (resp. Büchi, closed ω-automata) are used.

AL is a weaker logic than the modal μ-calculus, but many widely known temporal logics are fragments of AL. Every propositional linear-time property, in particular LTL properties, can be expressed in AL. CTL* is a fragment of AL since every path formula in CTL* corresponds to an LTL formula. Thus, CTL is also a fragment of AL. Clearly, $\forall$AL and $\exists$AL subsume the positive universal and existential fragments of CTL* denoted $\forall$CTL* and $\exists$CTL* (notice that LTL is a fragment of $\forall$CTL*).
We consider two fragments of CTL called EF and EG. The logic EF uses SC predicates, boolean operators, the one-step next operator and the operator EF which is defined by $[\![EF\varphi]\!] = pre^*([\![\varphi]\!])$, The logic EG is defined like EF, except that the operator EF is replaced by the operator EG, which is defined as follows: $s \models EG\varphi$ iff there exists a complete run that starts at s and always satisfies φ. By a complete run we mean either an infinite run or a finite run ending in a deadlock. We use the subscripts f or ω to denote the fragments of these logics obtained by interpreting their formulae on either finite or infinite paths only. Then, it can be seen that $\mathrm{EF} = \mathrm{EF}_f \subseteq \mathrm{CTL}_f \subseteq \mathrm{CTL}^*_f \subseteq \mathrm{AL}_f$. It can also be seen that EG_ω is a fragment of $\mathrm{AL}_{\omega c}$ but EG is not (due to the finite paths).

5 Model Checking

Definition 19 Model checking and global model checking problems.

1. The *model checking problem* is if $s \in [\![\varphi]\!]_{\mathcal{S}}$ for configuration s and formula φ.

2. The *global model checking problem* is whether for any formula φ the set $[\![\varphi]\!]_{\mathcal{S}}$ is effectively constructible.

Lemma 20. *Let $\mathcal{S}$ be a lossy VASS. Then for every formula φ of the form $\exists \mathcal{A}_f(\pi_1, \ldots, \pi_m)$ where all the π_i are SC, the set $[\![\varphi]\!]_{\mathcal{S}}$ is SC definable and effectively constructible.*

Proof. By a generalized pre^* construction (see Theorem 10). □

Theorem 21. *The global model checking problem for lossy VASS and the logic AL_f is decidable.*

Proof. By induction on the nesting-depth and Lemma 20. □

The following results even hold for non-lossy VASS. The aim is to show decidability of the global model checking problem for VASS and the logic $\exists AL_\omega(UC)$. We define a generalized notion of configurations of VASS which includes the symbol ω. This symbol denotes arbitrarily high numbers of tokens on a place. It is used as an abbreviation in the following way: $\langle q, (\omega, \omega, \ldots, \omega, x_{k+1}, \ldots, x_n)\rangle \models \varphi$: $\Longleftrightarrow$ $\exists n_1, \ldots, n_k \in I\!N$. $\langle q, (n_1, n_2, \ldots, n_k, x_{k+1}, \ldots, x_n)\rangle \models \varphi$. (Of course the ω can occur at any position, e.g. $\langle q, (x_1, x_2, \omega, x_4, \omega, x_6)\rangle$.)

Lemma 22. *Let $\mathcal{S}$ be a VASS and φ a formula of the form $\exists \mathcal{A}_\omega(\pi_1, \ldots, \pi_m)$ where all the π_i are in UC. Let s be a generalized configuration of $\mathcal{S}$ (i.e. it can contain ω). It is decidable if $s \models \varphi$.*

Proof. (Sketch) First construct the Karp-Miller coverability graph [KM69]. Then check for the existence of cycles in this graph that have an overall positive effect of the fired transitions. These cycles may contain the same node several times. This check is done with the help of Parikh's Theorem. The property holds iff such a cycle with overall positive effect exists, because it can be repeated infinitely often. □

Lemma 23. *Let $\mathcal{S}$ be a VASS and φ a formula of the form $\exists \mathcal{A}_\omega(\pi_1, \ldots, \pi_m)$ where all the π_i are in UC. The set $[\![\varphi]\!]_{\mathcal{S}}$ is UC definable and effectively constructible.*

Proof. $[\![\varphi]\!]_{\mathcal{S}}$ is upward closed, because all π_i are upward closed. Thus, it is characterized by the finite set of its minimal elements (see Lemma 3). To find the minimal elements, we use a construction that was described by Valk and Jantzen in [VJ85]. The important point here is that we can use Lemma 22 to check the existence of configurations that satisfy φ. For example, if $\langle q, (\omega, x_2, x_3)\rangle \models \varphi$ then we can check if $\langle q, (n_1, x_2, x_3)\rangle \models \varphi$ for $n_1 = 0$, $n_1 = 1$, $n_1 = 2$, ... until we find the minimal n_1 s.t. $\langle q, (n_1, x_2, x_3)\rangle \models \varphi$. □

Theorem 24. *The global model checking problem is decidable for VASS and the logic $\exists AL_\omega(UC)$.*

Proof. By induction on the nesting-depth of the formula and Lemma 23. □

Theorem 25. *The global model checking problem is decidable for lossy VASS and the logic $\exists AL(UC)$.*

Proof. By induction on the nesting depth and Theorems 21 and 24. □

Theorem 26. *The model checking problem for lossy VASS and $AL_{\omega c}$ is decidable.*

Proof. By induction on the nesting-depth of the formula and an analysis of all computations which is finite by Dickson's Lemma. □

Theorem 27. *Model checking lossy VASS with the logic EG is decidable.*

Theorems 26 and 27 say that the model checking problem is decidable for a lossy VASS and an EG-formula/ $AL_{\omega c}$-formula φ. However, in both cases the set $[\![\varphi]\!]_{\mathcal{S}}$ is not effectively constructible (although it is SC definable). If it were constructible then Lemma 20 could be used to decide model checking lossy VASS with formulae of the form $EFEG_{\omega}\pi$, where π is a constraint in SC. However, this problem has very recently been shown to be undecidable.

Proposition 28. *Model checking lossy VASS with formulae of the form $EFEG_{\omega}\pi$, where π is a constraint in SC is undecidable.*

Proof. This is a corollary of a more general undecidability result for lossy BPP (Basic Parallel Processes), which follows (not immediately) from the result on lossy counter machines in Proposition 30 (see [May98]). □

Remark. This undecidability result also implies undecidability of model checking lossy VASS with the logic $\exists AL_{\omega}$. One can encode properties of the form $EFEG_{\omega}\pi$ in $\exists AL_{\omega}$ in the following way: Let $\mathcal{A}_{\omega}$ be an automaton with states q, q', and transitions $q \to q$, $q \to q'$ and $q' \to q'$ which are labeled with any action. The predicate *true* is assigned to q and the predicate π is assigned to q'. q is the initial state and q' is the only repeating state. Let $\mathcal{A}'_{\omega}$ be an automaton with only one state q which is the initial state and repeating and a transition $q \to q$ with any action. The predicate π is assigned to q. Then for any lossy VASS s we have $s \models EFEG_{\omega}\pi \iff s \models \mathcal{A}_{\omega}(true, \pi) \vee \mathcal{A}'_{\omega}(\pi)$.

Lossy VASS can be extended with inhibitor arcs. This means introducing transitions that can only fire if some defined places are empty (i.e. they can test for zero). Thus lossy VASS with inhibitor arcs are equivalent to lossy counter machines. Normal VASS with inhibitor arcs are Turing-powerful, but lossy VASS with inhibitor arcs are not.

Theorem 29. *For lossy VASS with inhibitor arcs*

1. *the global model checking problem is decidable for the logic AL_f.*
2. *model checking is decidable for the logics $AL_{\omega c}$ and EG.*

Inhibitor arcs can never keep a transition from firing, because one can just loose the tokens on the places that inhibit it. However, after such a transition has fired, the number of tokens on the inhibiting places is fixed and known exactly. Such a guarantee is impossible to achieve in lossy VASS without inhibitor arcs. Thus not all results for lossy VASS carry over to lossy VASS with inhibitor arcs.

Proposition 30. *Let $\mathcal{S}$ be a lossy VASS with inhibitor arcs. It is undecidable if there exists an initial configuration s s.t. there is an infinite run of $(s, \mathcal{S})$.*

Proof. This is a corollary of a more general undecidability result for lossy counter machines in [May98]. The main idea is that one can enforce that lossiness occurs only finitely often in the infinite run. □

Theorem 31. *Model checking lossy VASS with inhibitor arcs with the logic LTL is undecidable.*

Proof. We reduce the problem of Proposition 30 to the model checking problem. We construct a lossy VASS with inhibitor arcs $\mathcal{S}'$ that does the following: First it guesses an arbitrary configuration s of $\mathcal{S}$ doing only the atomic action a. Then it simulates $\mathcal{S}$ on s doing only the atomic action b. Let $\mathcal{A}_\omega$ be a Büchi-automaton with initial state q and repeating state q' and transitions $q \xrightarrow{a} q$, $q \xrightarrow{b} q'$ and $q' \xrightarrow{b} q'$. Let s' be the initial state of $\mathcal{S}'$. We have reduced the question of Proposition 30 to the question if $(s', \mathcal{S}') \models \exists \mathcal{A}_\omega(true, true)$. This question can be expressed in LTL. □

It follows immediately that model checking lossy VASS with inhibitor arcs with $AL_\omega(UC)$ is undecidable. It is interesting to compare this result with Proposition 28. For undecidability it suffices to have either inhibitor arcs in the system or downward closed constraints in the logic. One can be encoded in the other and vice versa. The set $post^*(s)$ is DC definable since it is downward closed. However, it is not constructible for lossy VASS with inhibitor arcs (unlike for lossy VASS, see Theorem 11).

Theorem 32. *$post^*(s)$ is not constructible for lossy VASS with inhibitor arcs.*

Proof. Boundedness is undecidable for reset Petri nets [DFS98]. This result carries over to lossy reset Petri nets. Lossy VASS with inhibitor arcs can simulate lossy reset Petri nets. It follows that boundedness is undecidable for lossy VASS with inhibitor arcs and thus $post^*(s)$ is not constructible. □

6 Conclusion

We have established results for normal VASS and lossy VASS with inhibitor arcs (lossy counter machines). Interestingly, it turns out that these two models are incomparable. Moreover, all the positive/negative results we obtained for lossy VASS with inhibitor arcs are the same as for lossy fifo-channel systems. Note that lossy fifo-channel systems can simulate lossy VASS with inhibitor arcs, but only with some additional deadlocks.

The following table summarizes the results on the decidability of model checking for VASS, lossy VASS with test for zero, lossy VASS and lossy fifo-channel systems. By '++' we denote the fact that for any formula φ the set $[\![\varphi]\!]$ is SC definable and effectively constructible (global model checking), while '+' means that only model checking is decidable. We denote by — that model checking is undecidable. The symbol '?' denotes an open problem.

Logic	VASS	Lossy VASS+0	Lossy VASS	Lossy FIFO
AL_f/EF	— [Esp97]	++	++ [AJ93]	++ [AJ93]
$\exists AL_\omega(UC)/LTL$	++ /+[Esp97]	—	++	— [AJ96]
$\exists AL(UC)$	?	—	++	— [AJ96]
$AL_{\omega c}$/EG	— [EK95]	+	+ [AJ93]	+ [AJ93]
$\exists AL_\omega$/CTL	— [EK95]	—	—	— [AJ96]

The results in this table are new, except where references are given. For normal VASS and LTL, decidability of the model checking problem was known [Esp97], but the construction of the set $[\![\varphi]\!]$ is new. The results in [AJ93] are just about EF and EG formulae without nesting, not for the full logics AL_f and $AL_{\omega c}$.

Acknowledgment: We thank Peter Habermehl for interesting discussions.

References

[ACJT96] P. Abdulla, K. Cerans, B. Jonsson, and Y-K. Tsay. General Decidability Theorems for Infinite-state Systems. In *LICS'96*. IEEE, 1996.

[AJ93] P. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS'93*. IEEE, 1993.

[AJ96] P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[GL94] O. Grumberg and D. Long. Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems*, 16, 1994.

[KM69] R. Karp and R. Miller. Parallel program schemata. *JCSS*, 3, 1969.

[May98] R. Mayr. Lossy counter machines. Technical Report TUM-I9827, TU-München, October 1998. wwwbrauer.informatik.tu-muenchen.de/~mayrri.

[Tho89] W. Thomas. Computation Tree Logic and Regular ω-Languages. LNCS 354, 1989.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theo. Comp. Sci.* Elsevier Sci. Pub., 1990.

[VJ85] R. Valk and M. Jantzen. The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets. *Acta Informatica*, 21, 1985.

Appendix: Full version.

Model Checking Lossy Vector Addition Systems

Ahmed Bouajjani [*‡] Richard Mayr[†‡]

Abstract

Lossy VASS (vector addition systems with states) are defined as a subclass of VASS in analogy to lossy FIFO-channel systems. They can be used to model concurrent systems with unreliable communication. We analyze the decidability of model checking problems for lossy systems and several branching-time and linear-time temporal logics. We present an almost complete picture of the decidability of model checking for normal VASS, lossy VASS and lossy VASS with test for zero.

1 Introduction

Systems are usually modeled by finite control transition systems with different kinds of variables and data structures like counters, clocks, stacks, fifo-channels, etc. One of the widely used models of concurrent systems is the model of Petri nets which is equivalent to the model of vector addition systems with states (VASS for short). These models can be considered as particular cases of counter machines where tests to zero are forbidden (the addition of inhibitor arcs gives them the full power of counter machines). VASS's can model communicating systems through unbounded unordered buffers, and hence they can be seen as abstractions of fifo-channels systems, when the ordering between messages in the channels is not relevant but only their number. Actually, it is often the case that communicating systems must be analyzed under the assumption that they communicate through unreliable channels. Hence, it is natural to consider *lossy* models of communicating

[*]Verimag, Centre Equation, 2 avenue de Vignate, 38610 Gières, France.
[†]Institut für Informatik, TU-München, Arcisstr. 21, D-80290 München, Germany.
[‡]Ahmed.Bouajjani@imag.fr, mayrri@informatik.tu-muenchen.de

systems, i.e. models where messages can be lost. Several works have been devoted recently to the analysis of lossy unbounded fifo-channels systems [AJ93, AJ96, CFI96]. They have shown in particular that the reachability problem is decidable for these models, which implies the decidability of the verification problem for safety properties. They have also shown that, unfortunately, liveness properties cannot be checked for lossy fifo-channel systems, unless for very special ones like single eventualities. In particular, it is impossible to model check lossy channel systems under fairness conditions. In this paper, we study verification problems for VASS and VASS with inhibitor arcs (counter machines) under the assumption of lossiness. Here lossiness means that the contents of a place/counter can spontaneously get lower at any time.

Since verification is essentially based on reachability analysis, the first question we address is whether given a set S of configurations, it is possible to effectively construct the set $pre^*(S)$ of all its predecessors and the set $post^*(S)$ of all its successors. Using the approach introduced in [CFI96, ACJT96], it can be shown very easily that the pre^* image of any set of configurations is effectively constructible for lossy VASS even with inhibitor arcs, and that this set can be represented by simple linear constraints (SC for short), where integer variables can be compared only with constants. Moreover, we show that for lossy VASS, the $post^*$ images are SC definable and effectively constructible, but interestingly, for lossy VASS with inhibitor arcs these sets are not constructible although they are SC definable. This means that safety properties (at least) can be checked for VASS with both backward and forward reachability analysis, while they can only be checked using backward search for lossy VASS with inhibitor arcs (efficient but incomplete forward analysis procedures could be applied following the approach presented in [BGWW97, BH97, ABJ98]).

Then, the major part of the paper concerns model checking. We consider two versions of this problem: *local model checking*, or simply *model checking*, which consists in deciding whether a given configuration of a system satisfies a given formula of a temporal logic, and *global model checking* which consists in constructing the set of all configurations that satisfy a given formula. We address these problems for a variety of linear-time and branching-time properties. For the sake of generality, we express these properties in a temporal logic, called AL (Automata Logic), which is based on the use of automata on finite and infinite sequences to specify path properties (in the spirit of ETL [Wol83]), and the use of path quantifiers to express branching-time properties (like in ECTL* [CGK87, Tho89]). The basic state predicates in this logic are SC constraints.

Our main positive result is that for lossy VASS, the global model checking is

decidable for the logic $\exists$AL with only upward closed constraints, and dually for $\forall$AL with downward closed constraints ($\forall$AL and $\exists$AL are the universal and existential positive fragments of AL. They subsume respectively the corresponding well-known fragments $\forall$CTL* and $\exists$CTL* [GL94] of the logic CTL*). Actually, we show that when only infinite paths are considered our decidability result also holds for normal VASS. A corollary of these results is that, since in $\forall$AL we use automata to define path properties, all ∞-regular (ω-regular) linear-time properties on finite and infinite paths (on infinite paths only) are decidable for lossy VASS (normal VASS), and even more, we can construct the set of all the configurations satisfying these properties. This generalizes the result concerning VASS and ω-regular properties given in [Esp94] where only model checking is considered. Notice also that $\forall$AL is strictly more expressive than all linear-time temporal logics.

These results are interesting since systems are often verified under abstractions, and it can be shown that, if a system $\mathcal{S}$ simulates a system $\mathcal{C}$, then if a $\forall$AL formula holds on the "abstract" system $\mathcal{S}$, it also holds on the "concrete" system $\mathcal{C}$. For instance, if we start with a (lossy) fifo-channel system, we can abstract it to a VASS or a lossy VASS, and then check the property we are interested in on the abstract model, and if it holds, we can deduce that it holds on the original model.

Then we show that these decidability results break down if we relax any of the restrictions we have on the system models and the logic: model checking becomes undecidable if we consider $\forall$AL or $\exists$AL formulas with both downward and upward closed constraints, or if we consider lossy VASS with inhibitor arcs. Also, even if we use only propositional constraints in the logic (i.e., only constraints on control locations) the use of negation must be restricted: model checking is undecidable for CTL and lossy VASS. However, it is decidable for the fragments EF and EG of CTL even for lossy VASS with inhibitor arcs, but surprisingly, global model checking is undecidable for EG and lossy VASS (while it is decidable for EF and lossy VASS with inhibitor arcs). We also obtain as a side effect of our results that normal VASS (Petri nets) and lossy VASS with inhibitor arcs (lossy counter machines) are two incomparable models.

The rest of the paper is organized as follows: In the next section we recall the definition of VASS. In Section 3 we introduce simple constraints and show how they can be used to represent sets of VASS configurations. In Section 4, we show how the backward and forward reachability sets of lossy VASS can be effectively computed by means of simple constraints. In Section 5 we recall the definitions of automata on finite and infinite sequences since they are used in the definition of our logic AL. In Section 5 we introduce

the logic AL and its fragments. In Section 6 we give our results concerning model checking of lossy VASS and different fragments of AL. In Section 7, we examine how these results extend or break down in the case of lossy VASS with inhibitor arcs. Finally, we give concluding remarks and a table summarizing our results in Section 8.

2 Vector Addition Systems with States

Definition 2.1 *A n-dim VASS $\mathcal{S}$ is a tuple $(\Sigma, \mathcal{X}, Q, \delta)$ where*

- *Σ is a set of action labels,*
- *$\mathcal{X}$ is a set of variables such that $|\mathcal{X}| = n$,*
- *Q is a finite set of control states,*
- *δ is a finite set of transitions of the form (q_1, a, Δ, q_2) where $a \in \Sigma$, $\Delta \in \mathbb{Z}^n$.*

A *configuration* of $\mathcal{S}$ is a pair $\langle q, \vec{u} \rangle$ where $q \in Q$ and $\vec{u} \in \mathbb{N}^n$. Let $\mathcal{C}(\mathcal{S})$ be the set of configurations of $\mathcal{S}$. Given a configuration $s = \langle q, \vec{u} \rangle$, we let $State(s) = q$ and $Val(s) = \vec{u}$.

We define a *transition relation* $\longrightarrow$ on configurations as follows: $\langle q_1, \vec{u}_1 \rangle \xrightarrow{a} \langle q_2, \vec{u}_2 \rangle$ iff $\exists \tau = (q_1, a, \Delta, q_2) \in \delta$, $\vec{u}_2 = \vec{u}_1 + \Delta$. We let $post_\tau(\langle q_1, \vec{u}_1 \rangle)$ (resp. $pre_\tau(\langle q_2, \vec{u}_2 \rangle)$) denote the configuration $\langle q_2, \vec{u}_2 \rangle$ (resp. $\langle q_1, \vec{u}_1 \rangle$), i.e., the immediate successor (resp. predecessor) of $\langle q_1, \vec{u}_1 \rangle$ (resp. $\langle q_2, \vec{u}_2 \rangle$) by the transition τ. Then, we let *post* (resp. *pre*) denote the union of the $post_\tau$'s (resp. pre_τ's) for all the transitions $\tau \in \delta$. In other words, $post(\langle q, \vec{u} \rangle) = \{\langle q', \vec{u}' \rangle : \exists a \in \Sigma.\ \langle q, \vec{u} \rangle \xrightarrow{a} \langle q', \vec{u}' \rangle\}$, and $pre(\langle q, \vec{u} \rangle) = \{\langle q', \vec{u}' \rangle : \exists a \in \Sigma.\ \langle q', \vec{u}' \rangle \xrightarrow{a} \langle q, \vec{u} \rangle\}$. Let $post^*$ and pre^* be the reflexive-transitive closures of *post* and *pre*.

Given a configuration s, a *run* of the system $\mathcal{S}$ starting from s is a finite or infinite sequence $s_0 a_0 s_1 a_1 \ldots s_n$ such that $s = s_0$ and, for every $i \geq 0$, $s_i \xrightarrow{a_i} s_{i+1}$. We denote by $Run_f(s, \mathcal{S})$ (resp. $Run_\omega(s, \mathcal{S})$) the set of finite (resp. infinite) runs of $\mathcal{S}$ starting from s.

A *lossy VASS* is defined as a VASS with a *weak transition relation* $\Longrightarrow$ on configurations. We define the relation $\Longrightarrow$ as follows: $\langle q_1, \vec{u}_1 \rangle \stackrel{a}{\Longrightarrow} \langle q_2, \vec{u}_2 \rangle$ iff $\exists \vec{u}'_1, \vec{u}'_2 \in \mathbb{N}^n$, $\vec{u}_1 \geq \vec{u}'_1$, $\langle q_1, \vec{u}'_1 \rangle \xrightarrow{a} \langle q_2, \vec{u}'_2 \rangle$, and $\vec{u}'_2 \geq \vec{u}_2$.

The definition of the weak transition relation induces corresponding notions of runs, successor and predecessor functions defined exactly as in the case of perfect VASS's by considering the weak transition relation $\Longrightarrow$ instead of the relation $\longrightarrow$.

3 Representing Sets of Configurations

Definition 3.1 *Let $\leq$ denote the usual ordering on natural numbers. We extend this relation to vectors of natural numbers in the standard way. Let $\vec{u} = (u_1, \ldots, u_n)$ and $\vec{v} = (v_1, \ldots, v_n)$ be two vectors in $(\mathbb{N} \cup \infty)^n$. Then, we have $\vec{u} \leq \vec{v}$ iff $\forall i \in \{1, \ldots, n\}$. $u_i \leq v_i$.*
Given a set $S \subseteq \mathbb{N}^n$, we denote by $min(S)$ the set of minimal elements of S w.r.t. the relation $\leq$.
Let $S \subseteq \mathbb{N}^n$. Then, S is upward *(resp.* downward*)* closed *iff $\forall \vec{u} \in \mathbb{N}^n$. $\vec{u} \in S \Rightarrow (\forall \vec{v} \in \mathbb{N}^n$. $\vec{v} \geq \vec{u}$ (resp. $\vec{v} \leq \vec{u}$) $\Rightarrow \vec{v} \in S)$. Given a set $S \subseteq \mathbb{N}^n$, we denote by $S{\uparrow}$ (resp. $S{\downarrow}$) the upward (resp. downward) closure of S, i.e., the smallest upward (resp. downward) closed set which contains S.*

Lemma 3.2 *Every set $S \subseteq \mathbb{N}^n$ has a finite number of minimal elements.*

Proof Let $S \subseteq \mathbb{N}^n$ and suppose that S has an infinite set of minimals $min(S) = \{\vec{u}_1, \vec{u}_2, \ldots\}$. Then, by Dickson's lemma, there are two indices i and j such that $\vec{u}_i \leq \vec{u}_j$, which contradicts the fact that $\vec{u}_i$ and $\vec{u}_j$ are both minimal elements. ■

Lemma 3.3 *A set is upward closed if and only if $S = min(S){\uparrow}$.*

Lemma 3.4 *The union and the intersection of two upward (resp. downward) closed sets is an upward (resp. downward) closed set. The complement of an upward closed set is downward closed and vice-versa.*

Proof The cases of union and intersection are trivial. Let S be a downward closed set, and let $\overline{S} = \mathbb{N}^n \setminus S$. Then, let us consider two vectors $\vec{u} \in \overline{S}$ and $\vec{v} \in \mathbb{N}^n$ such that $\vec{v} \geq \vec{u}$. We have necessarily $\vec{v} \in \overline{S}$, because otherwise (i.e., if $\vec{v} \in S$) $\vec{u}$ must be also in S since it is downward closed, which contradicts the fact that $\vec{u} \in \overline{S}$. A symmetrical argument allows to show that the complement of an upward closed set is downward closed.

Definition 3.5 (Constraints) *Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables ranging over $\mathbb{N}$.*

1. *A* simple constraint *over $\mathcal{X}$, SC for short, is any boolean combination of constraints of the form $x \geq c$ where $x \in \mathcal{X}$ and $c \in \mathbb{N} \cup \{\infty\}$.*
2. *An* upward closed *(resp.* downward closed*) constraint over $\mathcal{X}$, UC (resp. DC) for short, is any positive boolean combination of constraints of the form $x \geq c$ (resp. $x < c$) where $x \in \mathcal{X}$ and $c \in \mathbb{N} \cup \{\infty\}$.*

Constraints are interpreted in the standard way as a subset of $\mathbb{N}^n$ ($\leq$ is the usual ordering and $<$ is the strict inequality). Given a simple constraint ξ, we let $[\![\xi]\!]$ denote the set of vectors in $\mathbb{N}^n$ satisfying ξ. Notice that the constraints $x < 0$ and $x \geq \infty$ correspond to $\emptyset$ and that $x \geq 0$ and $x < \infty$ correspond to $\mathbb{N}$.

Definition 3.6 *A set S is SC (resp. UC, DC) definable if there exists an SC (resp. UC, DC) ξ such that $S = [\![\xi]\!]$.*

Definition 3.7 (Normal forms)

1. *A* canonical product *is a constraint of the form $\vec{\ell} \leq \vec{x} \leq \vec{u}$,*
2. *A* canonical upward closed product *is a constraint of the form $\vec{\ell} \leq \vec{x}$,*
3. *A* canonical downward closed product *is a constraint of the form $\vec{x} \leq \vec{u}$,*

where $\vec{\ell} \in \mathbb{N}^n$ and $\vec{u} \in (\mathbb{N} \cup \{\infty\})^n$.
A SC (resp. UC, DC) in normal form *is either $\emptyset$, or a finite disjunction of canonical (resp. canonical upward closed, canonical downward closed) products.*

Lemma 3.8 *Every SC (resp. UC, DC) is equivalent to a SC (resp. UC, DC) in normal form.*

Lemma 3.9 (Expressiveness) *SC definable sets are closed under boolean operations, and UC definable sets as well as DC definable sets are closed under union and intersection. The complement of a UC definable set is a DC definable set and vice-versa. A subset of $\mathbb{N}^n$ is UC definable (resp. DC definable) if and only if it is an upward (resp. downward) closed set. A set is SC definable if and only if it is a boolean combination of upward closed sets.*

Proof Follows from Lemmas 3.2, 3.3, and 3.4. ■

Let $\mathcal{S} = (\Sigma, \mathcal{X}, Q, \delta)$ be a n-dim VASS with $Q = \{q_1, \ldots, q_m\}$. Then, every set of configurations of $\mathcal{S}$ is defined as a union $C = \{q_1\} \times S_1 \cup \cdots \cup \{q_n\} \times S_m$ where the S_i's are sets of n-dim vectors of natural numbers. The set of configurations C is SC (resp. UC, DC) definable if all the S_i's are SC (resp. UC, DC) definable.

We represent SC definable sets by simple constraints in normal form coupled with control states. From now on, we consider a canonical product to be a pair of the form $\langle q, \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle$ where $q \in Q$. A simple constraint is either $\emptyset$ or a finite disjunction of canonical products. We consider the same convention in the cases of upward closed and downward closed constraints. We use $\mathrm{SC}(Q, \mathcal{X})$ (resp. $\mathrm{UC}(Q, \mathcal{X})$, $\mathrm{DC}(Q, \mathcal{X})$) to denote the set of simple constraints (resp. upward closed, downward closed constraints). We omit the parameters Q and $\mathcal{X}$ when they are known from the context.

4 Computing Successors and Predecessors

Proposition 4.1 *The class SC is effectively closed under the under the operations post and pre for any lossy VASS's.*

Proof First, notice that these operations are distributive w.r.t. union (disjunction). Hence, it suffices to consider separately each transition $\tau = (q, a, \Delta, q')$ and show how to perform them on canonical products:

1. $post_\tau(\langle q, \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle) = \langle q', \vec{x} \leq \vec{u} + \Delta \rangle$.
2. $pre_\tau(\langle q', \vec{\ell} \leq \vec{x} \leq \vec{u} \rangle) = \langle q, (\vec{\ell} - \Delta) \sqcap \vec{0} \leq \vec{x} \rangle$,

where $\forall \vec{u}, \vec{v} \in \mathbb{N}^n$, $\vec{u} \sqcap \vec{v}$ is the vector such that $\forall i \in \{1, \ldots, n\}$. $(\vec{u} \sqcap \vec{v})_i = max(u_i, v_i)$. ■

Notice that for lossy VASS's, the *pre* image of any set of configurations is upward closed and its *post* image is downward closed. This also holds for pre^* and $post^*$.

Theorem 4.2 *For every n-dim lossy VASS $\mathcal{S}$, and every n-dim SC set S, the set $pre^*(S)$ is UC definable and effectively constructible.*

Proof Since the set $pre^*(S)$ is upward closed, by Proposition 3.9 we deduce that it is UC definable. The construction of this set is similar to the one given in [CFI96, ACJT96] for lossy channel systems. ■

Theorem 4.3 *For every n-dim lossy VASS $\mathcal{S}$, and every n-dim SC set S, the set $post^*(S)$ is DC definable and effectively constructible.*

Proof Since $post^*(S)$ is downward closed, by Proposition 3.9 we deduce that $post^*(S)$ is DC definable. This set can be constructed using the Karp-Miller algorithm for the construction of the coverability graph. ■

5 Automata and Automata Logic

In this section we define the finite automata we use to express properties of computations. These automata are rather standard, except that they are labeled on states and edges as well. This is natural in the context of specification, since state labels are associated with predicates on the configurations of a given system and edge labels are associated with the actions of the system.

Definition 5.1 *Let Λ and Σ be two finite alphabets. A labeled transition graph over (Λ, Σ) is a tuple $\mathcal{G} = (Q, q_{init}, \Pi, \delta)$ where*

- *Q is a finite set of states,*
- *q_{init} is the initial state,*
- *$\Pi : Q \to \Lambda$ is a state labeling function,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a finite set of labeled transitions.*

We write $q \xrightarrow{a} q'$ when $(q, a, q') \in \delta$.
Given a state q, a run of $\mathcal{G}$ starting from q is a finite or infinite sequence $q_0 a_0 q_1 a_1 q_2 \ldots$ such that $q_0 = q$ and $\forall i \geq 0.\ q_i \xrightarrow{a_i} q_{i+1}$.

Definition 5.2 (Automata on finite sequences) *A finite-state automaton over (Λ, Σ) on finite sequences is a tuple $\mathcal{A}_f = (Q, q_{init}, \Pi, \delta, F)$ where $(Q, q_{init}, \Pi, \delta)$ is a labeled transition graph over (Λ, Σ), and $F \subseteq Q$ is a set of final states. A finite sequence $\lambda_0 a_0 \lambda_1 a_1 \ldots \lambda_n \in \Lambda(\Sigma\Lambda)^*$ is* accepted *by $\mathcal{A}_f$ if there is a run $q_0 a_0 q_1 a_1 \ldots q_n$ of $\mathcal{A}_f$ starting from q_{init} such that $\forall i \in \{0, \ldots, n\}.\ \Pi(q_i) = \lambda_i$, and $q_n \in F$. We denote by $L(\mathcal{A}_f)$ the set of sequences in $\Lambda(\Sigma\Lambda)^*$ accepted by $\mathcal{A}_f$.*

Definition 5.3 (Büchi ω-automata) *A finite-state Büchi automaton over (Λ, Σ) is a tuple $\mathcal{A}_\omega = (Q, q_{init}, \Pi, \delta, F)$ where $(Q, q_{init}, \Pi, \delta)$ is a labeled transition graph over (Λ, Σ), and $F \subseteq Q$ is a set of repeating states. An infinite sequence $\lambda_0 a_0 \lambda_1 a_1 \ldots \lambda_n \in (\Lambda\Sigma)^\omega$ is* accepted *by $\mathcal{A}_\omega$ if there is a run $q_0 a_0 q_1 a_1 \ldots$ of $\mathcal{A}_\omega$ starting from q_{init} such that $\forall i \geq 0.\ \Pi(q_i) = \lambda_i$, and $\overset{\infty}{\exists}\, i \geq 0.\ q_i \in F$. We denote by $L(\mathcal{A}_\omega)$ the set of sequences in $(\Lambda\Sigma)^\omega$ accepted by $\mathcal{A}_\omega$.*

Definition 5.4 (Closed ω-automata) *A closed ω-automaton is a Büchi automaton $\mathcal{A}_{\omega c} = (Q, q_{init}, \Pi, \delta, F)$ such that $F = Q$.*

Remark 5.5 *[Tho90] Büchi automata define ω-regular sets of infinite sequences. They are closed under boolean operations. Closed ω-automata define closed ω-regular sets in the Cantor topology (the class F in the Borel hierarchy). They correspond to the class of ω-regular safety properties. Closed ω-automata are closed under intersection and union, but not under complementation.*

We introduce an automata-based branching-time temporal logic called AL (Automata Logic). This logic is defined in the spirit of the extended temporal logic ETL [Wol83] and is an extension of ECTL* [CGK87, Tho89] which allows to express temporal properties of (lossy) VASS's involving simple constraints. The logic AL is more expressive than the branching-time logics CTL and CTL*, and allows to express all ∞-regular linear-time properties on finite and infinite computations.

Definition 5.6 (Automata Logic) *Given a set of control states Q and a set of variables $\mathcal{X}$, we let $\mathcal{F}$ denote a subset of $SC(Q,\mathcal{X})$, and we let π range over elements of $\mathcal{F}$. Then, the set of AL($\mathcal{F}$) formulae is defined by the following grammar:*

$$\begin{aligned}\varphi ::= \;& \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists\mathcal{A}_f(\varphi_1,\ldots,\varphi_m) \mid \forall\mathcal{A}_f(\varphi_1,\ldots,\varphi_m) \mid \\ & \exists\mathcal{A}_\omega(\varphi_1,\ldots,\varphi_m) \mid \forall\mathcal{A}_\omega(\varphi_1,\ldots,\varphi_m)\end{aligned}$$

where $\mathcal{A}_f$ (resp. $\mathcal{A}_\omega$) is a finite-state automaton on finite (resp. infinite) sequences over $(\Lambda = \{\lambda_1,\ldots,\lambda_m\},\Sigma)$. We consider standard abbreviations like $\Rightarrow$.

Definition 5.7 *We use $\star$ to denote f or ω. Let $\mathcal{S} = (\Sigma,\mathcal{X},Q,\delta)$ be a n-dim (lossy) VASS, We define a satisfaction relation between configurations of $\mathcal{S}$ and AL($\mathcal{F}$) as follows:*

$$\begin{aligned}
s \models (q,\xi) \quad & \textit{iff} \quad State(s) = q \;\; \textit{and} \;\; Val(s) \in [\![\xi]\!] \\
s \models \neg\varphi \quad & \textit{iff} \quad s \not\models \varphi \\
s \models \varphi_1 \vee \varphi_2 \quad & \textit{iff} \quad s \models \varphi_1 \textit{ or } s \models \varphi_2 \\
s \models \varphi_1 \wedge \varphi_2 \quad & \textit{iff} \quad s \models \varphi_1 \textit{ and } s \models \varphi_2 \\
s \models \exists\mathcal{A}_\star(\varphi_1,\ldots,\varphi_m) \quad & \textit{iff} \quad \exists\rho = s_0a_0\ldots \in Run_\star(s,\mathcal{S}).\ \exists\sigma = \lambda_{i_0}a_0\ldots \in L(\mathcal{A}_\star). \\
& \qquad |\sigma| = |\rho| \textit{ and } \forall j.\ 0 \le j < |\rho|.\ s_j \models \varphi_{i_j} \\
s \models \forall\mathcal{A}_\star(\varphi_1,\ldots,\varphi_m) \quad & \textit{iff} \quad \forall\rho = s_0a_0\ldots \in Run_\star(s,\mathcal{S}).\ \exists\sigma = \lambda_{i_0}a_0\ldots \in L(\mathcal{A}_\star) \\
& \qquad |\sigma| = |\rho| \textit{ and } \forall j.\ 0 \le j < |\rho|.\ s_j \models \varphi_{i_j}
\end{aligned}$$

For every formula φ, let $[\![\varphi]\!]_{\mathcal{S}} := \{s \in \mathcal{S} \mid s \models \varphi\}$.

Definition 5.8 (Fragments of AL) *$\exists$AL($\mathcal{F}$) is the fragment of AL that uses only constraints from $\mathcal{F}$, conjunction, disjunction and existential path quantification. $\forall$AL($\mathcal{F}$) is the fragment of AL that uses only constraints from $\mathcal{F}$, conjunction, disjunction and universal path quantification. Let X be (some fragment of) the logic AL. Then X_f (resp. X_ω, $X_{\omega c}$) denote the fragment of X where only automata on finite sequences (resp. Büchi, closed ω-automata) are used.*

AL is a weaker logic than the modal μ-calculus, but many widely known temporal logics are fragments of AL. Every propositional linear-time property, in particular LTL properties, can be expressed by means of automata [VW86, Tho90] and hence they can be expressed in AL. Then, it is easy to see that CTL* is a fragment of AL since every path formula in CTL* corresponds to an LTL formula. Thus, CTL is also a fragment of AL. Clearly, $\forall$AL and $\exists$AL subsume the positive universal and existential fragments of CTL* denoted $\forall$CTL* and $\exists$CTL* (notice that LTL is a fragment of $\forall$CTL*).

We consider two fragments of CTL called EF and EG. The logic EF uses SC predicates, boolean operators, the one-step next operator and the operator EF which is defined by $[\![EF\varphi]\!] = pre^*([\![\varphi]\!])$, i.e., a configuration s satisfies $EF\varphi$ if there is a reachable configuration from s which satisfies φ. The logic EG is defined like EF, except that the operator EF is replaced by the operator EG, which is defined as follows: $s \models EG\varphi$ iff there exists a complete run that starts at s and always satisfies φ. By a complete run we mean either an infinite run or a finite run ending in a deadlock.

We use the subscripts f or ω to denote the fragments of these logics obtained by interpreting their formulas on either finite or infinite paths only. Then, it can be seen that $\mathrm{EF} = \mathrm{EF}_f \subseteq \mathrm{CTL}_f \subseteq \mathrm{CTL}^*_f \subseteq \mathrm{AL}_f$. It can also be seen that EG_ω is a fragment of $\mathrm{AL}_{\omega c}$ but EG is not (due to the finite paths).

Figure 1 shows the relationship between several logics. An arc between two logics means that the higher logic in the graph is more expressive.

6 Model Checking

Definition 6.1 (Model checking and global model checking)

1. *The* model checking problem *is, given a configuration s and a formula φ, whether $s \in [\![\varphi]\!]_{\mathcal{S}}$,*
2. *The* global model checking problem *is whether for any formula φ the set $[\![\varphi]\!]_{\mathcal{S}}$ is effectively constructible.*

Lemma 6.2 *Let $\mathcal{S}$ be a lossy VASS. Then for every formula φ of the form $\exists \mathcal{A}_f(\pi_1, \ldots, \pi_m)$ where all the π_i are SC, the set $[\![\varphi]\!]_{\mathcal{S}}$ is SC definable and effectively constructible.*

Proof First we compute the product of $\mathcal{S}$ and $\mathcal{A}_f$ and obtain a new lossy VASS $\mathcal{S}'$. All states in the finite control of $\mathcal{S}'$ have the form (q, q') where q is a state in the finite control of $\mathcal{S}$ and q' is a state in $\mathcal{A}_f$. The initial state is the

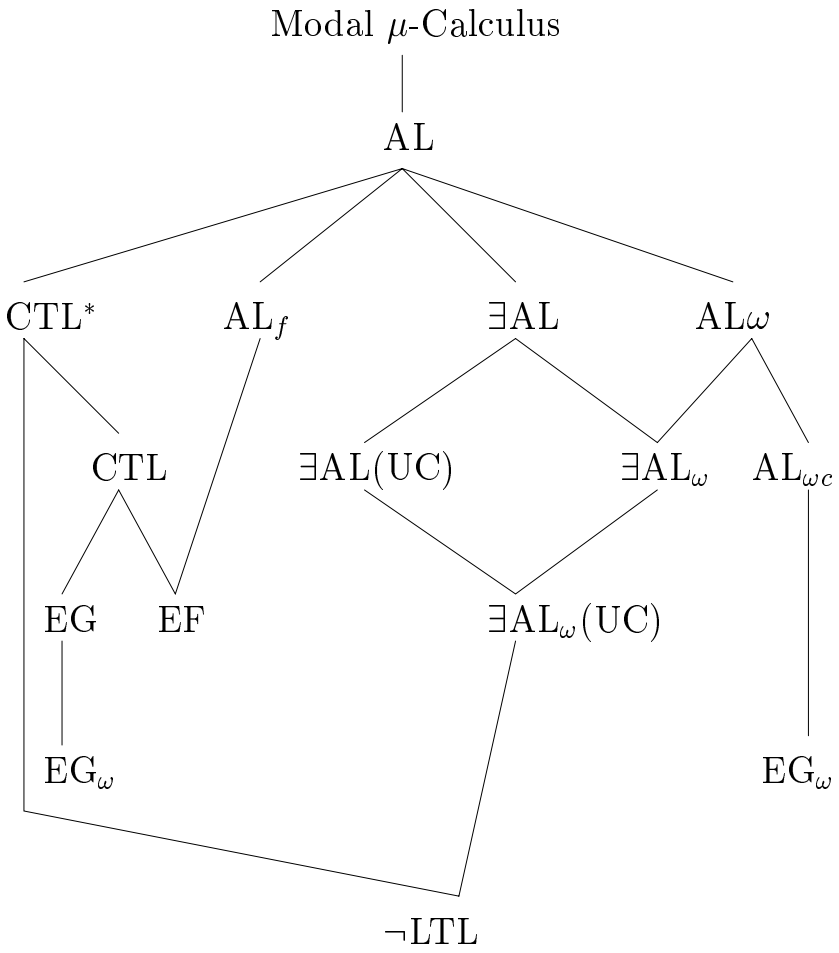


Figure 1: The relative expressive power of several logics.

product of the initial states in $\mathcal{S}$ and $\mathcal{A}_f$. A state in the finite control of $\mathcal{S}'$ is final iff the part of it in $\mathcal{A}_f$ is final. Similarly, each state in the finite control of $\mathcal{S}'$ is assigned the constraint π_i that is assigned to the $\mathcal{A}_f$-part of it. Every constraint π_i is SC definable. Thus the set of final configurations of $\mathcal{S}'$ can be written as a disjunction of canonical products of the form $\langle q_i, \vec{l_i} \leq \vec{x} \leq \vec{u}_i \rangle$, where q_i is a final state of $\mathcal{S}'$.

For every $\langle q_i, \vec{l_i} \leq \vec{x} \leq \vec{u}_i \rangle$ we compute a tree whose nodes are labeled with canonical products with $\langle q_i, \vec{l_i} \leq \vec{x} \leq \vec{u}_i \rangle$ at the root in the following way: For every node p with canonical product P we compute $pre(P)$ with respect to $\mathcal{S}'$. Since $\mathcal{S}'$ is lossy, this is a disjunction of upward closed canonical products.

For each of these we compute the intersection with the π_i assigned to the current state of the finite control of $\mathcal{S}'$. Thus we get a disjunction of canonical products of the form $\langle q, \vec{l} \leq \vec{x} \leq \vec{u} \rangle$ where the upper bound $\vec{u}$ is in a finite set of upper bounds $UB(\pi_1, \ldots, \pi_m)$ that occur in some canonical product in $\pi_1, \ldots, \pi_m$. Every child-node of p is labeled with one of the canonical products in this disjunction. We stop the construction of a branch of the tree when we encounter a node marked with a canonical product s.t. there is a previous node in the branch whose canonical product describes a larger set of states. The set of states of the finite control of $\mathcal{S}'$ is finite. Also all upper bounds used in canonical products in the tree are from the finite set $UB(\pi_1, \ldots, \pi_m)$. Thus by Dickson's Lemma the tree must be finite, because we eventually encounter larger lower bounds than previously in the branch.

Then we take the union of all canonical products in all these trees where the associated control state of $\mathcal{S}'$ has the form (q, q') where q' is an initial state in $\mathcal{A}_f$. We compute the projection on the first component q of the control state and thus get the description of a set of states of $\mathcal{S}$. This is $[\![\varphi]\!]_{\mathcal{S}}$. ■

Theorem 6.3 *The global model checking problem for lossy VASS and the logic AL_f is decidable.*

Proof By induction on the nesting-depth of operators $\exists \mathcal{A}_f$ in the formula. The result follows from boolean operations and Lemma 6.2. ■

The following results even hold for non-lossy VASS. The aim is to show decidability of the global model checking problem for VASS and the logic $\exists AL_\omega(UC)$. We define a generalized notion of configurations of VASS which includes the symbol ω. This symbol denotes arbitrarily high numbers of tokens on a place. It is used as an abbreviation in the following way:

$$\langle q, (\omega, \omega, \ldots, \omega, x_{k+1}, \ldots, x_n) \rangle \models \varphi \;\; :\Leftrightarrow$$
$$\exists n_1, \ldots, n_k \in I\!N.\; \langle q, (n_1, n_2, \ldots, n_k, x_{k+1}, \ldots, x_n) \rangle \models \varphi$$

(Of course the ω can occur at any position, e.g. $\langle q, (x_1, x_2, \omega, x_4, \omega, x_6) \rangle$.)

Lemma 6.4 *Let $\mathcal{S}$ be a VASS and φ a formula of the form $\exists \mathcal{A}_\omega(\pi_1, \ldots, \pi_m)$ where all the π_i are in UC. Let s be a generalized configuration of $\mathcal{S}$ (i.e. it can contain ω). It is decidable if $s \models \varphi$.*

Proof We compute the product of $\mathcal{S}$ and $\mathcal{A}_\omega$ and get a new VASS $\mathcal{S}'$. The states of the finite control of $\mathcal{S}'$ have the form (q, q') where q is a state of the finite control of $\mathcal{S}$ and q' is a state of $\mathcal{A}_\omega$. (q, q') is repeating iff q' is repeating in $\mathcal{A}_\omega$. It is assigned the constraint π_i that is assigned to q'. The initial state s' of $\mathcal{S}'$ is the product of s with the initial state q_0 of $\mathcal{A}_\omega$.

We search for an infinite run of $\mathcal{S}'$ that satisfies the assigned constraint π_i at every state and visits some repeating state infinitely often. The chances for finding such a run are always bigger if the initial state is larger, because all π_i are upward closed.

First we compute the Karp-Miller coverability graph [KM69] of $\mathcal{S}'$ with root s'. Only those nodes are considered that satisfy the assigned constraint π_i. By Dickson's Lemma this graph is finite. If there exists an infinite accepting run then there must be an infinite cyclic accepting run that starts at a Büchi-repeating node in this graph. This run corresponds to a cyclic path in the coverability graph that starts and ends at a Büchi-repeating node and has an overall positive effect of all fired transitions and can thus be repeated infinitely often.

For every Büchi-repeating node n in the coverability graph we compute a finite-state automaton A_n on finite sequences such that its labeled transition graph is the largest strongly connected subgraph containing n, and its initial state and its only final state is n. We label every arc in A_n with a unique symbol λ_i. To every λ_i we assign an effect-vector $\Delta_i \in \mathbb{Z}^n$ that describes the effect of the transition that was fired in the step from one node to the other. Let k be the number of states in A_n.

The aim is to find a cyclic path in A_n from node n back to n where the sum of all effect-vectors of all traversed arcs is $\geq \vec{0}$. (Note that the effect-vector of an arc that is traversed j times counts j times, i.e. it is multiplied by j.) Such a cyclic path with positive overall effect corresponds to a possible infinite accepting run of the system $\mathcal{S}'$.

Since n is the initial state and the only final state in A_n, every word in $L(A_n)$ corresponds to a cyclic path from n to n. For any word w, let $|w|_{\lambda_i}$ be the number of occurrences of λ_i in w. The question is now if there is a word $w \in L(A_n)$ s.t.

$$\sum_{1 \leq i \leq k} |w|_{\lambda_i} \Delta_i \geq \vec{0}$$

This is decidable, since the set $\{(|w|_{\lambda_1}, \ldots, |w|_{\lambda_k}) \mid w \in L(A_n)\}$ is semilinear by Parikh's Theorem [Par66]. We check this condition for every A_n, and we have $s \models \varphi$ if and only if a positive linear combination is found. ■

Lemma 6.5 *Let $\mathcal{S}$ be a VASS and φ a formula of the form $\exists \mathcal{A}_\omega(\pi_1, \ldots, \pi_m)$ where all the π_i are in UC. The set $[\![\varphi]\!]_{\mathcal{S}}$ is UC definable and effectively constructible.*

Proof The set $[\![\varphi]\!]_{\mathcal{S}}$ is upward closed, because all π_i are upward closed. Thus, this set is characterized by the finite set of its minimal elements (see

Lemma 3.2 and 3.3). To find the minimal elements, we use a construction that was described by Valk and Jantzen in [VJ85]. The important point here is that we can use Lemma 6.4 to check the existence of configurations that satisfy φ. For example, if $\langle q, (\omega, x_2, x_3)\rangle \models \varphi$ then we can check if $\langle q, (n_1, x_2, x_3)\rangle \models \varphi$ for $n_1 = 0$, $n_1 = 1$, $n_1 = 2$, ... until we find the minimal n_1 s.t. $\langle q, (n_1, x_2, x_3)\rangle \models \varphi$. ■

Theorem 6.6 *The global model checking problem is decidable for VASS and the logic $\exists AL_\omega(UC)$.*

Proof By induction on the nesting-depth of the formula and Lemma 6.5. ■

Theorem 6.7 *The global model checking problem is decidable for lossy VASS and the logic $\exists AL(UC)$.*

Proof By induction on the nesting depth of the formula and Theorem 6.3 and Theorem 6.6. ■

Theorem 6.8 *The model checking problem for lossy VASS and the logic $AL_{\omega c}$ is decidable.*

Proof Let $\mathcal{S}$ be a lossy VASS and φ a $AL_{\omega c}$ formula. Let s be some state of $\mathcal{S}$. We show decidability of the question $s \models \varphi$ by induction on the nesting-depth of the operator $\exists\mathcal{A}_{\omega c}$ in the formula. The base case where φ contains no operator $\exists\mathcal{A}_{\omega c}$ is trivial. In the general case we use boolean operations to reduce the problem to problems of the form $s \models \exists\mathcal{A}_{\omega c}(\varphi_1, \ldots, \varphi_n)$ where the φ_i have a smaller nesting-depth. We compute the product of $\mathcal{S}$ and $\mathcal{A}_{\omega c}$ and construct the tree of all possible successors of s that satisfy the assigned φ_i. (By induction hypothesis we can check if $s \models \varphi_i$ for any state s.) The construction of a branch stops if one of the following conditions is satisfied: (1) There is no successor that satisfies the assigned φ_i. (2) We reach a node with the same state of the finite control, but a larger marking than a previous node with a state s'.

By Dickson's Lemma every branch has finite length. Since the tree is finitely branching, it is finite and can be effectively constructed. Since $\mathcal{S}$ is lossy and all states in $\mathcal{A}_{\omega c}$ are repeating states, we know that $s \models \exists\mathcal{A}_{\omega c}(\varphi_1, \ldots, \varphi_n)$ iff some branch of the tree terminates by condition 2. ■

Theorem 6.9 *Model checking lossy VASS with the logic EG is decidable.*

Proof The proof is very similar to the proof of Theorem 6.8. The only differences are that one also has to consider finite runs that end in a deadlock and the one-step next operator. This can easily be added to the construction in Theorem 6.8. ■

Theorems 6.8 and 6.9 say that the model checking problem is decidable for a lossy VASS and an EG-formula/ $AL_{\omega c}$-formula φ. However, in both cases the set $[\![\varphi]\!]_{\mathcal{S}}$ is not effectively constructible (although it is SC definable). If it were constructible then Lemma 6.2 could be used to decide model checking lossy VASS with formulae of the form $EFEG_\omega\pi$, where π is a constraint in SC. However, this problem has very recently been shown to be undecidable.

Proposition 6.10 *Model checking lossy VASS with formulae of the form $EFEG_\omega\pi$, where π is a constraint in SC is undecidable.*

Proof This is a corollary of a more general undecidability result for lossy BPP (Basic Parallel Processes), which follows (not immediately) from the result on lossy counter machines in Proposition 7.2 (see [May98]). ■

Remark 6.11 *This undecidability result also implies undecidability of model checking lossy VASS with the logic $\exists AL_\omega$. One can encode properties of the form $EFEG_\omega\pi$ in $\exists AL_\omega$ in the following way: Let $\mathcal{A}_\omega$ be an automaton with states q, q', and transitions $q \to q$, $q \to q'$ and $q' \to q'$ which are labeled with any action. The predicate true is assigned to q and the predicate π is assigned to q'. q is the initial state and q' is the only repeating state. Let $\mathcal{A}'_\omega$ be an automaton with only one state q which is the initial state and repeating and a transition $q \to q$ with any action. The predicate π is assigned to q. Then for any lossy VASS s we have*

$$s \models EFEG_\omega\pi \iff s \models \mathcal{A}_\omega(true, \pi) \vee \mathcal{A}'_\omega(\pi)$$

7 Lossy VASS with Inhibitor Arcs

Lossy VASS can be extended with inhibitor arcs. This means introducing transitions that can only fire if some defined places are empty (i.e. they can test for zero). Thus lossy VASS with inhibitor arcs are equivalent to lossy counter machines. Normal VASS with inhibitor arcs are Turing-powerful, but lossy VASS with inhibitor arcs are not.

Theorem 7.1 *For lossy VASS with inhibitor arcs*

1. *the global model checking problem is decidable for the logic AL_f.*
2. *model checking is decidable for the logics $AL_{\omega c}$ and EG.*

Proof The constructions in Lemma 6.2 and Theorem 6.8 carry over directly to lossy VASS with inhibitor arcs. ■

Inhibitor arcs can never keep a transition from firing, because one can just loose the tokens on the places that inhibit it. However, after such a transition has fired, the number of tokens on the inhibiting places is fixed and known exactly. Such a guarantee is impossible to achieve in lossy VASS without inhibitor arcs. Thus not all results for lossy VASS carry over to lossy VASS with inhibitor arcs.

Proposition 7.2 *Let $\mathcal{S}$ be a lossy VASS with inhibitor arcs. It is undecidable if there exists an initial configuration s s.t. there is an infinite run of the system $(s, \mathcal{S})$.*

Proof This is a corollary of a more general undecidability result for lossy counter machines in [May98]. The main idea is that one can enforce that lossiness occurs only finitely often in the infinite run.

Theorem 7.3 *Model checking lossy VASS with inhibitor arcs with the logic LTL is undecidable.*

Proof We reduce the problem of Proposition 7.2 to the model checking problem. We construct a lossy VASS with inhibitor arcs $\mathcal{S}'$ that does the following: First it guesses an arbitrary configuration s of $\mathcal{S}$ doing only the atomic action a. Then it simulates $\mathcal{S}$ on s doing only the atomic action b. Let $\mathcal{A}_\omega$ be a Büchi-automaton with initial state q and repeating state q' and transitions $q \xrightarrow{a} q$, $q \xrightarrow{b} q'$ and $q' \xrightarrow{b} q'$. Let s' be the initial state of $\mathcal{S}'$. We have reduced the question of Proposition 7.2 to the question if $(s', \mathcal{S}') \models \exists \mathcal{A}_\omega(true, true)$. This question can be expressed in LTL. ■

It follows immediately that model checking lossy VASS with inhibitor arcs with $AL_\omega(UC)$ is undecidable. It is interesting to compare this result with Proposition 6.10. It shows that for undecidability of the model checking problem it suffices to have either inhibitor arcs in the system or downward closed constraints in the logic. One can be encoded in the other and vice versa.

The set $post^*(s)$ is DC definable since it is downward closed. However, it is not constructible for lossy VASS with inhibitor arcs (unlike for lossy VASS, see Theorem 4.3).

Theorem 7.4 *The set $post^*(s)$ is not constructible for lossy VASS with inhibitor arcs.*

Proof Boundedness is undecidable for reset Petri nets [DFS98]. This result carries over to lossy reset Petri nets. Lossy VASS with inhibitor arcs can simulate lossy reset Petri nets. It follows that boundedness is undecidable for lossy VASS with inhibitor arcs and thus $post^*(s)$ is not constructible. ■

8 Conclusion

We have addressed verification problems for VASS with lossy variables. These systems can model communicating systems with unbounded lossy unordered buffers. They can be seen as abstractions of (lossy) fifo-channel systems. Considering lossy VASS instead of lossy fifo-channel systems has several advantages: First, the set of reachable configurations backward and forward is effectively constructible, while for lossy channel systems only the backward reachability set is computable. This is interesting, since in practice forward reachability analysis can be more efficient than backward analysis, and it can be combined with efficient graph exploration techniques (like on-the-fly, partial-order based procedures) in order to avoid state-explosion. Another advantage of considering lossy VASS is that in addition to safety properties (Theorem 6.3), also liveness properties can be checked and fairness constraints can be taken into account (Theorem 6.7), while it has been shown that this is impossible for lossy fifo-channel systems [AJ96].

In our work, we have also established results for normal VASS and lossy VASS with inhibitor arcs (lossy counter machines). Interestingly, it turns out that these two models are incomparable. Moreover, all the positive/negative results we obtained for lossy VASS with inhibitor arcs are the same as for lossy fifo-channel systems. So, an interesting open question is whether these two models are comparable.

The following table summarizes the results on the decidability of model checking for VASS, lossy VASS with test for zero, lossy VASS and lossy fifo-channel systems. By '++' we denote the fact that for any formula φ the set $[\![\varphi]\!]$ is SC definable and effectively constructible (global model checking), while '+' means that only model checking is decidable. We denote by — that model checking is undecidable. The symbol '?' denotes an open problem.

Logic	**VASS**	**Lossy VASS+0**	**Lossy VASS**	**Lossy fifo**
AL_f/EF	— [Esp97]	++	++ [AJ93]	++ [AJ93]
$\exists AL_\omega(UC)/LTL$	++ /+[Esp94]	—	++	— [AJ96]
$\exists AL(UC)$	?	—	++	— [AJ96]
$AL_{\omega c}$/EG	— [EK95]	+	+ [AJ93]	+ [AJ93]
$\exists AL_\omega$/CTL	— [EK95]	—	—	— [AJ96]

The results in this table are new, except where references are given. For normal VASS and LTL, decidability of the model checking problem was known [Esp94], but the construction of the set $[\![\varphi]\!]$ is new. The results in [AJ93] are just about EF and EG formulas without nesting, not for the full logics AL_f and $AL_{\omega c}$.

As for the open problem in this table, note that we can decide for a VASS

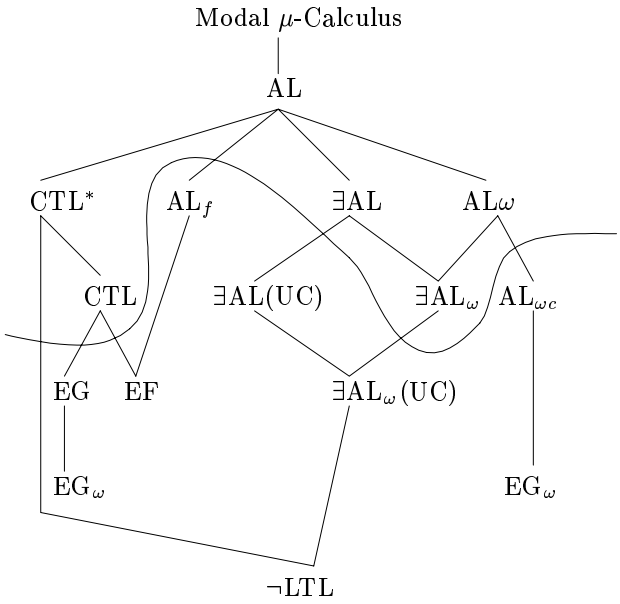


Figure 2: Decidability of the model checking problem for lossy VASS.

and any $\exists$AL(UC) formula, whether the set $[\![\varphi]\!]$ is SC definable, and in this case we can effectively construct it and decide the model checking question. However, if this set is not SC definable (in general this set could be even not semilinear), we cannot answer the model checking question.

The main new result in this paper is that some liveness properties, as described by $\exists AL_\omega(UC)$, are decidable for lossy VASS, unlike for lossy FIFO-channel systems. Also these liveness properties are undecidable for lossy VASS with test for zero. The general conclusion is that safety properties are always decidable for lossy systems, while for liveness properties this depends on the particular model and on the atomic propositions used in the logic.

The Figures 2 and 3 show the limits of the decidability of the model checking problem for lossy VASS and lossy VASS with inhibitor arcs. Model checking is decidable for the less expressive logics below the border and undecidable for those above it.

Acknowledgment: We thank Peter Habermehl for interesting discussions.

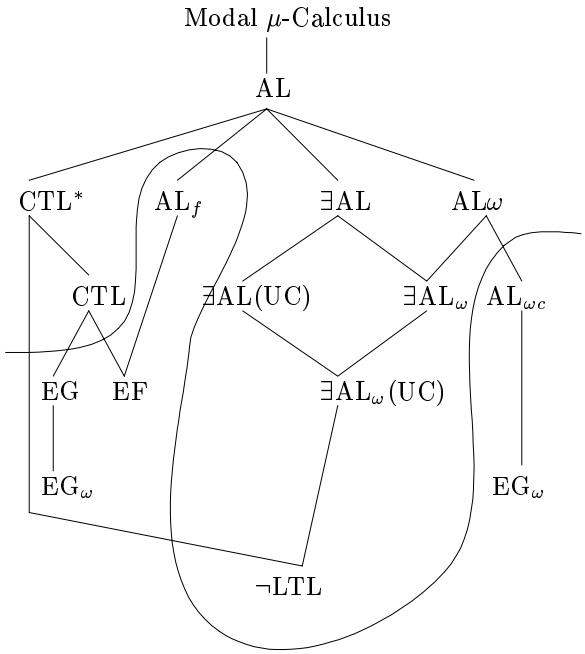


Figure 3: Decidability of the model checking problem for lossy VASS with inhibitor arcs.

References

[ABJ98] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *10th Intern. Conf. on Computer Aided Verification (CAV'98)*. LNCS 1427, June 1998.

[ACJT96] P. Abdulla, K. Cerans, B. Jonsson, and Y-K. Tsay. General Decidability Theorems for Infinite-state Systems. In *LICS'96*. IEEE, 1996.

[AJ93] P. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS'93*. IEEE, 1993.

[AJ96] P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS'97*. LNCS 1302, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Config-

urations. In *24th International Colloquium on Automata, Languages and Programming (ICALP'97)*. LNCS 1256, July 1997.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[CGK87] E. Clarke, O. Grümberg, and R. Kurshan. A Synthesis of two Approaches for Verifying Finite State Concurrent Systems. In *manuscript, Carnegie Mellon University*, 1987.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[Esp94] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Trees in Algebra and Programming – CAAP'94*, volume 787 of *LNCS*. Springer Verlag, 1994.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[GL94] O. Grumberg and D. Long. Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems*, 16, 1994.

[KM69] R. Karp and R. Miller. Parallel program schemata. *JCSS*, 3, 1969.

[May98] R. Mayr. Lossy counter machines. Technical Report TUM-I9827, TU-München, October 1998. wwwbrauer.informatik.tu-muenchen.de/∼mayrri.

[Par66] R.J. Parikh. On Context-Free Languages. *JACM*, 13, 1966.

[Tho89] W. Thomas. Computation Tree Logic and Regular ω-Languages. LNCS 354, 1989.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theo. Comp. Sci.* Elsevier Sci. Pub., 1990.

[VJ85] R. Valk and M. Jantzen. The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets. *Acta Informatica*, 21, 1985.

[VW86] M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS'86*. IEEE, 1986.

[Wol83] P. Wolper. Temporal logic can be more expreessive. *Information and Control*, 56, 1983.

Undecidable Problems in Unreliable Computations

Richard Mayr [a]

[a] *LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu, F-75251 Paris Cedex 05. France. E-mail:* mayr@liafa.jussieu.fr

Abstract

Lossy counter machines are defined as Minsky counter machines where the values in the counters can spontaneously decrease at any time. While termination is decidable for lossy counter machines, structural termination (termination for every input) is undecidable. This undecidability result has far-reaching consequences. Lossy counter machines can be used as a general tool to prove the undecidability of many problems, for example (1) The verification of systems that model communication through unreliable channels (e.g., model checking lossy fifo-channel systems and lossy vector addition systems). (2) Several problems for reset Petri nets, like structural termination, boundedness and structural boundedness. (3) Parameterized problems like fairness of broadcast communication protocols.

Key words: Counter machines, lossy counter machines, decidability

1 Introduction

Lossy counter machines (LCM) are defined just like Minsky counter machines [25], but with the addition that the values in the counters can spontaneously decrease at any time. This is called 'lossiness', since a part of the counter is lost. (In a different framework this corresponds to lost messages in unreliable communication channels.) There are many different kinds of lossiness, i.e., different ways in which the counters can decrease. For example, one can define that either a counter can only spontaneously decrease by 1, or it can only become zero, or it can change to any smaller value. All these different ways are described by different *lossiness relations* (see Section 2).

The addition of lossiness to counter machines weakens their computational power. Some types of lossy counter machines (with certain lossiness relations) are not Turing-powerful, since reachability and termination are decidable for them. Since lossy counter machines are *weaker* than normal counter machines, any *undecidability* result for lossy counter machines is particularly interesting.

The main result of this paper is that *structural termination* (termination for every input) is undecidable for every type of lossy counter machine (i.e., for every lossiness relation).
This result can be applied to prove the undecidability of many problems. To prove the undecidability of a problem X, it suffices to choose a suitable lossiness relation L and reduce the structural termination problem for lossy counter machines with lossiness relation L to the problem X. The important and nice point here is that problem X does *not* need to simulate a counter machine perfectly. Instead, it suffices if X can simulate a counter machine imperfectly, by simulating only a lossy counter machine. Furthermore, one can choose the right type of imperfection (lossiness) by choosing the lossiness relation L.
Thus lossy counter machines can be used as a general tool to prove the undecidability of problems. Firstly, they can be used to prove new undecidability results, and secondly they can be used to give more elegant, simpler and much shorter proofs of existing results (see Section 5).
Historically, the notion of 'lossiness' was first defined to model communication through unreliable channels. The main example are lossy fifo-channel systems, which are systems of finite-state processes that communicate through lossy fifo-channels (buffers) of unbounded length. These lossy fifo-channels are unreliable, because they can spontaneously lose messages. Since normal (non-lossy) fifo-channel systems are Turing-powerful, automatic analysis of them is restricted to special cases [4]. Lossy fifo-channel systems are not Turing-powerful, since reachability and some safety-properties are decidable for them [2,6,1]. However, some liveness-properties like the so-called 'recurrent-state problem' are undecidable even for lossy fifo-channel systems [3]. The result of this paper, the undecidability of structural termination for lossy counter machines, is much more general and subsumes this result (see Section 5).
The rest of the paper is structured as follows. In Section 2 we define lossiness relations and lossy counter machines. In Section 3 we show some decidable properties of lossy counter machines, and in Section 4 we prove the main undecidability result. Section 5 gives several examples how this result can be applied. In the last two sections we discuss possible generalizations and draw some conclusions.

2 Definitions

Definition 1 *A n-counter machine [25] M is described by a finite set of states Q, an initial state $q_0 \in Q$, a final state accept $\in Q$, n counters $c_1, \ldots, c_n$ and a finite set of instructions of the form $(q : \ c_i := c_i + 1; \mathsf{goto}\ q')$ or $(q : \ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1; \mathsf{goto}\ q'')$ where $i \in \{1, \ldots, n\}$ and $q, q', q'' \in Q$.*
A configuration *of M is described by a tuple $(q, m_1, \ldots, m_n)$ where $q \in Q$ and $m_i \in \mathbb{N}$ is the content of the counter c_i ($1 \leq i \leq n$). The size of a configuration*

is defined by $size((q, m_1, \ldots, m_n)) := \sum_{i=1}^{n} m_i$. *The possible computation steps are defined as follows:*

(1) $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_i + 1, \ldots, m_n)$
if there is an instruction $(q : \ c_i := c_i + 1; \mathsf{goto}\ q')$.

(2) $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_n)$
if there is an instruction $(q : \ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1; \mathsf{goto}\ q'')$ *and* $m_i = 0$.

(3) $(q, m_1, \ldots, m_n) \to (q'', m_1, \ldots, m_i - 1, \ldots, m_n)$
if there is an instruction $(q : \ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1; \mathsf{goto}\ q'')$ *and* $m_i > 0$.

A counter machine is deterministic *iff for every control-state* $q \in Q$ *there is at most one instruction* $(q : \ \ldots\ldots)$ *at this control-state. A* run *of a counter machine is a (possibly infinite) sequence of configurations* $s_0, s_1, \ldots$ *with* $s_0 \to s_1 \to s_2 \to s_3 \to \ldots$.

Now we define *lossiness relations*, which describe spontaneous changes in the configurations of lossy counter machines.

Definition 2 *Let* $\xrightarrow{s}$ *(for 'sum') be a relation on configurations of* n*-counter machines which is defined as follows.*

$$(q, m_1, \ldots, m_n) \xrightarrow{s} (q', m'_1, \ldots, m'_n) :\Leftrightarrow$$
$$(q, m_1, \ldots, m_n) = (q', m'_1, \ldots, m'_n) \vee$$
$$\left(q = q' \wedge \sum_{i=1}^{n} m_i > \sum_{i=1}^{n} m'_i\right)$$

This relation means that either nothing is changed or the sum of all counters strictly decreases. Let id be the identity relation. A relation $\xrightarrow{l}$ *is a* lossiness relation *iff* $id \subseteq \xrightarrow{l} \subseteq \xrightarrow{s}$. *A* lossy counter machine *(LCM) is given by a counter machine* M *and a lossiness relation* $\xrightarrow{l}$. *Let* $\to$ *be the normal transition relation of* M. *The lossy transition relation* $\Longrightarrow$ *of the lossy counter machine is defined by*

$$s_1 \Longrightarrow s_2 \ :\Leftrightarrow\ \exists s'_1, s'_2.\ s_1 \xrightarrow{l} s'_1 \to s'_2 \xrightarrow{l} s_2$$

An arbitrary lossy counter machine *is a lossy counter machine with an arbitrary (unspecified) lossiness relation. The following relations are lossiness relations:*

Perfect *The relation id is a lossiness relation. Thus arbitrary lossy counter machines subsume normal counter machines.*

Classic Lossiness *The* classic lossiness relation $\xrightarrow{cl}$ *is defined by*

$$(q, m_1, \ldots, m_n) \xrightarrow{cl} (q', m'_1, \ldots, m'_n) \ :\Leftrightarrow\ q = q' \ \wedge\ \forall i.\, m_i \geq m'_i$$

Here the contents of the counters can become any smaller value. A relation $\xrightarrow{l}$ *is called a* subclassic lossiness relation *iff* $id \subseteq \xrightarrow{l} \subseteq \xrightarrow{cl}$.

Bounded Lossiness *A counter can lose at most $x \in \mathbb{N}$ before and after every computation step. Here the lossiness relation $\xrightarrow{l(x)}$ is defined by*

$$(q, m_1, \ldots, m_n) \xrightarrow{l(x)} (q', m'_1, \ldots, m'_n) :\Leftrightarrow$$
$$q = q' \wedge \forall i.\, m_i \geq m'_i \geq max\{0, m_i - x\}$$

Note that $\xrightarrow{l(x)}$ is a subclassic lossiness relation for every $x \in \mathbb{N}$.

Reset Lossiness *If a counter is tested for zero, then it can suddenly become zero. The lossiness relation $\xrightarrow{rl}$ is defined as follows:*
$(q, m_1, \ldots, m_n) \xrightarrow{rl} (q, m'_1, \ldots, m'_n)$ iff for all i either
(1) $m_i = m_i$, or
(2) $m'_i = 0$ and there is an instruction
$(q :$ If $c_i = 0$ then goto q' else $c_i := c_i - 1$; goto $q'')$.
Note that $\xrightarrow{rl}$ is a subclassic lossiness relation.

The definition of these lossiness relations carries over to other models like Petri nets [27], where places are considered instead of counters and the control-states q are ignored.

Definition 3 *For any arbitrary lossy n-counter machine and any configuration s let $runs(s)$ be the set of runs that start at configuration s. (There can be more than one run if the counter machine is nondeterministic or lossy.) Let $runs^\omega(s)$ be the set of infinite runs that start at configuration s. A run $r = \{(q^i, m^i_1, \ldots, m^i_n)\}_{i=0}^\infty \in runs^\omega(s)$ is* space-bounded *iff $\exists c \in \mathbb{N}. \forall i. \sum_{j=1}^n m^i_j \leq c$. Let $runs^\omega_b(s)$ be the space-bounded infinite runs that start at s. For a run r and a configuration s we write $s \in r$ to indicate that s is one of the configurations that occur in r. An (arbitrary lossy) n-counter machine M is*

zero-initializing *iff in the initial state q_0 it first sets all counters to 0.*
space-bounded *iff the space used by M is bounded by a constant c.*

$$\exists c \in \mathbb{N}. \forall r \in runs((q_0, 0, \ldots, 0)). \forall s \in r.\, size(s) \leq c$$

input-bounded *iff in every run from any configuration the size of every reached configuration is bounded by the size of the input.*

$$\forall s. \forall r \in runs(s). \forall s' \in r.\, size(s') \leq size(s)$$

strongly-cyclic *iff every infinite run from any configuration visits the initial state q_0 infinitely often.*

$$\forall q \in Q, m_1, \ldots, m_n \in \mathbb{N}. \forall r \in runs^\omega((q, m_1, \ldots, m_n)).$$
$$\exists m'_1, \ldots, m'_n \in \mathbb{N}. (q_0, m'_1, \ldots, m'_n) \in r$$

bounded-strongly-cyclic *iff every space-bounded infinite run from any configuration visits the initial state q_0 infinitely often.*

$$\forall q \in Q, m_1, \ldots, m_n \in \mathbb{N}. \forall r \in runs_b^\omega((q, m_1, \ldots, m_n)).$$
$$\exists m'_1, \ldots, m'_n \in \mathbb{N}. (q_0, m'_1, \ldots, m'_n) \in r$$

If M is input-bounded then it is also space-bounded. If M is strongly-cyclic then it is also bounded-strongly-cyclic. If M is input-bounded and bounded-strongly-cyclic then it is also strongly-cyclic.

3 Decidable Properties

Since arbitrary LCM subsume normal counter machines, no interesting properties are decidable for them. However, some problems are decidable for classic LCM (with the classic lossiness relation). They are not Turing-powerful. The following results in this section are special cases of positive decidability results in [5,6,2].

Lemma 4 (Dickson's Lemma [10])
Given an infinite sequence of vectors $\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots$ in $\mathbb{N}^k$ there are $i < j$ s.t. $\vec{x}_i \leq \vec{x}_j$ ($\leq$ taken componentwise).

Lemma 5 *Let M be a classic LCM and s a configuration of M. The set $pre^*(s) := \{s' \mid s' \Longrightarrow^* s\}$ of predecessors of s is effectively constructible.*

PROOF. *Since M is a classic LCM, the set $pre^*(s)$ is upward closed and can thus be characterized by its finitely many minimal elements. These minimal elements can be effectively constructed because of Dickson's Lemma [10] (see also [5]).* □

Theorem 6 *Reachability is decidable for classic LCM.*

PROOF. *Given two configurations s and s', the question if $s \Longrightarrow^* s'$ is equivalent to $s \in pre^*(s')$. This is decidable by Lemma 5.* □

Lemma 7 *Let M be a classic LCM with initial configuration s_0. It is decidable if there is an infinite run that starts at s_0, i.e., if $runs^\omega(s_0) \neq \emptyset$.*

PROOF. *We analyze all runs by breadth-first search. If there is no infinite run then all runs will eventually terminate. Thus, the algorithm will terminate and give the correct answer 'no'. If there is an infinite run, then, by Dickson's Lemma [10], we will eventually reach a configuration s s.t. there is a previous configuration s' in the same run with $s \geq s'$. In this case there is an infinite cyclic run from s' to s', because M is classical lossy. Thus, in this case the algorithm also terminates and gives the correct answer 'yes'.* □

In [2] Abdulla and Jonsson proved a more general result that subsumes Lemma 7. They showed that the existence of an infinite run from a given initial configuration is decidable even for lossy FIFO-channel systems.

Theorem 8 *Termination is decidable for classic LCM.*

PROOF. *A classic LCM M with initial configuration s_0 is terminating iff $runs^{\omega}(s_0) = \emptyset$. This is decidable by Lemma 7.* □

It has been shown in [5] that even model checking classic LCM with the temporal logics EF and EG (natural fragments of computation tree-logic (CTL) [9,14]) is decidable.

Another interesting observation is that Petri nets and classical lossy counter machines are incomparable. For Petri nets, model checking with the temporal logic EF is undecidable, but model checking with LTL is decidable [15]. For classical lossy counter machines it is just vice-versa. For classical lossy counter machines model checking with EF is decidable [5], but model checking with LTL is undecidable [5] (see also Theorem 11).

4 The Undecidability Result

We show that structural termination (i.e., termination for every input) is undecidable for LCM for every lossiness relation. We start with the problem CM, which was shown to be undecidable by Minsky [25].

CM

Instance: A deterministic 2-counter machine M with initial state q_0.

Question: Does M accept $(q_0, 0, 0)$?

We reduce the problem CM to the following problem.

BSC-ZI-CM$_b^{\omega}$

Instance: A deterministic bounded-strongly-cyclic, zero-initializing 3-counter machine M with initial state q_0.

Question: Does M have an infinite space-bounded run from $(q_0, 0, 0, 0)$, i.e., $runs_b^{\omega}((q_0, 0, 0, 0)) \neq \emptyset$?

Lemma 9 *BSC-ZI-CM$_b^{\omega}$ is undecidable.*

PROOF. *We reduce CM to BSC-ZI-CM$_b^{\omega}$. Let M be a 2-counter machine with initial state q_0. We construct a 3-counter machine M' as follows: First M' sets all three counters to 0. Then it does the same as M, except that after every instruction it increases the third counter c_3 by 1. Every instruction of M of the form $(q : \ c_i := c_i + 1;$ goto $q')$ with $(1 \leq i \leq 2)$ is replaced by $(q : c_i := c_i + 1;$ goto $q_2)$ and $(q_2 : c_3 := c_3 + 1;$ goto $q')$, where q_2 is a new state. Every instruction of the form*

$$(q : \ \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{goto } q'')$$

with $(1 \leq i \leq 2)$ *is replaced by*

$$q: \text{ If } c_i = 0 \text{ then goto } q_2 \text{ else } c_i := c_i - 1; \text{ goto } q_3$$
$$q_2 : c_3 := c_3 + 1; \text{ goto } q'$$
$$q_3 : c_3 := c_3 + 1; \text{ goto } q''$$

where q_2, q_3 *are new states.*
Finally, we replace the accepting state 'accept' of M *by the initial state* q_0' *of* M'*, i.e., we replace every instruction (*goto *accept) by (*goto q_0'*). M' is deterministic, because M is deterministic. M' is zero-initializing by definition. M' is bounded-strongly-cyclic, because c_3 is increased after every instruction and only set to zero at the initial state q_0'.*

$\Rightarrow$ *If M is a positive instance of CM then it has exactly one accepting run from $(q_0, 0, 0)$ (we assume without restriction that M has exactly one accepting state and that this state has no outgoing transitions). This run has finite length and is therefore space-bounded. Then M' has an infinite space-bounded cyclic run that starts at $(q_0', 0, 0, 0)$. Thus M' is a positive instance of BSC-ZI-CM$_b^\omega$.*

$\Leftarrow$ *If M' is a positive instance of BSC-ZI-CM$_b^\omega$ then there exists an infinite space-bounded run that starts at the configuration $(q_0', 0, 0, 0)$. By the construction of M' this run contains an accepting run of M from the configuration $(q_0, 0, 0)$. Thus M is a positive instance of CM.* $\square$

Now we consider the central problem for lossy counter machines.

$\underline{\exists n\text{LCM}^\omega}$

Instance: A strongly-cyclic, input-bounded 4-counter LCM M with initial state q_0.

Question: Does there exist an $n \in \mathbb{N}$ s.t. $runs^\omega((q_0, 0, 0, 0, n)) \neq \emptyset$?

Theorem 10 *$\exists n$LCM$^\omega$ is undecidable for every lossiness relation.*

PROOF. *We reduce BSC-ZI-CM$_b^\omega$ to $\exists n$LCM$^\omega$ with any lossiness relation $\xrightarrow{l}$. For any bounded-strongly-cyclic, zero-initializing 3-counter machine M we construct a strongly-cyclic, input-bounded lossy 4-counter machine M' with initial state q_0' and lossiness relation $\xrightarrow{l}$ as follows: The 4-th counter c_4 holds the 'capacity'. In every operation it is changed in a way s.t. the sum of all counters never increases. (More exactly, the sum of all counters can increase by 1, but only if it was decreased by 1 in the previous step.) Every instruction of M of the form $(q: \ c_i := c_i + 1;$ goto $q')$ with $(1 \leq i \leq 3)$ is replaced by*

$$q: \text{ If } c_4 = 0 \text{ then goto } \textit{fail} \text{ else } c_4 := c_4 - 1; \text{ goto } q_2$$
$$q_2 : c_i := c_i + 1; \text{ goto } q'$$

where 'fail' is a special final state and q_2 is a new state. Every instruction of the form $(q:\ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1; \mathsf{goto}\ q'')$ *with* $(1 \leq i \leq 3)$ *is replaced by*

$$q:\ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1;\ \mathsf{goto}\ q_2$$

$$q_2 : c_4 := c_4 + 1;\ \mathsf{goto}\ q''$$

where q_2 is a new state.

M' is bounded-strongly-cyclic, because M is bounded-strongly-cyclic. M' is input-bounded, because every run from a configuration $(q, m_1, \ldots, m_4)$ is space-bounded by $m_1 + m_2 + m_3 + m_4$. Thus M' is also strongly-cyclic.

$\Rightarrow$ *If M is a positive instance of BSC-ZI-CM$_b^\omega$ then there exists a $n \in \mathbb{N}$ and an infinite run of M that starts at $(q_0, 0, 0, 0)$, visits q_0 infinitely often and always satisfies $c_1 + c_2 + c_3 \leq n$. Since id $\subseteq \xrightarrow{l}$, there is also an infinite run of M' that starts at $(q_0, 0, 0, 0, n)$, visits q_0 infinitely often and always satisfies $c_1 + c_2 + c_3 + c_4 \leq n$. Thus M' is a positive instance of $\exists n LCM^\omega$.*

$\Leftarrow$ *If M' is a positive instance of $\exists n LCM^\omega$ then there exists an $n \in \mathbb{N}$ s.t. there is an infinite run that starts at the configuration $(q_0', 0, 0, 0, n)$. This run is space-bounded, because it always satisfies $c_1 + c_2 + c_3 + c_4 \leq n$. By the construction of M', the sum of all counters can only increase by 1 if it was decreased by 1 in the previous step. By the definition of lossiness (see Def. 2) we get the following: If lossiness occurs (when the contents of the counters spontaneously change) then this strictly and permanently decreases the sum of all counters. It follows that lossiness can only occur at most n times in this infinite run and the sum of all counters is bounded by n. Thus there is an infinite suffix of this run of M' where lossiness does not occur. Thus there exist $q' \in Q$, $m_1', \ldots, m_4' \in \mathbb{N}$ s.t. an infinite suffix of this run of M' without lossiness starts at $(q', m_1', \ldots, m_4')$. It follows that there is an infinite space-bounded run of M that starts at $(q', m_1', \ldots, m_3')$. Since M is bounded-strongly-cyclic, this run must eventually visit q_0. Thus there exist $m_1'', \ldots, m_3'' \in \mathbb{N}$ s.t. an infinite space-bounded run of M starts at $(q_0, m_1'', \ldots, m_3'')$. Since M is zero-initializing, there is an infinite space-bounded run of M that starts at $(q_0, 0, 0, 0)$. Thus M is a positive instance of BSC-ZI-CM$_b^\omega$.* $\square$

Note that this undecidability result even holds under the additional condition that the LCMs are strongly-cyclic and input-bounded.

This result can be used to show that model checking LCM with the temporal logics CTL (computation-tree logic [9,14]) and LTL (linear-time temporal logic [28]) is undecidable, since the question of $\exists n\mathrm{LCM}^\omega$ can be encoded in these logics.

Theorem 11 *Model checking LCM with the temporal logics CTL and LTL is undecidable for every lossiness relation.*

PROOF. *Let M be a lossy 4-counter LCM with lossiness relation $\xrightarrow{l}$ and initial state q_0. We construct the LCM M' as follows: Let q_0' be the new initial state of M'. M' has the same instructions as M plus the following ones:*

$$q_0' : c_4 := c_4 + 1;\ \mathsf{goto}\ q_0'$$

$$q_0' : c_4 := c_4 + 1;\ \mathsf{goto}\ q_0$$

We label these two new instructions with action 'a' and all others with action 'b'. Then we have that M is a positive instance of $\exists nLCM^{\omega}$ iff M' satisfies the LTL formula $(q_0', 0, 0, 0, 0) \models \langle a \rangle true\, U\, (\langle b \rangle true\, wU\, false)$ or the CTL formula $(q_0', 0, 0, 0, 0) \models E[\langle a \rangle true\, U\, E[(\langle b \rangle true\, wU\, false)]]$ □

The following two variants of the structural termination problem are equivalent. An LCM is a positive instance of variant 1 iff it is a positive instance of variant 2, because of the imposed condition that the LCM is strongly-cyclic. The only reason why we define both variants is to point out this fact.

STRUCTTERM-LCM, VARIANT 1

Instance: A strongly-cyclic, input-bounded 4-counter LCM M with initial state q_0.

Question: Does M terminate for all inputs from q_0 ? Formally: $\forall n_1, \ldots, n_4 \in \mathbb{N}.\ runs^{\omega}((q_0, n_1, n_2, n_3, n_4)) = \emptyset$?

STRUCTTERM-LCM, VARIANT 2

Instance: A strongly-cyclic, input-bounded 4-counter LCM M with initial state q_0.

Question: Does M terminate for all inputs from every control state q ? Formally: $\forall n_1, \ldots, n_4 \in \mathbb{N}. \forall q \in Q.$
$runs^{\omega}((q, n_1, n_2, n_3, n_4)) = \emptyset$?

Theorem 12 *Structural termination is undecidable for lossy counter machines. Both variants of STRUCTTERM-LCM are undecidable for every lossiness relation.*

PROOF. *The proof of Theorem 10 carries over, because the LCM is strongly-cyclic and the 3-CM in BSC-ZI-CM$_b^{\omega}$ is zero-initializing.* □

SPACE-BOUNDEDNESS FOR LCM

Instance: A strongly-cyclic 4-counter LCM M with the initial configuration $(q_0, 0, 0, 0, 0)$.

Question: Is M space-bounded ?

Theorem 13 *Space-boundedness for LCM is undecidable for all lossiness relations.*

PROOF. *We reduce BSC-ZI-CM$_b^\omega$ to the space-boundedness problem for LCM. Let M be the 3-CM from BSC-ZI-CM$_b^\omega$. We take the LCM M' from the proof of Theorem 10 and modify it as follows (obtaining a new LCM M''): At the final state 'fail' we do not stop. Instead we add c_1, c_2 and c_3 to c_4, set c_1, c_2 and c_3 to 0 and increase c_4 by 1 and go to the initial state q_0'' of M''. Formally, this is defined by*

$$
\begin{aligned}
&\mathit{fail}: \mathsf{If}\ c_1 = 0\ \mathsf{then\ goto}\ f_2\ \mathsf{else}\ c_1 := c_1 - 1;\ \mathsf{goto}\ d_1\\
&d_1: \quad c_4 := c_4 + 1;\ \mathsf{goto}\ \mathit{fail}\\
&f_2: \quad \mathsf{If}\ c_2 = 0\ \mathsf{then\ goto}\ f_3\ \mathsf{else}\ c_2 := c_2 - 1;\ \mathsf{goto}\ d_2\\
&d_2: \quad c_4 := c_4 + 1;\ \mathsf{goto}\ f_2\\
&f_3: \quad \mathsf{If}\ c_3 = 0\ \mathsf{then\ goto}\ f_4\ \mathsf{else}\ c_3 := c_3 - 1;\ \mathsf{goto}\ d_3\\
&d_3: \quad c_4 := c_4 + 1;\ \mathsf{goto}\ f_3\\
&f_4: \quad c_4 := c_4 + 1;\ \mathsf{goto}\ q_0''
\end{aligned}
$$

The initial configuration of M'' is $(q_0'', 0, 0, 0, 0)$. Now we show that M is a positive instance of BSC-ZI-CM$_b^\omega$ iff M'' is bounded.

$\Rightarrow$ *If M is a positive instance of BSC-ZI-CM$_b^\omega$ then it uses only a finite amount k of space, i.e., we have always $c_1 + c_2 + c_3 < k$ in both M and M''. If the value in c_4 becomes larger than k then there are two cases.*

(1) If M'' does not lose then it will enter an infinite space-bounded cyclic computation which never visits the state 'fail' again. Thus these runs of M'' are bounded.

(2) In order to visit the state 'fail' again M'' must lose at least once. This is at most compensated in the state 'fail' (the sum of the counters is increased by 1), but not more than that. Thus these runs of M'' are bounded as well.

Thus all computations of M'' from $(q_0'', 0, 0, 0, 0)$ are space-bounded.

$\Leftarrow$ *If M is a negative instance of BSC-ZI-CM$_b^\omega$ then the computation of M'' from $(q_0'', 0, 0, 0, 0)$ without losses will visit the state 'fail' infinitely often and the sum of all counters will become arbitrarily high. (The run without losses is one possible run, since by Def. 2 id $\subseteq \xrightarrow{l}$.) Thus M'' is not space-bounded.* $\square$

Remark 14 *It follows directly from Theorem 13 that the set of reachable configurations of a LCM cannot be effectively constructed. (If one could construct this set then one could decide boundedness). In particular, this non-constructibility result also holds for classical LCM. The set of reachable configurations of a classical LCM is always semilinear, since it is downward closed. Thus, the set of reachable configurations of a classical LCM is semilinear, but not effectively semilinear.*

It has already been stated in [6] that the regular expression that describes the set of reachable configurations of a lossy fifo-channel system cannot be effectively constructed, although it always exists. (The proof in [6] contains a slight error.) This result is subsumed by the more general Theorem 13 and Remark 14.

STRUCTURAL SPACE-BOUNDEDNESS FOR LCM

Instance: A strongly-cyclic 5-counter LCM. M.
Question: Is M space-bounded for every initial configuration $(q, n_1, n_2, n_3, n_4, n_5)$?

Theorem 15 *Structural space-boundedness for LCM is undecidable for every lossiness relation.*

PROOF. *The proof is similar to Theorem 12. An extra counter c_5 is used to count the length of the run. It is unbounded iff the run is infinite. All other counters are bounded.* □

5 Applications

Lossy counter machines can be used to prove the undecidability of many problems.

5.1 Lossy Fifo-Channel Systems

Fifo-channel systems are systems of finitely many finite-state processes that communicate with each other by sending messages via unbounded fifo-channels (queues, buffers). In lossy fifo-channel systems these channels are lossy, i.e., they can spontaneously lose (arbitrarily many) messages. This can be used to model communication via unreliable channels. While normal fifo-channel systems are Turing-powerful, some safety-properties are decidable for lossy fifo-channel systems [2,6,1]. However, liveness properties are undecidable even for lossy fifo-channel systems. In [3] Abdulla and Jonsson showed the undecidability of the *recurrent-state problem* for lossy fifo-channel systems. This problem is if certain states of the system can be visited infinitely often. The undecidable core of the problem is essentially if there exists an initial configuration of a lossy fifo-channel system s.t. it has an infinite run. The undecidability proof in [3] was done by a reduction from a variant of Post's correspondence problem, namely 2-permutation PCP. The undecidability of 2-permutation PCP has been shown by Ruohonen in [29]. There is some confusion of the names of the problems in the literature. Abdulla and Jonsson [3] use Ruohonen's result on the undecidability of 2-permutation PCP and cite [29], but they refer to the problem '2-permutation PCP' as 'cyclic PCP'. However, the real cyclic PCP is a different problem, which is also defined and shown to be undecidable by Ruohonen in [29].

Lossy counter machines can be used to give a much simpler proof of the undecidability results for lossy FIFO-channel systems. The lossiness of lossy fifo-channel systems is classic lossiness, i.e., the contents of a fifo-channel can change to any substring at any time. A lossy fifo-channel system can simulate a classic LCM (with some additional deadlocks) in the following way: Every lossy fifo-channel contains a string in X^* (for some symbol X) and is used as a classic lossy counter. The length of the string encodes the value in the counter. The only problem is the test for zero. We test the emptiness of a fifo-channel by adding a special symbol Y and removing it in the very next step. If it can be done then the channel is empty (or has become empty by lossiness). If this cannot be done, then the channel was not empty or the symbol Y was lost. In this case we get a deadlock. These additional deadlocks do not affect the existence of infinite runs, and thus the results of Section 4 carry over. Thus the problem $\exists n\text{LCM}^\omega$ (for the classic lossiness relation) can be reduced to the problem above for lossy fifo-channel systems and the undecidability follows immediately from Theorem 10.

5.2 Model Checking Lossy Basic Parallel Processes

Petri nets [27] (also described as 'vector addition systems' in a different framework) are a widely known formalism used to model concurrent systems. They can also be seen as counter machines without the ability to test for zero, and are not Turing-powerful, since the reachability problem is decidable for them [22]. Basic Parallel Processes [7] correspond to communication-free nets, the (very weak) subclass of labeled Petri nets where every transition has exactly one place in its preset. They have been studied intensively in the framework of model checking and semantic equivalences (e.g., [17,23,24,8,20,26]).

An instance of the *model checking problem* is given by a system S (e.g., a counter machine, Petri net, pushdown automaton,...) and a temporal logic formula φ. The question is if the system S has the properties described by φ, denoted $S \models \varphi$.

The branching-time temporal logics EF, EG and EG_ω are defined as extensions of Hennessy-Milner Logic [18,19,14] by the operators EF, EG and EG_ω, respectively. $s \models EF\varphi$ iff there exists an s' s.t. $s \xrightarrow{*} s'$ and $s' \models \varphi$. $s_0 \models EG_\omega\varphi$ iff there exists an infinite run $s_0 \to s_1 \to s_2 \to \ldots$ s.t. $\forall i.\, s_i \models \varphi$. EG is similar, except that it also includes finite runs that end in a deadlock. Alternatively, EF and EG can be seen as fragments of computation-tree logic (CTL [9,14]), since $EF\varphi = E[true\,\mathcal{U}\,\varphi]$ and $EG\varphi = E[\varphi\, w\mathcal{U}\, false]$.

Model checking Petri nets with the logic EF is undecidable [15], but model checking Basic Parallel Processes with EF is *PSPACE*-complete [23]. Model checking Basic Parallel Processes with EG is undecidable [17]. It is different for lossy systems: By induction on the nesting-depth of the operators EF, EG and EG_ω, and constructions similar to the ones in Lemma 5 and Lemma 7, it can be shown that model checking classic LCM with the logics EF, EG and

EG_ω is decidable. Thus it is also decidable for classical lossy Petri nets and classical lossy Basic Parallel Processes (see [5]).
However, model checking lossy Basic Parallel Processes with nested EF and EG/EG_ω operators is still undecidable for every subclassic lossiness relation. This is quite surprising, since lossy Basic Parallel Processes are an extremely weak model of infinite-state concurrent systems and the temporal logic used is very weak as well. (Note in particular that lossy Basic Parallel Processes are *normed*, i.e., from every reachable state there is a terminating computation.)

Theorem 16 *Model checking lossy Basic Parallel Processes (with any subclassic lossiness relation) with formulae of the form $EF\,EG_\omega\Phi$, where Φ is a Hennessy-Milner Logic formula, is undecidable.*

PROOF. *Esparza and Kiehn showed in [17] that for every counter machine M (with all counters initially 0) a Basic Parallel Processes P and a Hennessy-Milner Logic formula φ can be constructed s.t. M does not halt iff $P \models EG_\omega\varphi$. The construction carries over to subclassic LCM and subclassic lossy Basic Parallel Processes. The control-states of the counter machine are modeled by special places of the Basic Parallel Processes. In every infinite run that satisfies φ exactly one of these places is marked at any time.*
We reduce $\exists nLCM^\omega$ to the model checking problem. Let M be a subclassic LCM. Let P be the corresponding Basic Parallel Processes as in [17] and let φ be the corresponding Hennessy-Milner Logic formula as in [17]. We use the same subclassic lossiness relation on M and on P. P stores the contents of the 4-th counter in a place Y. Thus $P\|Y^n$ corresponds to the configuration of M with n in the 4-th counter (and 0 in the others). We define a new initial state X and transitions $X \xrightarrow{a} X\|Y$ and $X \xrightarrow{b} P$, where a and b do not occur in P. Let $\Phi := \varphi \wedge \neg\langle b\rangle true$. Then M is a positive instance of $\exists nLCM^\omega$ iff $X \models EF\,EG_\omega\Phi$. The result follows from Theorem 10. □

For Petri nets and Basic Parallel Processes, the meaning of Hennessy-Milner Logic formulae can be expressed by boolean combinations of constraints of the form $p \geq k$ (at least k tokens on place p). Thus the results also hold if boolean combinations of such constraints are used instead of Hennessy-Milner Logic formulae. Another consequence of Theorem 16 is that model checking lossy Petri nets with CTL is undecidable.

5.3 Reset/Transfer Petri Nets

Reset Petri nets are an extension of Petri nets by the addition of reset-arcs. A reset-arc between a transition and a place has the effect that, when the transition fires, all tokens are removed from this place, i.e., it is reset to zero. Transfer nets and transfer arcs are defined similarly, except that all tokens on this place are moved to some different place. It was shown in [12] that termination is decidable for 'Reset Post G-nets', a more general extension

of Petri nets that subsumes reset nets and transfer nets. (For normal Petri nets termination is *EXPSPACE*-complete [30]). While boundedness is trivially decidable for transfer nets, the same question for reset nets was open for some time (and even a wrong decidability proof was published). Finally, it was shown in [12] that boundedness (and structural boundedness) is undecidable for reset Petri nets. The proof in [12] was done by a complex reduction from Hilbert's 10th problem (a simpler proof was later given in [11,13]).

Here we generalize these results by using lossy counter machines. This also gives a unified framework and considerably simplifies the proofs.

Lemma 17 *Reset Petri nets can simulate the infinite runs of lossy counter machines with reset-lossiness.*

PROOF. *For every n-counter LCM M (with the reset lossiness relation* $\stackrel{rl}{\rightarrow}$*) we construct a reset Petri net N in the following way: Let there be places* $c_1, \ldots, c_n$ *that hold the contents of the counters and a place q for every state* $q \in Q$ *of the finite control of M. Every marking of this net N where exactly one of the places q contains exactly one token corresponds to a configuration of the counter machine M and vice versa. For every instruction of M of the form* $(q: \ c_i := c_i + 1; \ \mathsf{goto}\ q')$ *with* $(1 \leq i \leq n)$ *there is a transition that takes one token from q, puts one token on* c_i*, puts one token on* q' *and resets all places except* $q', c_1, \ldots, c_n$*. The firing of this transition exactly simulates the computation step of M. For every instruction of M of the form* $(q: \ \mathsf{If}\ c_i = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else}\ c_i := c_i - 1; \mathsf{goto}\ q'')$ *with* $(1 \leq i \leq n)$ *there are two transitions: The first transition takes a token from q, puts a token on* q' *and resets* c_i *and all places except* $q', c_1, \ldots, c_n$*. Instead of being tested for zero the place/counter* c_i *is reset to zero. If the place/counter* c_i *actually was zero before the transition fired, then this was a faithful simulation of the computation step of the counter machine. If the place/counter* c_i *was not zero before, then it was still a faithful simulation of a computation step of the* reset-lossy *counter machine, because* c_i *could suddenly have become zero (empty) by lossiness (see the Def. 2 of reset lossiness* $\stackrel{rl}{\rightarrow}$*). The second transition takes one token from q and one from* c_i*, puts one token on* q'' *and resets all places except* $q'', c_1, \ldots, c_n$*. This transition can only fire if* c_i *is not zero (empty) and faithfully simulates the computation step of the counter machine.*

The only problem with this simulation is that it is possible that in N all tokens on the places q are lost. This causes a deadlock in N. The same thing cannot happen in M, because the finite-control cannot be lost. Thus, N simulates M with some extra deadlocks. However, we still have that

- *For every infinite run of M there is an infinite run of N that faithfully simulates it.*
- *For every infinite run of N there is an infinite run of M that faithfully simulates it.*

Thus, the reset net N faithfully simulates all infinite runs of M. □

Theorem 18 *Structural termination, boundedness and structural boundedness are undecidable for lossy reset Petri nets with every subclassic lossiness relation.*

PROOF. *It follows from Lemma 17 that a lossy reset Petri net with subclassic lossiness relation $\stackrel{l}{\rightarrow}$ can simulate the infinite runs of a lossy counter machine with lossiness relation $\stackrel{l}{\rightarrow} \cup \stackrel{rl}{\rightarrow}$. The results follow from Theorem 12, Theorem 13 and Theorem 15.* $\square$

The undecidability result on structural termination carries over to transfer nets (instead of a reset the tokens are moved to a special 'dead' place), but the others don't. For example, boundedness is decidable for transfer nets [12]. Note that for normal Petri nets structural termination and structural boundedness can be decided in polynomial time (just check if there is a positive linear combination of effects of transitions).

Theorem 16 and Theorem 18 also hold for arbitrary lossiness relations instead of just subclassic ones, but this requires an additional argument. When a Petri net (weakly) simulates a lossy counter machine (e.g., like in Lemma 17) then special places are used to encode the finite-control. If the lossiness relation on the Petri net is not subclassic then the simulated control-state could change by lossiness. This is a problem for lossy counter machines, because (by using the 'capacity' in c_4) one wants to make sure that lossiness cannot occur infinitely often. But now it can happen again as follows:

$$q: \ c_1 := c_1 + 1; \ \mathsf{goto}\ q'$$

By lossiness the control-state could change from q' back to q while the counter c_1 is decreased by 1. The result is an infinite loop at q where c_1 stays at the same value.

On can get around this problem by using the special features of Petri nets. Petri nets (unlike counter machines) can increase a place/counter and decrease another in the same step. So, instead of decreasing the capacity and increasing a counter in the next step (like in Theorem 10) we can do both in one step with one transition. This solves the problem, because now the sum of all places never increases, not even temporarily as in lossy counter machines. Then the proofs of Theorem 16 and Theorem 18 carry over to all lossiness relations.

5.4 Parameterized Problems

We consider verification problems for systems whose definition includes a parameter $n \in \mathbb{N}$. Intuitively, n can be seen as the size of the system. Examples are

- Systems of n indistinguishable communicating finite-state processes.
- Systems of communicating pushdown automata with n-bounded stack.

- Systems of (a fixed number of) processes who communicate through (lossy) buffers or queues of size n.

Let $P(n)$ be such a system with parameter n. For every fixed n, $P(n)$ is a system with finitely many states and thus (almost) every verification problem is decidable for it. So the problem $P(n) \models \Phi$ is decidable for any temporal logic formula Φ from any reasonable temporal logic, e.g., modal μ-calculus [21] or monadic second-order theory. The *parameterized verification problem* is if a property holds independently of the parameter n, i.e., for any size. Formally, the question is if for given P and Φ we have $\forall n \in \mathbb{N}.\, P(n) \models \Phi$ (or $\neg\exists n \in \mathbb{N}.\, P(n) \models \neg\Phi$). Many of these parameterized problems are undecidable by the following meta-theorem.

Theorem 19 *A parameterized verification problem is undecidable if it satisfies the following conditions:*

(1) It can encode an n-space-bounded lossy counter machine (for some lossiness relation) in such a way that $P(n)$ corresponds to the initial configuration with n in one counter and 0 in the others.

(2) It can check for the existence of an infinite run.

PROOF. *By a reduction of $\exists nLCM^{\omega}$ and Theorem 10. The important point here is that in the problem $\exists nLCM^{\omega}$ one can require that the LCM is input-bounded.* □

The technique of Theorem 19 is used in [16] to show the undecidability of the fairness problem for broadcast communication protocols. These are systems of n indistinguishable communicating finite-state processes. The rules for communication are as follows:

(1) Two processes can communicate directly by handshake.

(2) One process can broadcast a message, which is received (immediately) by all other $n-1$ processes.

Every message sent or received by a process can change its internal state, which in turn defines what actions it can perform and how it reacts to messages. The rules for communication are defined independently from the number n of processes in the system. If one considers processes with k internal states then any configuration of the broadcast protocol with n processes can be described by a tuple $(m_1, m_2, \ldots, m_k)$ where m_i is the number of processes in state i and $\sum_{j=1}^{k} m_j = n$. Every such m_i can be seen as the content of a counter which is bounded by n. A broadcast can cause all processes in a certain state to change to another state. This can be used to reset such a simulated space-bounded counter to zero. Note however, that no test for zero is possible. The problem if such a broadcast protocol terminates (i.e., for every number n of processes the system terminates) is undecidable, because it satisfies the conditions of Theorem 19 (the lossiness relation used here is reset-lossiness). Thus all fairness properties, like those expressible in the temporal logics CTL [9,14]) and LTL (linear-time temporal logic [28]), are undecidable as well.

In the same way, similar results can be proved for parameterized problems about systems with bounded buffers, stacks, etc.

6 Extensions

The proofs of the main undecidability results in Theorem 10 and Theorem 12 work only for LCM with at least 4 counters. The question arises, if fewer counters suffice, like the two counters used in normal counter machines. However, the methods used to reduce the number of counters in normal counter machines do not carry over to LCM. They use codings which are not robust under lossiness. Also these codings require a lot of computation and some types of LCM are not exactly Turing-powerful. The decidability of structural termination for LCM with 3 or less counters probably depends on the particular lossiness relation.

The computational power of lossy counter machines also depends very much on the particular lossiness relation. However, a few general observations can be made. The utmost one can expect from a LCM is the following:

- There is at least one computation that gives the correct result, since $id \subseteq \xrightarrow{l}$.
- There may be other computations that give results that are smaller than the correct result (by the definition of lossiness).

For some operations, e.g., addition and multiplication, this optimal behavior can be achieved. However, for other operations like subtraction it is impossible, since the obtained result may even be larger than the correct one. In fact, many versions of LCM cannot even compare two numbers. Thus, it should be stressed that we do not advocate LCM as a model of computation, but rather as a means of proving undecidability.

Another question is if the undecidability results can be extended to more general lossiness relations than $\xrightarrow{s}$ (see Def. 2). (Even $\xrightarrow{s}$ can hardly be called lossiness any more, since it allows some counters to increase while others decrease.) One idea is to introduce functions $f : \mathbb{N}^n \mapsto \mathbb{N}$ s.t. if $s \xrightarrow{l} s'$ then either $s = s'$ or $f(s') < f(s)$. (In the case of $\xrightarrow{s}$ the function f is the sum.) Again this depends very much on the lossiness relation relation $\xrightarrow{l}$. In the proof of Theorem 10 a balance must be kept in the 4th counter, to ensure that the LCM is input-bounded and lossiness can occur only finitely often in the infinite run. This balance must be updated (computed) on the lossy counter machine, which is not always Turing-powerful. In the simple case of the 'sum' function this is trivial, but for more general functions f it is a problem.

7 Conclusion

Lossy counter machines can be used as a general tool to show the undecidability of many problems. It provides a unified way of reasoning about many quite different classes of systems. For example the recurrent-state problem for

lossy fifo-channel systems, the boundedness problem for reset Petri nets and the fairness problem for broadcast communication protocols were previously thought to be completely unrelated. Yet lossy counter machines show that the principles behind their undecidability are the same. Moreover, the undecidability proofs for lossy counter machines are very short and much simpler than previous proofs of weaker results [3,12].

Lossy counter machines have also been used in this paper to show that even for very weak temporal logics and extremely weak models of infinite-state concurrent systems, the model checking problem is undecidable (see Subsection 5.2). We expect that many more problems can be shown to be undecidable with the help of lossy counter machines, especially in the area of parameterized problems (see Subsection 5.4).

Acknowledgments: Thanks to Javier Esparza and Petr Jančar for fruitful discussions and to an anonymous referee for detailed comments.

References

[1] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *10th Intern. Conf. on Computer Aided Verification (CAV'98)*. LNCS 1427, 1998.

[2] P. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS'93*. IEEE, 1993.

[3] P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.

[4] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. In *Proc. of ICALP'97*, volume 1256 of *LNCS*, 1997.

[5] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. of STACS'99*, volume 1563 of *LNCS*. Springer Verlag, 1999.

[6] G. Cécé, A. Finkel, and S.P. Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[7] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[8] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[9] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. volume 131 of *LNCS*, pages 52–71, 1981.

[10] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.

[11] C. Dufourd. *Réseaux de Petri avec Reset/Transfert: Décidabilité et indécidabilité*. PhD thesis, ENS de Cachan, 1998.

[12] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[13] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T Nets. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.

[14] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Volume B, FORMAL MODELS AND SEMANTICS*. Elsevier, 1994.

[15] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[16] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. of LICS'99*. IEEE, 1999.

[17] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[18] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. volume 85 of *LNCS*, pages 295–309, 1980.

[19] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of Association of Computer Machinery*, 32:137–162, 1985.

[20] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.

[21] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

[22] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460, 1984.

[23] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[24] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP 2000*, volume 1853 of *LNCS*. Springer Verlag, 2000.

[25] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[26] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[27] J.L. Peterson. *Petri net theory and the modeling of systems.* Prentice-Hall, 1981.

[28] A. Pnueli. The temporal logic of programs. In *FOCS'77.* IEEE, 1977.

[29] K. Ruohonen. On some variants of Post's correspondence problem. *Acta Informatica*, 19:357–367, 1983.

[30] H. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96(1):119–137, 1992.

On the Verification of Broadcast Protocols

Javier Esparza * Alain Finkel† Richard Mayr‡

Abstract

We analyze the model-checking problems for safety and liveness properties in parameterized broadcast protocols, a model introduced in [5]. We show that the procedure suggested in [5] for safety properties may not terminate, whereas termination is guaranteed for the procedure of [1] based on upward closed sets. We show that the model-checking problem for liveness properties is undecidable. In fact, even the problem of deciding if a broadcast protocol may exhibit an infinite behavior is undecidable.

1. Introduction

In [5], Emerson and Namjoshi present an abstract reachability procedure —called the EN-procedure in the sequel— for the construction of a "covering graph". It generalizes the Karp-Miller construction of a covering graph for Petri nets [9]. The EN-procedure can be applied to classes of systems satisfying some abstract conditions (essentially, computability of the least upper bounds of certain chains). By combining it with the automata-theoretic approach to model-checking [11], Emerson and Namjoshi show that it can be used to verify safety and liveness properties. Similar constructions have been studied in the framework of well-structured transition systems [6].

The termination of the EN-procedure depends on the class of systems being considered. In [5] termination is proved for the parameterized systems of [4, 8]; termination for Petri nets and vector addition systems was already proved in [9].

In the case of parameterized systems, the EN-procedure can be used to prove that a property holds *independently* of the number of processes participating in the protocol. In other words, it can show that *all* the elements of an infinite family of finite-state systems satisfy a certain property.

*Institut für Informatik, Technische Universität München, D-80290 München, Germany. E-mail: esparza@in.tum.de

†Lab. Spécification et Vérification, ENS de Cachan, 61, av. du Président Wilson, 94235 Cachan Cedex, France. E-mail: finkel@lsv.ens-cachan.fr

‡LFCS, Dept. of Computer Science, Univ. of Edinburgh, Edinburgh EH9 3JZ, UK. E-mail: mayrri@dcs.ed.ac.uk

One of the most interesting points of [5] is the application of the EN-procedure to a new parameterized model called parameterized broadcast protocols—shortened to *broadcast protocols* in the sequel. Broadcast protocols are systems composed of a finite but arbitrarily large number of indistinguishable processes that communicate by rendezvous (two processes exchange a message) or by broadcasts (a process sends a message to all other processes). It is also possible to incorporate a distinguished control process. While the case in which processes communicate only by rendezvous had already been studied in [8, 4], the extension to broadcasts is considered in [5] for the first time. The addition of broadcasts allows to model simplified versions of cache coherence protocols like MESI-protocols.

In [5] it is shown that broadcast protocols satisfy the abstract conditions necessary for the applicability of the EN-procedure. However, neither the termination issue nor the decidability of the model-checking problems for safety and liveness properties are examined. In this paper we address these points and obtain the following results:

- The EN-procedure may not terminate for broadcast protocols.
- The model-checking problem for safety properties is decidable. The decision procedure—which obviously cannot be the EN-procedure—is the result of instantiating for broadcast protocols an abstract backwards reachability algorithm introduced in [1].
- The model-checking problem for liveness properties is undecidable.

The paper is organized as follows: Section 2 introduces broadcast protocols and formalizes the model-checking problems for safety and liveness properties. Sections 3, 4, 5 present the results above, respectively.

2. Broadcast Protocols: Basic Definitions

2.1. Syntax

A *broadcast protocol* is a triple (S, L, R) where S is a finite set of *states*. L is a finite set of *labels* composed of: a set Σ_l

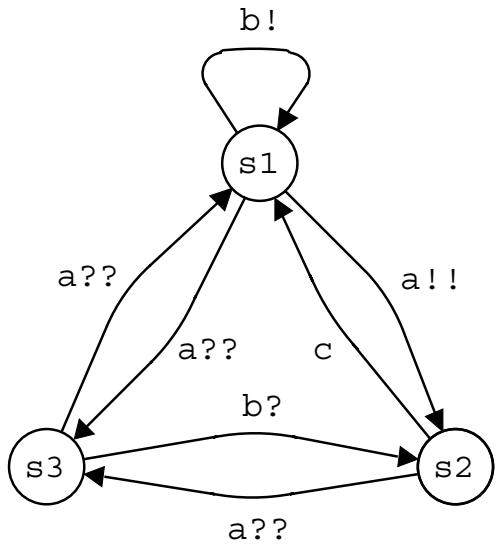


Figure 1. A broadcast protocol

of *local* labels, two sets $\Sigma_r \times \{?\}$ and $\Sigma_r \times \{!\}$ of *input* and *output rendez-vous* labels, and two sets $\Sigma_b \times \{??\}$ and $\Sigma_b \times \{!!\}$ of *input* and *output broadcast* labels, where $\Sigma_l, \Sigma_r, \Sigma_b$ are disjoint finite sets.

Along the paper $a, b, c, \ldots$ denote elements of $\Sigma = \Sigma_l \cup \Sigma_r \cup \Sigma_b$. Rendezvous and broadcast labels like $(a, ?)$ or $(b, !!)$ are shortened to $a?$ and $b!!$. Elements of Σ are called *actions*. $R \subseteq S \times L \times S$ is a set of *transitions* satisfying the following property: for every $a \in \Sigma_b$ and every state $s \in S$, there exists a state $s' \in S$ such that $s \xrightarrow{a??} s'$. Intuitively, this condition guarantees that a process is always willing to receive a broadcasted message. We represent broadcast protocols graphically as shown in Figure 1.

In this paper we consider broadcast protocols satisfying the following additional constraints. (a) For each state s and each broadcast label $a??$ there is exactly one state s' such that $s \xrightarrow{a??} s'$ (determinism). (b) Each label of the form a, $a!$, $a?$ and $a!!$ appears in exactly one transition.

These constraints are only used to simplify the presentation. All our decidability/undecidability results are valid for general broadcast protocols.

2.2. Semantics

Let $B = (S, L, R)$ be a broadcast protocol where $S = \{s_1, \ldots, s_n\}$. A *configuration* of B is a function $\mathbf{c}\colon S \to \mathbb{N}$. Intuitively, $\mathbf{c}(s_i)$ indicates how many processes are in the state s_i. We identify $\mathbf{c}$ with the vector $(\mathbf{c}(s_1), \ldots, \mathbf{c}(s_n)) \in \mathbb{N}^n$. We denote by $\mathbf{u}_i$ the configuration given by $\mathbf{u}_i(s_j) = 1$ if $i = j$ and $\mathbf{u}_i(s_j) = 0$ otherwise. Moves between configurations are either rendezvous (two processes exchange a message and move to new states) or broadcasts (a process sends a message to all other processes; all processes move to new states). The semantics of B is the smallest subset of $\mathbb{N}^n \times \Sigma \times \mathbb{N}^n$ satisfying the three conditions below, where a triple $(\mathbf{c}, a, \mathbf{c}') \in \mathbb{N}^n \times \Sigma \times \mathbb{N}^n$ is denoted by $\mathbf{c} \xrightarrow{a} \mathbf{c}'$.

- If $s_i \xrightarrow{a} s_j$ then $\mathbf{c} \xrightarrow{a} \mathbf{c}'$ for every $\mathbf{c}$, $\mathbf{c}'$ such that $\mathbf{c}(s_i) > 0$ and $\mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j$.
  I.e. one process is removed from s_i, and one process is added to s_j.

- If $s_i \xrightarrow{a!} s_j$ and $s_k \xrightarrow{a?} s_l$ then $\mathbf{c} \xrightarrow{a} \mathbf{c}'$ for every $\mathbf{c}$, $\mathbf{c}'$ such that $\mathbf{c}(s_i) > 0$, $\mathbf{c}(s_k) > 0$ and $\mathbf{c}' = \mathbf{c} - \mathbf{u}_i - \mathbf{u}_k + \mathbf{u}_j + \mathbf{u}_l$.
  I.e. one process is removed from s_i and s_k, and one process is added to s_j and s_l.

- If $s_i \xrightarrow{a!!} s_j$ then $\mathbf{c} \xrightarrow{a} \mathbf{c}'$ for every $\mathbf{c}$, $\mathbf{c}'$ such that $\mathbf{c}(s_i) > 0$ and $\mathbf{c}'$ can be computed from $\mathbf{c}$ in the following three steps:

$$\mathbf{c}_1 = \mathbf{c} - \mathbf{u}_i \quad (1)$$

$$\mathbf{c}_2(s_k) = \sum_{\{s_l \mid s_l \xrightarrow{a??} s_k\}} \mathbf{c}_1(s_l) \quad (2)$$

$$\mathbf{c}' = \mathbf{c}_2 + \mathbf{u}_j \quad (3)$$

  I.e. the sending process leaves s_i (1), all other processes receive the broadcast and move to their destinations (2), and the sending process reaches s_j (3).

Thanks to our constraints (a) and (b) above, the configuration $\mathbf{c}'$ is completely determined by $\mathbf{c}$ and the action a. In the example of Figure 1 we have

$$(3,1,2) \xrightarrow{c} (4,0,2)$$
$$(3,1,2) \xrightarrow{b} (3,2,1)$$
$$(3,1,2) \xrightarrow{a} (2,1,3)$$

Given a broadcast protocol with n states, we call the $n \times n$ matrices having unit vectors as columns *broadcast matrices* [5]. Given an action $a \in \Sigma$, it is easy to see that there exists a broadcast matrix M_a and a vector $\mathbf{v}_a$ such that $\mathbf{c}' = M_a \cdot \mathbf{c} + \mathbf{v}_a$ holds whenever $\mathbf{c} \xrightarrow{a} \mathbf{c}'$. For example, for the action a in the example of Figure 1 we have

$$M_a = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \quad \mathbf{v}_a = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

Since broadcast matrices are closed under product, this observation can be generalized to arbitrary sequences $\sigma \in \Sigma^*$: If $\mathbf{c} \xrightarrow{\sigma} \mathbf{c}'$ then $\mathbf{c}' = M_\sigma \cdot \mathbf{c} + \mathbf{v}_\sigma$ for some broadcast matrix M_σ and vector $\mathbf{v}_\sigma$.

The *language* of B from an initial configuration $\mathbf{c}_0$, denoted by $L(B, \mathbf{c}_0)$, is the set of sequences $\sigma \in \Sigma^*$ such that $\mathbf{c}_0 \xrightarrow{\sigma} \mathbf{c}$ for some configuration $\mathbf{c}_0$. The ω-language of B from $\mathbf{c}_0$, denoted by $L_\omega(B, \mathbf{c}_0)$, is defined accordingly.

A *parameterized configuration* is a *partial* function $\mathbf{p} \colon S \to \mathbb{N}$. We identify it with a set of configurations, namely those extending $\mathbf{p}$ to a total function. So we identify the parameterized configuration of the broadcast protocol of Figure 1 given by $\mathbf{p}(s_1) = \mathbf{p}(s_2) = \perp$ (undefined) and $\mathbf{p}(s_3) = 3$ with the set of configurations $\{(n_1, n_2, 3) \mid n_1, n_2 \in \mathbb{N}\}$.

The *language* of B from an initial parameterized configuration $\mathbf{p}_0$, denoted by $L(B, \mathbf{p}_0)$, is defined as

$$L(B, \mathbf{p}_0) = \bigcup_{\mathbf{c} \in \mathbf{p}_0} L(B, \mathbf{c})$$

So $L(B, \mathbf{p}_0)$ contains all sequences of actions that the protocol can execute from all initial configurations that belong to the initial parameterized configuration $\mathbf{p}_0$. $L_\omega(B, \mathbf{p}_0)$ is defined analogously.

2.3. Model-Checking Problems

Following the automata-theoretic approach to model-checking (see for instance [11]), we formalize a linear safety property as a regular set of dangerous sequences of actions the protocol should *not* engage in. Similarly, a liveness property is formalized as an ω-regular language over Σ.

Notice that we consider languages over Σ, corresponding to properties on the *actions* of the system. In [5] properties on the *configurations* satisfying certain conditions are considered instead; for that, configurations are labeled with atomic properties. All the results of this paper hold for the languages of [5] as well.

We study the decidability of the following two model-checking problems:

> **Safety properties**
> Given: a broadcast protocol B, a parameterized configuration $\mathbf{p}_0$, a regular language L.
> To decide: if $L(B, \mathbf{p}_0) \cap L = \emptyset$.

> **Liveness properties**
> Given: a broadcast protocol B, a parameterized configuration $\mathbf{p}_0$, an ω-regular language L.
> To decide: if $L(B, \mathbf{p}_0) \cap L = \emptyset$.

These two problems can be approached using well-known automata-theoretic techniques. For the safety problem, we take a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ accepting the language L. The *combined system* of a protocol B with n states and an automaton A is a subset of $(\mathbb{N}^n \times Q) \times \Sigma \times (\mathbb{N}^n \times Q)$ defined by: $(\mathbf{c}, q) \xrightarrow{a} (\mathbf{c}', q')$ if and only if $\mathbf{c} \xrightarrow{a} \mathbf{c}'$ in B and $q \xrightarrow{a} q'$ in A. Clearly, $L(B, \mathbf{p}_0) \cap L = \emptyset$ if and only if no path of the combined system starting at any $(\mathbf{c}, q_0)$, where $\mathbf{c} \in \mathbf{p}_0$, ever visits a combined state of the form $(\mathbf{c}', q)$ where $q \in F$. For the liveness problem we replace A by a Büchi automaton, and 'visits a state' by 'visits a state infinitely often'.

3. The EN-Procedure may not Terminate

The EN-procedure for the construction of the covering graph is described below. We exhibit a broadcast protocol for which it does not terminate. It is then straightforward to show that the procedure may not terminate either for combined systems.

Fix for the rest of this section a broadcast protocol $B = (S, L, R)$, where $S = \{s_1, \ldots, s_n\}$, and a parameterized initial configuration $\mathbf{p}_0$.

Let $(\mathbb{N} \cup \{\omega\})^n$ be the set of *ω-configurations* of B. The semantics of broadcast protocols is generalized to ω-configurations by letting $\omega + n = \omega - n = \omega$ for all $n \in \mathbb{N}$, $\omega + \omega = \omega$ and $\omega - \omega = 0$. Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be ω-configurations. We say $\mathbf{e}_1 \preceq \mathbf{e}_2$ if $\mathbf{e}_1$ is pointwise smaller than or equal to $\mathbf{e}_2$, where $n \leq \omega$ for every $n \in \mathbb{N} \cup \{\omega\}$. Clearly, $\preceq$ is a complete partial order on ω-configurations The least upper bound (*lub*) of a chain is the vector of *lubs* of the component chains. For a sequence of actions σ, define T_σ as the affine operator given by $T_\sigma(\mathbf{e}) = M_\sigma(\mathbf{e}) + \mathbf{v}_\sigma$.

The EN-procedure examines pairs $(\mathbf{e}, a)$, where $\mathbf{e}$ is an ω-configuration and $a \in \Sigma$. It is initialized with the empty graph and the set of unexamined pairs $\{(\mathbf{e}_0, a) \mid a \in \Sigma\}$, where $\mathbf{e}_0$ is defined by

$$\mathbf{e}_0(s_i) = \begin{cases} \mathbf{p}_0(s_i) & \text{if } \mathbf{p}_0(s_i) \text{ defined} \\ \omega & \text{otherwise.} \end{cases}$$

The procedure goes as follows:

0. Add the node $\mathbf{e}_0$ to the graph.
1. Choose an unexamined pair $(\mathbf{e}, a)$; if there are none, stop.
2. If there is no $\mathbf{e}'$ such that $\mathbf{e} \xrightarrow{a} \mathbf{e}'$, then mark $(\mathbf{e}, a)$ as examined and go to 1.
3. If $\mathbf{e} \xrightarrow{a} \mathbf{e}'$ for some $\mathbf{e}'$ then
   3.1 If the graph contains an ω-configuration $\mathbf{d} \succeq \mathbf{e}'$ then make $\mathbf{d}$ the a-successor of $\mathbf{e}$;
   3.2 else, if the graph contains a path from some node $\mathbf{d}$ to $\mathbf{e}$ such that $\mathbf{d} \preceq \mathbf{e}'$, then let σ be the sequence of actions of this path, let $\mathbf{l}$ be the *lub* of the chain

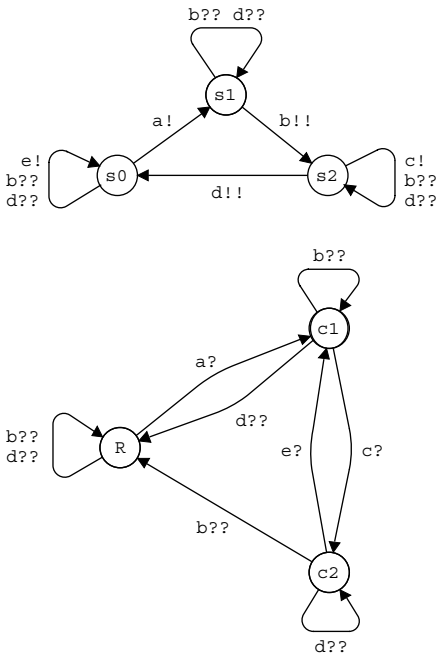


Figure 2. A protocol with an infinite covering graph

$\mathbf{d} \preceq T_{\sigma a}(\mathbf{d}) \preceq T^2_{\sigma a}(\mathbf{d}) \preceq \ldots$,
and make $\mathbf{l}$ the a-successor of $\mathbf{e}$;

3.3 else, create $\mathbf{e}'$ as the a-successor of $\mathbf{e}$.

4. Mark $(\mathbf{e}, a)$ as examined and go to 1.

Two questions arise: (a) is the *lub* of a chain effectively computable? and, (b) does the procedure terminate, i.e., is the covering graph finite? In [5], Emerson and Namjoshi answer (a) positively (this is essentially a consequence of the fact that there are only finitely many broadcast matrices for a given n), but they do not study (b). We present an example, inspired by [3], showing that the procedure may not terminate.

Consider the broadcast protocol B of Figure 2. Initially there is a process in state s_0 and arbitrarily many processes in state R. Following the terminology of [4, 8], the example consists of a *control process*, which is always in one of the states s_0, s_1, s_2, and an arbitrary number of identical *user processes*, initially in state R. The protocol simulates a machine operating on two counters modeled by the states c_1 and c_2, which draw their items from a repository, modeled by state R. The meaning of the different actions is:

- a: add 1 to c_1;
- b: reset c_2 to 0;
- c: transfer one item from c_1 to c_2;
- d: reset c_1 to 0;
- e: transfer one item from c_2 to c_1.

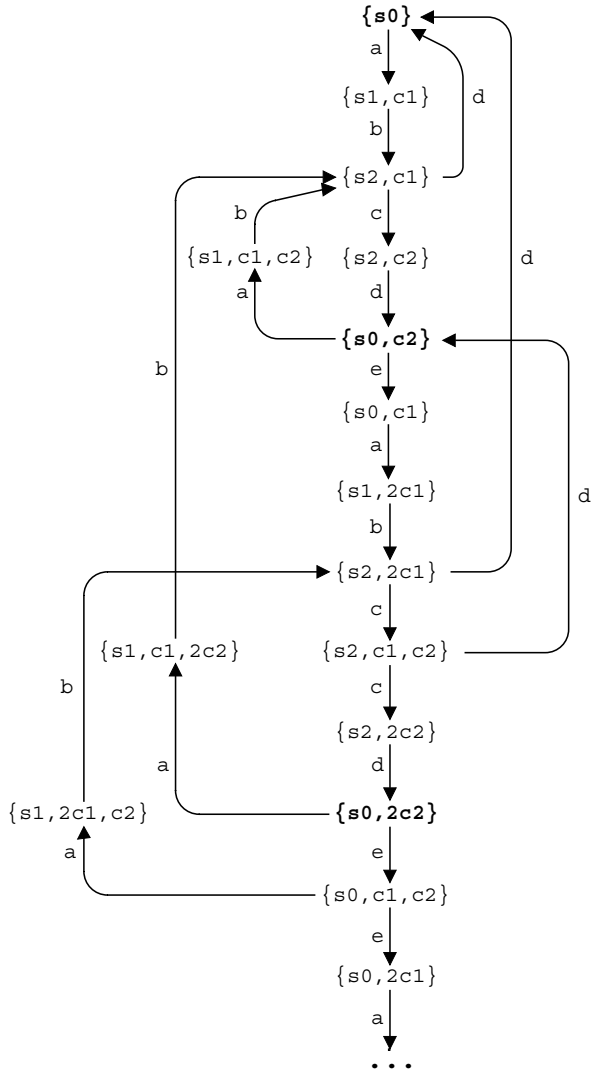


Figure 3. Semantics of the protocol of Figure 2

We construct the covering graph from $\mathbf{e}_0$, the ω-configuration putting 1 process in s_0, ω processes in R, and 0 processes elsewhere. We use a multiset notation for ω-configurations; for example, $\{s_2, 3c_1\}$ denotes the ω-configuration putting one process in s_2, 3 processes in c_1, and ω processes in R. Notice that every ω-configuration reachable from $\mathbf{e}_0$ puts ω processes in R, and so we omit this part. With this notation we have $\mathbf{e}_0 = \{s_0\}$. An initial part of the configurations of B reachable from $\mathbf{e}_0$ is shown in Figure 3. Notice that the sequence of actions $abcdeabc^2de^2abc^3de^3 \cdots abc^nde^n \cdots$ can be executed from $\mathbf{e}_0$, and that all the ω-configurations reached along this sequence are different. So, in particular, there are infinitely many reachable configurations from e_0.

Proposition 3.1 *The covering graph for the broadcast protocol of Figure 2 and the ω-configuration $\mathbf{e}_0 = \{s_0\}$ is infinite.*

Proof: Let $\sigma\tau$ be an arbitrary sequence of actions such that $\mathbf{e}_0 \xrightarrow{\sigma} \mathbf{e}_1 \xrightarrow{\tau} \mathbf{e}_2$, $\mathbf{e}_1 \preceq \mathbf{e}_2$, and $\mathbf{e}_1 \neq \mathbf{e}_2$. Since in every configuration reachable from $\mathbf{e}_0$ the total number of processes in the states s_1, s_2, s_3 is 1, both $\mathbf{e}_1$ and $\mathbf{e}_2$ coincide on these states. Since $\mathbf{e}_1 \neq \mathbf{e}_2$, τ contains at least one occurrence of b and d. Assume that the last occurrence of b precedes the last occurrence of d (the other case is similar). Then, τ has the form $\tau_1 b \tau_2 d \tau_3$, where $\tau_2\tau_3$ contains no b's and τ_3 contains no d's.

For the construction of the covering graph we can replace $\mathbf{e}_2$ by the *lub* of the chain $\mathbf{e}_1 \preceq \mathbf{e}_2 \preceq \mathbf{e}_3 \cdots$ where $\mathbf{e}_i = T_\tau^{i-1}(\mathbf{e}_1)$. We prove that $\mathbf{e}_i = \mathbf{e}_2$ for every $i \geq 2$, which implies that the *lub* is $\mathbf{e}_2$. This shows that for the protocol of Figure 2 the EN-procedure and the EN-procedure without step 3.2 compute the same graph. Since the latter computes an infinite graph, the covering graph is infinite.

To show $\mathbf{e}_i = \mathbf{e}_2$, we observe that, since $\mathbf{e}_2$ and $\mathbf{e}_i$ coincide on the states s_1, s_2, s_3 and R, it suffices to prove $\mathbf{e}_2(c_1) = \mathbf{e}_i(c_1)$ and $\mathbf{e}_2(c_2) = \mathbf{e}_i(c_2)$. We prove $\mathbf{e}_2(c_1) = \mathbf{e}_i(c_1)$, the other case being similar.

By the definition of T_τ, we have $\mathbf{e}_1 \xrightarrow{\tau} \mathbf{e}_2 \xrightarrow{\tau^{i-2}} \mathbf{e}_i$ for every $i \geq 2$. Since the occurrence of d removes all processes from c_1, $\mathbf{e}_2(c_1)$ and $\mathbf{e}_i(c_1)$ are determined by the suffix of τ and τ^{i-1} starting right after the last occurrence of d. This suffix is τ_3 in the two cases, and so we have $\mathbf{e}_2(c_1) = \#(\tau_3, a) + \#(\tau_3, e) - \#(\tau_3, c) = \mathbf{e}_i(c_1)$, where $\#$ denotes the number of occurrences of an action in a sequence. ■

Since the protocol of Figure 2 contains both broadcast and rendezvous actions, the EN-procedure might still terminate for broadcast protocols with only broadcast moves. Unfortunately, this is not the case. To prove it, given a broadcast protocol $B = (S, L, R)$, we define the broadcast protocol $Exp(B)$ (*expansion* of B) as the result of performing the following two operations:

- each transition $s \xrightarrow{a} s'$ where a is a local action is replaced by the transition $s \xrightarrow{a!!} s'$, and a transition $t \xrightarrow{a??} t$ is added for each state t;
- each pair of transitions $s \xrightarrow{a!} s'$ and $t \xrightarrow{a?} t'$, where a is a rendezvous action, is replaced by the construction shown in Figure 4.[1] Moreover, in order to make sure that for each state s there is a state s' such that $s \xrightarrow{a_1??} s'$ and $s \xrightarrow{a_2??} s'$, transitions $s \xrightarrow{a_1??} s$ and $s \xrightarrow{a_2??} s$ are added where needed.

[1] The construction introduces two new states per rendezvous.

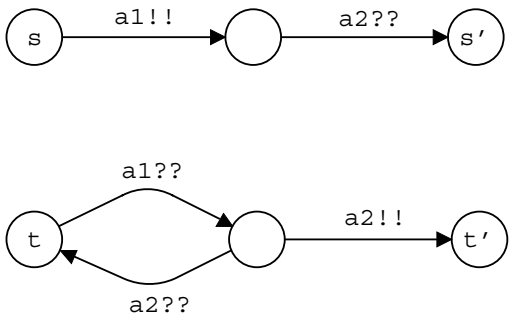


Figure 4. Simulation of a rendezvous by broadcasts

Observe that $Exp(B)$ only contains broadcast actions. We also define a morphism φ between the action sequences of B and $Exp(B)$ as follows: $\varphi(a) = a_1 a_2$ if a is a rendezvous action, and $\varphi(a) = a$ otherwise.

It is immediate to see that if $\mathbf{c} \xrightarrow{\sigma} \mathbf{c}'$ in B, then $\mathbf{c} \xrightarrow{\varphi(\sigma)} \mathbf{c}'$ in $Exp(B)$, and vice versa. Here we interpret $\mathbf{c}$ as the configuration of $Exp(B)$ that coincides with $\mathbf{c}$ on the states of B and puts no process in the new states. We now have:

Proposition 3.2 *Let B the broadcast protocol shown in Figure 2. The covering graph for the protocol $Exp(B)$ and the ω-configuration $\{s_0\}$ is infinite.*

Proof: The sequence

$$\varphi(abcdeabc^2de^2abc^3de^3 \cdots abc^n de^n \cdots)$$

can be executed from $\mathbf{e}_0$ in $Exp(B)$, and all the ω-configurations reached along this sequence are different. So there are infinitely many reachable configurations from $\mathbf{e}_0$ in $Exp(B)$. The argument used in the proof of Proposition 3.1, namely that every sequence τ must contain occurrences of b and d, is still valid, and in fact the proof can be carried out in the same way. ■

The Exp construction also leads to the following result:

Proposition 3.3 *The safety and liveness problems for arbitrary protocols can be reduced to the same problems for broadcast protocols with only broadcast actions.*

Proof: Given an arbitrary protocol B and a regular or ω-regular language L, we have $L(B, \mathbf{p}_0) \cap L = \emptyset$ if and only if $L(Exp(B), \mathbf{p}_0) \cap \varphi(L) = \emptyset$. ■

We finish this section with a small remark. It was shown in [8] that non-broadcast protocols with a control process and arbitrarily many user processes are more complicated to

analyze than those in which all processes are identical. So one could ask if this is also the case for broadcast protocols. The answer is no. We can easily simulate the protocol of Figure 2 by another one in which all processes are identical: It suffices to add a new state *Init* and two new transitions $Init \xrightarrow{init!!} s_0$ and $Init \xrightarrow{init??} R$, and put all processes initially in the *Init* state. The new protocol must first do an *init*, by which essentially a process tells the others that it becomes the control process and the others become user processes.

4. A Model-Checking Algorithm for Safety Properties

Let B be a broadcast protocol with states $S = \{s_1, \ldots, s_n\}$ and a parameterized initial configuration $\mathbf{p}_0$, and let $A = (Q, \Sigma, \delta, q_0, F)$ be an automaton. The model-checking problem for safety properties can be reformulated as follows: Can some combined state $\mathbf{n} \in \mathbb{N}^n \times F$ be reached from a combined state $(\mathbf{c}_0, q_0)$ such that $\mathbf{c}_0 \in \mathbf{p}_0$?

We can use this observation to apply a general backwards reachability algorithm presented in [1] (see also [7]), which we "instantiate" for broadcast protocols in the rest of this section. The algorithm constructs the set of predecessors of $\mathbb{N}^n \times F$, and checks whether it has an empty intersection with $(\mathbf{p}_0, q_0)$.

We need some preliminaries. A set C of combined states is *upwards-closed* if $(\mathbf{c}, q) \in C$ implies $(\mathbf{c}', q') \in C$ for every $(\mathbf{c}, q) \trianglelefteq (\mathbf{c}', q')$, where $(\mathbf{c}, q) \trianglelefteq (\mathbf{c}', q')$ if $\mathbf{c} \preceq \mathbf{c}'$ and $q = q'$. Denote by $pred(C)$ the set of *immediate predecessors* of C (i.e. the combined states from which C can be reached in one step). We have the following result:

Proposition 4.1 *Let C be an upwards-closed set of combined states. Then:*

1. *The set of minimal elements of C is finite.*
2. *The set $pred(C)$ is upwards-closed.*
3. *The minimal elements of $pred(C)$ are effectively computable from the minimal elements of C.*

Proof: 1. Follows immediately from the fact that $\trianglelefteq$ is a well-ordering.

2. It suffices to prove that for each action a the set of immediate predecessors of C through the action a is upwards-closed. We do it for the case in which a is a broadcast action, the other cases being simpler. Assume we have $s_1 \xrightarrow{a!!} s_2$. The immediate prodecessors of C through a is the set of combined states $(\mathbf{c}, q)$ such that the following conditions hold for some minimal element $(\mathbf{c}', q')$ of C: (1) $M_a \cdot \mathbf{c} + \mathbf{v}_a \geq \mathbf{c}'$, (2) $\mathbf{c}(s_1) \geq 1$, and (3) $q \xrightarrow{a} q'$. Since M_a is a broadcast matrix, this set is upward-closed.

3. Again, it suffices to prove the result for the set of immediate predecessors of C through the action a. A little algebra shows that the minimal elements of this set are the combined states satisfying (2) and (3) above, plus a new condition (1′) of the form $M_a \cdot \mathbf{c} = \mathbf{d}$, where $\mathbf{d}$ is defined as follows. Since M_a is a broadcast matrix, there is exactly one state s such that $M_a(s, s_1) = 1$. We take $\mathbf{d}(s') = \mathbf{c}'(s') - \mathbf{v}_a(s')$ for every $s' \neq s$, and $\mathbf{d}(s) = \max(1, \mathbf{c}'(s) - \mathbf{v}_a(s))$. The set of solutions of (1′), (2), and (3) is clearly computable. ■

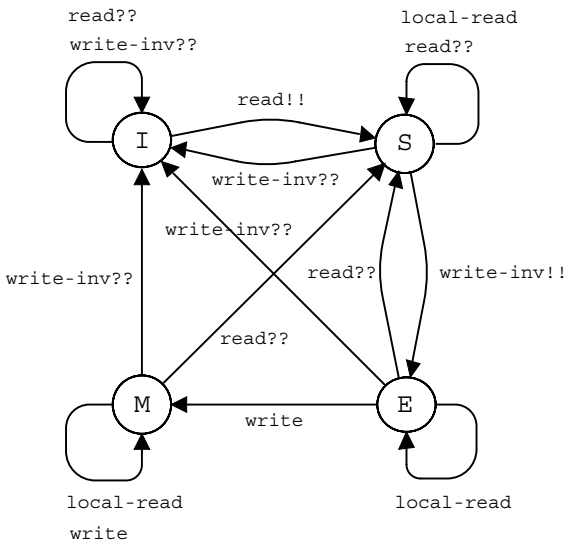


Figure 5. A MESI-protocol

Since $\mathbb{N}^n \times F$ is an upwards-closed set, we can apply Proposition 4.1 and iteratively compute the minimal elements of $C_0 = \mathbb{N}^n \times F$, $C_1 = C_0 \cup pred(\mathbb{N}^n \times F)$, $C_2 = C_1 \cup pred^2(\mathbb{N}^n \times F)$, etc. But we know that in any infinite set of combined states there exist two elements $\mathbf{n}, \mathbf{n}'$ such that $\mathbf{n} \trianglelefteq \mathbf{n}'$ (i.e. $\trianglelefteq$ is a so-called *well-quasi-ordering*). Therefore, there is an n such that the minimal elements of C_n and $pred^*(\mathbb{N}^n \times F) = \bigcup_{i \geq 0} C_i$ coincide, and so the algorithm terminates.

In [5] the EN-procedure is applied to the protocol shown in Figure 5, a simplified version of a MESI-protocol for cache coherence. The initial configuration puts arbitrarily many processes in state I, and none in the other three states. For this particular protocol the EN-procedure terminates and yields a covering graph with four nodes [5]. The invariants $\#M = 0 \vee \#S = 0$ and $\#M + \#E \leq 1$, where $\#s$ denotes the number of processes in the state s, are proved to hold by observing that no node covers a configuration violating the invariants.

We can prove the same two invariants using our algorithm.

For these simple properties we can do without an automaton[2]: It suffices to compute the set of predecessors of the upwards-closed sets $\#M \geq 1 \wedge \#S \geq 1$ and $\#M + \#E \geq 2$, respectively, which we call U and V in the sequel. The reader can easily check that $pred(V) = V$, and so the procedure terminates after one step with $pred^*(V) = V$. For U we have

$$\begin{array}{rl} U\text{:} & \#M \geq 1 \wedge \#S \geq 1 \\ pred(U)\text{:} & (\#M \geq 1 \wedge \#S \geq 1) \vee \\ & (\#M = 0 \wedge \#E = 1 \wedge \#S \geq 1) \\ pred^2(U)\text{:} & pred(U) \end{array}$$

i.e. the procedure terminates after 2 steps. Since the predecessors of U and V do not contain any initial configuration, the invariants hold.

5. The Model-Checking Problem for Liveness Properties is Undecidable

We prove that it is undecidable if $L_\omega(B, \mathbf{p}_0) = \emptyset$, i.e. it is undecidable if the broadcast protocol B with initial parameterized configuration $\mathbf{p}_0$ can execute an infinite sequence. The undecidability of the model-checking problem follows.

The proof is by reduction from a problem on counter machines. It is closely related to the undecidability of a similar problem for lossy counter machines proved in [10] (in fact, it follows as a corollary from the results in [10]), and has been inspired by the undecidability proofs of [2].

We start by introducing some notations and definitions. A *counter machine* is a tuple $M = (Q, C, \Delta, q_0, H)$ where Q is a set of states, C is a set of counters, q_0 is an initial state, H is a set of halting states, and Δ is a set of transitions. Transitions are of three types:

- $q \xrightarrow{c:=c+1} q'$, which increase counter c,
- $q \xrightarrow{c:=c-1} q'$, which decrease counter c; these transitions can only be taken if the counter has a positive value;
- $q \xrightarrow{c=0} q'$, zero-tests that can only occur if the value of the counter is 0.

A *configuration* of $\mathcal{M}$ is a tuple $(q, j_1, \ldots, j_m)$, where q is a state, and $j_1, \ldots, j_m$ are natural numbers indicating the contents of the counters. The semantics of a counter machine is a relation $\rightarrow$ between configurations, defined as expected. A *run* is either an infinite sequence $c_1 \rightarrow c_2 \rightarrow \ldots$ or a finite sequence $c_1 \rightarrow \ldots \rightarrow c_n$ where c_n is halting. A configuration $(q, j_1, \ldots, j_m)$ is *initial* if $q = q_0$, and *n-bounded* if $\sum_{1 \leq i \leq m} j_i \leq n$. A run is *initial* if its first configuration is initial, *n-bounded* if all its configurations contain only n-bounded configurations, and *bounded* if it is n-bounded for some number n.

Theorem 5.1 *The following problem is undecidable:*

> *Given: a broadcast protocol B, a parameterized configuration $\mathbf{p}_0$.*
> *To decide: if $L_\omega(B, \mathbf{p}_0) = \emptyset$.*

Proof: We proceed by reduction from the following undecidable problem:

> Given: a 2-counter machine M.
> To decide: Does M halt on the input $(0, 0)$?

Let M' be a counter machine with 3 counters, behaving as follows. Initially, M' sets all counters to 0; then it simulates M on the counters c_1 and c_2, but after each step in the simulation it increases c_3 by 1. If M halts, then M' goes back to its initial state.

We make the following two observations about M':

- M' has an infinite bounded initial run if and only if M halts for $(0, 0)$.

  The only bounded initial run of M', if any, corresponds to the infinite iteration of the accepting run of M on $(0, 0)$ (all other infinite runs continuously increase c_3).

- Every infinite bounded run of M' (not necessarily initial!) contains infinitely many initial configurations.

  Such a run must set c_3 to 0 infinitely often, and this can only be done after visiting an initial configuration.

We simulate in a weak sense the machine M' by a broadcast protocol B. In B we have a state for each state and each counter of M', and two special states D and I. D is a special 'dead' state and I is introduced to keep an invariant (see below). The total number of processes in the counters of B plus the number of processes in I never increases. The following table describes the simulation:

Counter machine	Broadcast protocol
$q \xrightarrow{c:=c+1} q'$	$q \xrightarrow{inc_c!} q'$ $I \xrightarrow{inc_c?} c$
$q \xrightarrow{c:=c-1} q'$	$q \xrightarrow{dec_c!} q'$ $c \xrightarrow{dec_c?} I$
$q \xrightarrow{c=0} q'$	$q \xrightarrow{reset_c!!} q'$ $c \xrightarrow{reset_c??} D$

The parameterized configuration $\mathbf{p}_0$ puts 1 process in the initial state q_0, arbitrarily many in I, and 0 processes elsewhere.

[2] No automaton is used in [5] either.

The only situation in which the broadcast protocol does not faithfully simulate a step of the counter machine occurs when a $reset_c$ broadcast is executed at a configuration having at least one process in the counter c. We call such a broadcast a *cheat*.

Take an arbitrary run of the broadcast protocol and compute for all configurations $\mathbf{c}$ the sum $S(\mathbf{c}) = \mathbf{c}(c_1) + \mathbf{c}(c_2) + \mathbf{c}(c_3) + \mathbf{c}(I)$. The sums form a non-increasing sequence. Moreover, the sequence decreases only when the protocol cheats. We prove:

(1) If M halts for $(0,0)$, then $L_\omega(B, \mathbf{p}_0) \neq \emptyset$.
If M halts for $(0,0)$, then M' has a bounded infinite initial run, which iterates infinitely often the accepting run of M on $(0,0)$. Let b be the bound of this run. We consider the configuration $\mathbf{c} \in \mathbf{p}_0$ that puts b processes in I. We claim that B has an infinite run from $\mathbf{c}$. This run exactly mimics the infinite run of M'; since the infinite run is b-bounded, the total number of processes in the counters of B never exceeds b, and so B can mimic it even though there are only b processes in I. Since in this run the protocol only executes $q \xrightarrow{reset_c!!} q'$ when there are no processes in c, there are no cheats. So this run of B faithfully simulates the run of M', and so it is infinite.

(2) If $L_\omega(B, \mathbf{p}_0) \neq \emptyset$, then M halts for $(0,0)$.
Let $\mathbf{c} \in \mathbf{p}_0$ be a configuration such that B has an infinite run from $\mathbf{c}$. Since each cheat strictly decreases the sum $S(\mathbf{c})$, the run contains only finitely many cheats. Take a suffix of the run containing no cheats. Since the suffix is infinite, it corresponds to an infinite run r of M'. Moreover, r is bounded, because no counter can ever be larger than $\mathbf{c}(I)$. Now recall that every infinite bounded run of M' contains infinitely many initial configurations. So some suffix r' of r is an initial run of M'. Clearly M halts for the input $(0,0)$. ■

6. Conclusions

In this paper we have studied (parameterized) broadcast protocols, a model introduced by Emerson and Namjoshi in [5]. We have shown that the covering graph procedure proposed there for the verification of safety properties may not terminate, whereas termination is guaranteed for the procedure of [1] based on upward closed sets. So, while the covering graph technique is certainly adequate for several classes of systems, it is not the most suitable for broadcast protocols. Finally, we have shown that the model-checking problem for liveness properties is undecidable. In fact, even the problem of deciding if a broadcast protocol may exhibit an infinite behaviour is undecidable.

Acknowledgements Many thanks to Kedar Namjoshi for helpful discussions, to Giorgio Delzanno for implementing the backwards reachability algorithm using constraint programming, and to three ananymous referees, whose comments helped us to improve the presentation and correct a minor mistake.

References

[1] P. Abdulla, K. Cerans, B. Jonnson, Y.K. Tsay. General Decidability Theorems for Infinite State Systems. LICS, 1996.

[2] P. Abdulla, B. Jonnson. Undecidable Verification problems for Programs with Unreliable Channels. ICALP, LNCS 820, 1994.

[3] C. Dufourd, A. Finkel, P. Schnoebelen. Reset Nets Between Decidability and Undecidability. ICALP, LNCS 1443, 1998.

[4] E.A. Emerson, K.S. Namjoshi. Automatic Verification of Parameterized Synchronous Systems. CAV, LNCS 1102, 1996.

[5] E.A. Emerson, K.S. Namjoshi. On Model Checking for Non-Deterministic Infinite-State Systems. LICS, 1998.

[6] A. Finkel. Reduction and covering of infinite reachability trees. Information and Computation, 89(2):144-179, 1990.

[7] A. Finkel, P. Schnoebelen. Well-structured Transition Systems Everywhere! Research Report LSV-98-4, Lab. Specification and Verification, ENS de Cachan, France, 1998. To appear in TCS.

[8] S.M. German, A.P. Sistla. Reasoning about Systems with Many Processes. JACM 39(3), 1992.

[9] R. Karp, R. Miller. Parallel Program Schemata. JCSS 3, 1969.

[10] R. Mayr. Lossy Counter Machines. Technical Report TUM-I9827, Technische Universität München, 1998.

[11] M. Vardi, P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. LICS, 1986.

Decidability of Model Checking with the Temporal Logic EF

Richard Mayr

Institut für Informatik
Technische Universität München
Arcisstr. 21, D-80290 München, Germany
e-mail: mayrri@informatik.tu-muenchen.de

Abstract

The branching-time temporal logic EF is a simple, but natural fragment of computation-tree logic (CTL) and the modal μ-calculus. We study the decidability of the model checking problem for EF and infinite-state systems. We use process rewrite systems (PRS) to describe infinite-state systems and define a hierarchy of subclasses of PRS that includes Petri nets, pushdown processes, Basic Parallel Processes (BPP), context-free processes and PA-Processes. Then we establish the exact limits of the decidability of model checking with EF in this hierarchy.

Model checking with EF is undecidable for Petri nets and even for parallel pushdown automata (the pushdown extension of Basic Parallel Processes). On the other hand, model checking with EF is decidable for PAD, a process model that subsumes both PA-processes and pushdown processes.

Key words: infinite-state systems, temporal logic, EF, model checking, process algebra, PA-processes, pushdown processes

1 Introduction

The branching-time temporal logic EF (also called UB^- in [11] and [19]) uses the boolean operators, the one-step next operator EX (for some successor), and the operator EF (for some path eventually in the future). It is a fragment of computation tree logic (CTL), which in turn is weaker than the modal μ-calculus [5]. EF-formulae are interpreted over (possibly infinite) trees describing all possible computations of a process. The processes can also have infinite state spaces.

There are many models for systems with infinite state spaces. Some of the most common are Milner's Calculus of Communicating Systems (CCS) [25], Basic Parallel Processes (BPP) [9], context-free processes (BPA), pushdown processes and Petri nets. The process algebra PA is a common generalization of BPP and BPA and has operators for nondeterministic choice, parallel composition, sequential composition and recursion. Unlike BPP, PA is not a syntactical subset of CCS [25], because CCS does not have an explicit operator for sequential composition. However, as CCS can simulate sequential composition by parallel composition and synchronization, PA is still a weaker model than CCS. PA-processes, pushdown processes and Petri nets are mutually incomparable (see Section 2).

Except for CCS, all these models can be represented by special subclasses of a general rewriting formalism. These rewrite systems called "Process Rewrite Systems (PRS)" were introduced in [18,22], together with a hierarchy of its subclasses (the "PRS-hierarchy"). The PRS-hierarchy is a common generalization of two separate hierarchies for rewrite systems with sequential and parallel composition that were defined by Stirling, Moller and Caucal [26] (see also [11,7]) in analogy to the Chomsky-hierarchy. In this hierarchy, there is a natural common generalization of PA-processes and pushdown processes. This model was called PAD (for PA + PD) in [18,22] and it is strictly more general than PA and pushdown processes with respect to bisimulation equivalence.

The model checking problem is the problem if a given process satisfies a property encoded as a formula in a certain temporal logic. We study the model checking problem for the logic EF and the models in the PRS-hierarchy. The main new result in this paper is that model checking with EF is decidable even for PAD. This completes the general picture of the decidability of model checking with EF.

In Section 2 we define Process Rewrite Systems (PRS) and the PRS-hierarchy of its subclasses. In Section 3 we define the logic EF and some generalizations of EF. In Section 4 we show that model checking PAD with EF is decidable. In Section 5 we describe a small example. In Section 6 we show that model checking with EF is undecidable for PPDA, the pushdown extension of BPP, which is a subclass of Petri nets. In the last section we present a general view of the limits of the decidability of model checking with EF and other temporal logics.

2 Process Models

Many classes of concurrent systems can be described by a (possibly infinite) set of process terms, representing the states, and a finite set of rewrite rules

describing the dynamics of the system.

Definition 1 *Let $Act = \{a, b, \dots\}$ be a countably infinite set of atomic actions and $Const = \{\epsilon, X, Y, Z, \dots\}$ a countably infinite set of process constants. The process terms that describe the states of the system have the following form:*

$$t ::= \epsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2$$

where ϵ is the empty term, $X \in Const$ is a process constant (used as an atomic process in this context), "$\|$" means parallel composition and "." means sequential composition. Parallel composition is associative and commutative. Sequential composition is associative. Let $\mathcal{T}$ be the set of process terms.

Convention 1: *We always work with equivalence classes of terms modulo commutativity and associativity of parallel composition and modulo associativity of sequential composition. Also we define that $\epsilon.t = t = t.\epsilon$ and $t\|\epsilon = t$.*

Convention 2: *We defined that sequential composition is associative. However, when we look at terms we think of it as left-associative. So when we say that a term t has the form $t_1.t_2$, then we mean that t_2 is either a single constant or a parallel composition of process terms.*

The size *of a process term is defined as the number of occurrences of constants in it plus the number of occurrences of operators in it.*

$$\begin{aligned} size(\epsilon) &:= 0 \\ size(X) &:= 1 \\ size(t_1.t_2) &:= size(t_1) + size(t_2) + 1 \\ size(t_1 \| t_2) &:= size(t_1) + size(t_2) + 1 \end{aligned}$$

For a term t the set $Const(t)$ is the set of constants that occur in t.

$$\begin{aligned} Const(\epsilon) &:= \emptyset \\ Const(X) &:= \{X\} \\ Const(t_1.t_2) &:= Const(t_1) \cup Const(t_2) \\ Const(t_1 \| t_2) &:= Const(t_1) \cup Const(t_2) \end{aligned}$$

The dynamics of the system is described by a finite set of rules Δ of the form $(t_1 \xrightarrow{a} t_2)$ where t_1 and t_2 are process terms and $a \in Act$ is an atomic action. The finite set of rules Δ induces a (possibly infinite) labeled transition system with relations $\xrightarrow{a}$ with $a \in Act$. For every $a \in Act$, the transition relation $\xrightarrow{a}$

is the smallest relation that satisfies the following inference rules.

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 \| t_2 \xrightarrow{a} t_1' \| t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 . t_2 \xrightarrow{a} t_1' . t_2}$$

where t_1, t_2, t_1', t_2' are process terms. Note that parallel composition is commutative and thus the inference rule for parallel composition also holds with t_1 and t_2 exchanged.

Since Δ is finite, the generated LTS is finitely branching. (For some classes of systems (e.g. Petri nets) the branching-degree is bounded by a constant that depends on Δ. For other classes (e.g. PA) the branching-degree is finite at every state, but it can get arbitrarily high.) Also every single Δ uses only a finite subset $Const(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} (Const(t_1) \cup Const(t_2))$ of constants and only a finite subset $Act(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} \{a\}$ of atomic actions. Thus for every Δ only finitely many of the generated transition relations $\xrightarrow{a_i}$ for $a_i \in Act$ are nonempty. (Those for which $a_i \in Act(\Delta)$). Still the generated transition system can be infinite. (Consider the analogy: Every labeled Petri net has only finitely many transitions and uses only finitely many different atomic actions, but the state space can be infinite.) The relation $\xrightarrow{a}$ is generalized to sequences of actions in the standard way. Sequences are denoted by σ.

Remark 2 *There is no operator "+" for nondeterministic choice in the process terms, because this is encoded in the set of rules Δ! There can be several rules with the same term on the left hand side. It is also possible that several rules are applicable at different places in a term. The rule that is applied and the position where it is applied are chosen nondeterministically. Also there is no such thing as action prefixes in the process terms. The atomic actions are introduced by the rules.*

Many common models of systems fit into this scheme. In the following we characterize subclasses of rewrite systems. The expressiveness of a class depends on what kind of terms are allowed on the left hand side and right hand side of the rewrite rules in Δ.

Definition 3 (Classes of process terms)
We distinguish four classes of process terms:

- ***1*** *Terms consisting of a single process constant like X.*
- ***S*** *Terms consisting of a single constant or a sequential composition of process constants like $X.Y.Z$.*
- ***P*** *Terms consisting of a single constant or a parallel composition of process constants like $X \| Y \| Z$.*
- ***G*** *General process terms with arbitrary sequential and parallel composition like $(X.(Y \| Z)) \| W$.*

Also let $\epsilon \in S, P, G$, but $\epsilon \notin 1$. It is easy to see that the relations between these classes of process terms are: $1 \subset S$, $1 \subset P$, $S \subset G$ and $P \subset G$. S and P are incomparable and $S \cap P = 1 \cup \{\epsilon\}$.

We characterize classes of process rewrite systems (PRS) by the classes of terms allowed on the left hand sides and the right hand sides of rewrite rules.

Definition 4 (PRS)
Let $\alpha, \beta \in \{1, S, P, G\}$. A (α, β)-PRS is a finite set of rules Δ where for every rewrite rule $(l \xrightarrow{a} r) \in \Delta$ the term l is in the class α and $l \neq \epsilon$ and the term r is in the class β (and can be ϵ). The initial state is given as a term $t_0 \in \alpha$. A (G, G)-PRS is simply called PRS.

Remark 5 *W.l.o.g. it can be assumed that the initial state t_0 of a PRS is a single constant. There are only finitely many terms $t_1, \ldots, t_n$ s.t. $t_0 \xrightarrow{a_i} t_i$. If t_0 is not a single constant then we can achieve this by introducing a new constant X_0 and new rules $X_0 \xrightarrow{a_i} t_i$ and declaring X_0 to be the initial state.*

(α, β)-PRS where α is more general than β or incomparable to β (for example $\alpha = G$ and $\beta = S$) do not make any sense. This is because the terms that are introduced by the right side of rules must later be matched by the left sides of other rules. So in a (G, S)-PRS the rules that contain parallel composition on the left hand side will never be used (assuming that the initial state is a single constant). Thus one may as well use a (S, S)-PRS. So we restrict our attention to (α, β)-PRS with $\alpha \subseteq \beta$.

Figure 1 shows a graphical description of the hierarchy of (α, β)-PRS.

Many of these (α, β)-PRS correspond to widely known models like Petri nets, pushdown processes, context-free processes and others.

(1) A $(1, 1)$-PRS is a finite-state system. Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.
(2) $(1, S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [7,11]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.
(3) It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left side of rules). Caucal [8] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labeling of states. Thus (S, S)-PRS are equivalent to pushdown processes, the processes described by pushdown automata.
(4) (P, P)-PRS are equivalent to Petri nets. Every constant corresponds to a place in the net and the number of occurrences of a constant in a term

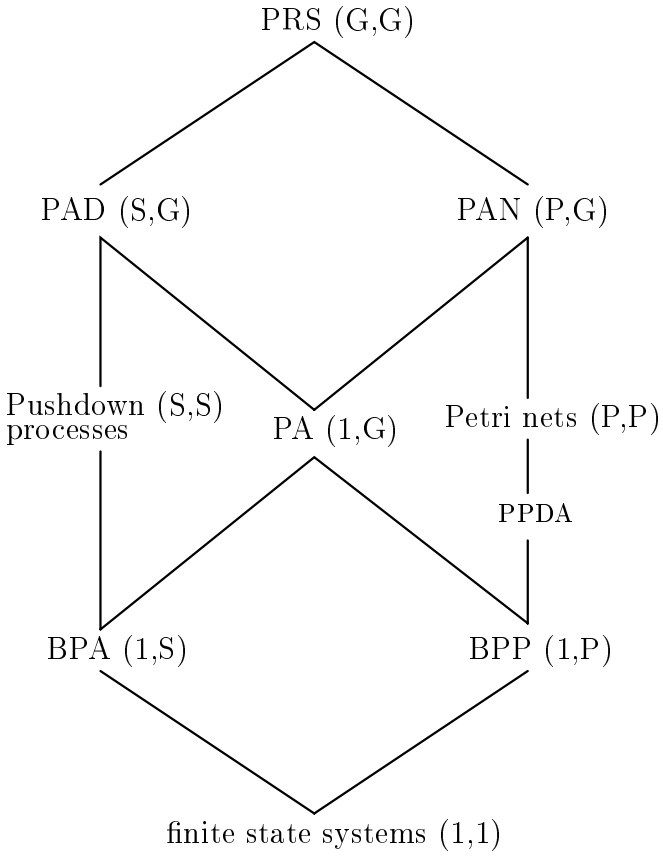


Fig. 1. The PRS-hierarchy

corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.

(5) $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [7,11]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [9].

(6) $(1, G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication (see [1,21,16]).

(7) (P, G)-PRS are called *PAN-processes* in [20]. It is a common generalization of Petri nets and PA-processes and it is strictly more general than both of them (e.g. PAN can describe all Chomsky-2 languages while Petri nets cannot).

(8) (S, G)-PRS are a common generalization of pushdown processes and PA-processes. They are called *PAD* (PA + PD) in [23].

(9) The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [18,22]. They subsume all the previously mentioned classes.

What does it mean that parallel/sequential/arbitrary composition is allowed in terms on the left/right hand sides of rules? The general intuition is as follows:

If parallel composition is allowed on the right hand side of rules, then there can be rules of the form $t \xrightarrow{a} t_1 \| t_2$. This means that it is possible to create processes that run in parallel. The rule can be interpreted that, by action a, the process t becomes the process t_1 and spawns off the process t_2 or vice versa. If sequential composition is allowed on the right hand side of rules, then there are rules of the form $t \xrightarrow{a} t_1.t_2$. The interpretation is that process t calls a subroutine t_1 and becomes process t_2. It resumes its execution when the subroutine t_1 terminates. If arbitrary sequential and parallel composition is allowed on the right hand side of rules then both parallelism and subroutines are possible. If parallel composition is allowed on the left hand side of rules, then there are rules of the form $t_1 \| t_2 \xrightarrow{a} t$. This can be interpreted as synchronization/communication of the parallel processes t_1 and t_2. This is because this action can only occur if both t_1 and t_2 change in a certain defined way. If sequential composition is allowed on the left hand side of rules, then there can be rules of the form $t_1'.t_2 \xrightarrow{a} t'$ and $t_1''.t_2 \xrightarrow{a} t''$. The intuition is that a process t called a subroutine t_1 and became process t_2 by a rule $t \xrightarrow{a} t_1.t_2$. The subroutine may in its computation reach a state t_1' or t_1''. Now one of these rules is applicable. This means that the result of the computation of the subroutine affects the behavior of the caller when it becomes active again, since the caller can become t' or t''. The interpretation is that the subroutine returns a value to the caller when it terminates. If arbitrary sequential and parallel composition is allowed on the left hand sides of rules then both synchronization and returning of values by subroutines are possible. Rules with nested sequential and parallel composition (on the left side or the right side) do not increase the expressiveness [18].

Thus, for example, the processes of class PAD (type (S, G) in the PRS-hierarchy) have parallel composition and subroutines that can return values to their caller, but they lack the ability to synchronize parallel processes.

It has been shown in [18] (with the help of earlier results from [6,26]) that this hierarchy of subclasses of PRS is strict w.r.t. bisimulation equivalence, i.e., more general subclasses are strictly more expressive. (See [25,26] for more on bisimulation.)

Theorem 6 ([18])
The PRS-hierarchy is strict with respect to bisimulation equivalence.

It has also been shown in [18] that PRS are not Turing-powerful.

Theorem 7 ([18])
The reachability problem is decidable for PRS.

3 The Temporal Logic EF

Temporal logics are used to describe properties of systems. The verification process consists in showing that a given system satisfies a property encoded in a given formula. We use the logic $EF^{=}_{DC}$, an extended version of the logic EF [11,7]. In addition to the standard operators of EF, the logic $EF^{=}_{DC}$ uses strong atomic propositions of the form 'The current state is term t' and can thus express the reachability problem. The "=" in the name stands for these strong propositions, because they express that the current state is equal to a given state t. Note that, because of this feature, the logic $EF^{=}_{DC}$ (unlike EF) is not a fragment of CTL or the modal μ-calculus. The modal μ-calculus (and CTL) cannot distinguish bisimilar states, but $EF^{=}_{DC}$ can. The logic $EF^{=}_{DC}$ can also express weak constraints on sequences of actions. These constraints are called *decomposable constraints* (thus the DC in the name).

Definition 8 ($EF^{=}_{DC}$)
The syntax of the formulae is as follows:

$$\Phi ::= t \;\mid\; \neg\Phi \;\mid\; \Phi_1 \wedge \Phi_2 \;\mid\; \Diamond_C \Phi$$

where $t \in \mathcal{T}$ is a process term and C is a decomposable constraint *(see Def. 10).*

Let $\mathcal{F}$ be the set of all $EF^{=}_{DC}$-formulae. Let $\mathcal{T}$ be the set of all processes terms (as in Def. 1) in the process algebra. The denotation $[\![\Phi]\!]$ of an $EF^{=}_{DC}$-formula Φ is the set of process terms defined inductively by the following rules:

$$\begin{aligned}
[\![t]\!] &:= \{t\} \\
[\![\neg\Phi]\!] &:= \mathcal{T} - [\![\Phi]\!] \\
[\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\
[\![\Diamond_C \Phi]\!] &:= \{t \in \mathcal{T} \mid \exists t', \sigma.\ t \xrightarrow{\sigma} t' \ \wedge\ t' \in [\![\Phi]\!] \ \wedge\ C(\sigma)\}
\end{aligned}$$

Disjunction can be expressed by conjunction and negation.

The property $t \in [\![\Phi]\!]$ is also denoted by $t \models \Phi$.

The model checking problem is the problem if a process satisfies a property encoded as a formula in a temporal logic.

Model Checking

Instance: A description of a process (for example an (α, β)-PRS Δ with $\alpha, \beta \in \{1, S, P, G\}$ (see Def. 3)) and a state t and a formula Φ from a temporal logic (for example EF or $EF^{=}_{DC}$).

Question: Is it true that $t \models \Phi$?

Model checking finite-state systems with EF can be done in polynomial time, since EF is a fragment of the alternation-free modal μ-calculus [11,7]. Model checking Petri nets with EF has been shown to be undecidable [11,7] by reduction of the reachability set containment problem for Petri nets. Model checking with EF is *PSPACE*-complete for Basic Parallel Processes (BPP) [23,19], and context-free processes (BPA) [2,24]. For pushdown processes the complexity of model checking with EF is between *PSPACE*and *EXPTIME*[2,29,30]. It was claimed in [2] that model checking pushdown processes with EF is *PSPACE*-complete. Unfortunately, the given proof is wrong. It assumes that an accepting polynomial space-bounded Turing-machine has an accepting computation of polynomial length, which is not true in general. It was shown in [21] that model checking with EF is decidable for PA-processes. Lugiez and Schnoebelen [17] later proved the same result by using a completely different method (using tree-automata to represent infinite sets of configurations). In the next section we show that model checking with the more general logic $EF^{=}_{DC}$ is decidable for the more general model PAD (type (S, G) in the PRS-hierarchy). But first we reduce the problem to a simpler form.

Definition 9 *For any $EF^{=}_{DC}$-formula Φ let $terms(\Phi)$ be the set of process terms used in Φ as atomic propositions.*

$$\begin{aligned} terms(t) &:= \{t\} \\ terms(\neg\Phi) &:= terms(\Phi) \\ terms(\Phi_1 \wedge \Phi_2) &:= terms(\Phi_1) \cup terms(\Phi_2) \\ terms(\Diamond_C \Phi) &:= terms(\Phi) \end{aligned}$$

The logic $EF^{=}_{DC}$ uses constraints on sequences of actions. These constraints are called *decomposable*, because they can be decomposed with respect to sequential and parallel composition of sequences of actions.

Definition 10 (Decomposable Constraints)
A set of decomposable constraints $\mathcal{DC}$ is a finite set of predicates on finite sequences of actions that satisfy the following conditions.

(1) $\mathcal{DC}$ contains the predicates true (all sequences satisfy it) and false (no sequence satisfies it).
(2) For every predicate $C \in \mathcal{DC}$ it is decidable if C is satisfiable.
(3) For every $C \in \mathcal{DC}$ there is a finite index set I and a finite set of decomposable constraints $\{C^1_i, C^2_i \in \mathcal{DC} \mid i \in I\}$ s.t.

$$\forall \sigma, \sigma_1, \sigma_2.\ \sigma_1\sigma_2 = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C^1_i(\sigma_1) \wedge C^2_i(\sigma_2) \right)$$

(4) For every $C \in \mathcal{DC}$ there is a finite index set I and a finite set of decom-

posable constraints $\{C'_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma, \sigma'.\ a\sigma' = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C'_i(\sigma') \right)$$

(5) For every $C \in \mathcal{DC}$ *there is a finite index set* I *and a finite set of decomposable constraints* $\{C^1_i, C^2_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma, \sigma_1, \sigma_2. \forall a \in Act.\ \sigma_1 a \sigma_2 = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C^1_i(\sigma_1) \wedge C^2_i(\sigma_2) \right)$$

(6) For every $C \in \mathcal{DC}$ *there is a finite index set* I *and a finite set of decomposable constraints* $\{C^1_i, C^2_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma_1, \sigma_2. \left((\exists \sigma \in \mathit{interleave}(\sigma_1, \sigma_2). C(\sigma)) \iff \bigvee_{i \in I} (C^1_i(\sigma_1) \wedge C^2_i(\sigma_2)) \right)$$

$\sigma \in \mathit{interleave}(\sigma_1, \sigma_2)$ *means that* σ *is an arbitrary interleaving of* σ_1 *and* σ_2*. The formal definition of the function interleave is as follows: Let* λ *be the empty sequence.*

$$\begin{aligned} \mathit{interleave}(\lambda, \sigma) &:= \{\sigma\} \\ \mathit{interleave}(\sigma, \lambda) &:= \{\sigma\} \\ \mathit{interleave}(a_1\sigma_1, a_2\sigma_2) &:= \begin{array}{l} \{a_1\sigma \mid \sigma \in \mathit{interleave}(\sigma_1, a_2\sigma_2)\} \cup \\ \{a_2\sigma \mid \sigma \in \mathit{interleave}(a_1\sigma_1, \sigma_2)\} \end{array} \end{aligned}$$

Lemma 11 *If* $\mathcal{DC}$ *is a set of decomposable constraints, then the closure* $\mathcal{DC}'$ *of* $\mathcal{DC}$ *under the boolean operations of conjunction and disjunction is also a set of decomposable constraints.*

PROOF. *The formulae in* $\mathcal{DC}'$ *can be transformed into disjunctive normal form, such that the formulae in* $\mathcal{DC}$ *are the atomic formulae. Since* $\mathcal{DC}$ *is finite,* $\mathcal{DC}'$ *is finite too.*

Remark 12 *A set of decomposable constraints need not be closed under negation.*

Now we give an example for a set of decomposable constraints. Let $A \subset Act$, be a finite set of atomic actions. For any $a \in A$ let $\#_a(\sigma)$ be the number of occurrences of action a in σ. For $u, v \in \mathbb{N}$ let $[u]_v$ denote u modulo v. We define the following constraints:

(1) $\mathit{length}(\sigma) \geq i$ or $\mathit{length}(\sigma) \leq i$ for all $i \leq k$ for some fixed constant k.
(2) $\#_a(\sigma) \geq i$ or $\#_a(\sigma) \leq i$ for all $i \leq n$ for some fixed constant n.

(3) $[\#_a(\sigma)]_k = i$ for all $i, k \leq m$ for some fixed constant m.
(4) $\mathit{first}(\sigma) = a$ for any action $a \in A$.

For any choice of A, k, n, m let $\mathcal{C}_{A,k,n,m}$ denote the closure of the set of these constraints under conjunction and disjunction.

Lemma 13 *For any A, k, n, m, the set $\mathcal{C}_{A,k,n,m}$ is a set of decomposable constraints. It is even closed under negation.*

PROOF. *Directly from the definitions.*

Example 14 *The constraint $[\#_a(\sigma)]_2 = 0$ expresses that the number of occurrences of action a in σ is even. Let $\sigma \in \mathit{interleave}(\sigma_1, \sigma_2)$ be an interleaving of two sequences. Then the number of occurrences of the action a in σ is even iff it is either even in both σ_1 and σ_2 or odd in both σ_1 and σ_2. This can be expressed by the following decomposition.*

$$\begin{aligned}[\#_a(\sigma)]_2 = 0 \iff & ([\#_a(\sigma_1)]_2 = 0 \ \wedge \ [\#_a(\sigma_2)]_2 = 0) \ \vee \\ & ([\#_a(\sigma_1)]_2 = 1 \ \wedge \ [\#_a(\sigma_2)]_2 = 1)\end{aligned}$$

Decomposable constraints increase the expressiveness of the logic. They have proved to be useful for constructing characteristic formulae for finite-state systems up to (different kinds of) bisimulation-like equivalences [15]. For more details on decomposable constraints and decomposable languages see [17,28].

We use these constraints to show that the usual definition of EF is a fragment of $EF^{=}_{DC}$. The usual $\Diamond$ is just $\Diamond_{\mathit{true}}$. The normal one-step nexttime operator EX is often denoted by $\langle a \rangle$ and defined by

$$[\![\langle a \rangle \Phi]\!] := \{t \mid \exists t'.\, t \xrightarrow{a} t' \ \wedge \ t' \in [\![\Phi]\!]\}$$

It is clear that $\langle a \rangle = \Diamond_C$ with $C := [\mathit{first}(\sigma) = a \wedge \mathit{length}(\sigma) = 1]$. The normal version of EF also does not have atomic propositions t (meaning that the state is equal to t; see Def. 8), but propositions "a" (meaning that the atomic action a is enabled). This can be expressed by $\langle a \rangle \mathit{true}$, where $\mathit{true} = t \vee \neg t$ for any term t.

It is also possible to express the modal operator $\Box$ (meaning 'always') by defining $\Box_C := \neg \Diamond_C \neg$. $\Box_C \Phi$ then means that Φ holds in all states that are reachable via a sequence of actions σ s.t. $C(\sigma)$.

Definition 15 *The* nesting-depth $nd(\Phi)$ *of an $EF^{=}_{DC}$-formula Φ is defined by*

$$
\begin{aligned}
nd(t) &:= 0 \\
nd(\neg\Phi) &:= nd(\Phi) \\
nd(\Phi_1 \wedge \Phi_2) &:= max\{nd(\Phi_1), nd(\Phi_2)\} \\
nd(\Diamond_C \Phi) &:= nd(\Phi) + 1
\end{aligned}
$$

Definition 16 *$\mathcal{F}_d \subset \mathcal{F}$ is defined as the set of all $EF^{=}_{DC}$-formulae with a nesting-depth of modal operators $\Diamond_C$ of at most d.*

$$\mathcal{F}_d := \{\Phi \in \mathcal{F} \mid nd(\Phi) \leq d\}$$

It follows that formulae in $\mathcal{F}_0$ contain no modal operators.

In order to simplify the notation we use some abbreviations:
Let $T = \{t_1, \ldots, t_n\} \subseteq \mathcal{T}$ be a finite set of process terms, then

$$t \models -T \iff t \models \neg t_1 \wedge \cdots \wedge \neg t_n$$

For reasons of symmetry we also define

$$t \models T : \iff t \models t_1 \wedge \cdots \wedge t_n$$

Of course this cannot be true if $n \geq 2$.

Definition 17 *We define a subset $\mathcal{F}^c_d \subset \mathcal{F}_d$ of formulae that do not contain disjunction. Thus the formulae in $\mathcal{F}^c_d$ are called* conjunctive formulae. *$\mathcal{F}^c_d$ is defined as the minimal set of formulae Φ_d that are defined by the following grammar.*

$$\Phi_0 = T^+ \wedge -T^-$$

for every finite $T^+, T^- \subset \mathcal{T}$ and

$$\Phi_d = T^+ \wedge -T^- \mid \Phi_d \wedge \Diamond_C \Phi_{d-1} \mid \Phi_d \wedge \neg\Diamond_C \Phi_{d-1}$$

for every finite $T^+, T^- \subset \mathcal{T}$ and every decomposable constraint C and every $\Phi_{d-1} \in \mathcal{F}^c_{d-1}$.

It follows that every formula in $\mathcal{F}^c_d$ has the form

$$T^+ \wedge -T^- \wedge \bigwedge_{i \in I} \Diamond_{C_i} \Psi_i \wedge \bigwedge_{j \in J} \neg\Diamond_{D_j} \Upsilon_j$$

where $T^+, T^- \subset \mathcal{T}$, and C_i, D_j are decomposable constraints and $\Psi_i \in \mathcal{F}^c_{d-1}$ and $\Upsilon_j \in \mathcal{F}^c_{d-1}$.

A formula Φ is in normal form *if $\Phi = \bigvee_{i \in I} \Diamond_{C_i} \Psi_i$ s.t. the Ψ_i are conjunctive formulae.*

Lemma 18 *Any $EF^{=}_{DC}$-formula $\Diamond_C\Phi$ is equivalent to a formula in normal form.*

PROOF. *By induction on the nesting-depth d of modal operators in Φ. The important property here is that $\Diamond_C(\Phi_1 \vee \Phi_2) = \Diamond_C\Phi_1 \vee \Diamond_C\Phi_2$. We transform the subformulae into disjunctive normal form, and then push the disjunctions outwards.*

Lemma 19 *Every model checking problem for $EF^{=}_{DC}$ is decidable iff it is decidable for all formulae $\Diamond_C\Phi$ with $\Phi \in \bigcup_{d\in\mathrm{N}} \mathcal{F}^c_d$.*

PROOF. *If it is decidable for formulae of the form $\Diamond_C\Phi$ with $\Phi \in \mathcal{F}^c_d$, then it is decidable for formulae in normal form and thus by Lemma 18 for all formulae of the form $\Diamond_C\Psi$, with $\Psi \in \mathcal{F}$. Simple boolean operations yield the decidability of the whole model checking problem. The other direction is trivial.*

4 Model Checking PAD

We prove the decidability of the model checking problem for $EF^{=}_{DC}$ and PAD by construction of a sound and complete tableau. By Lemma 19 it suffices to consider formulae of the form $\Diamond_C\Phi$ for $\Phi \in \mathcal{F}^c_d$ for any d and C in some set of decomposable constraints.

4.1 Decomposition

The key to the construction of the tableau system in Subsection 4.2 is that properties of the form $t_1.t_2 \models \Diamond_C\Phi$ or $t_1\|t_2 \models \Diamond_C\Phi$ can be decomposed into properties of t_1 and properties of t_2. First we give a small example how this is done and then we do it in general.

Example 20 *We show how to do the decomposition for the following simple formula of nesting-depth two:*

$$\Phi := \Diamond(\neg u \ \wedge \ \Diamond(v) \ \wedge \ \neg\Diamond(w))$$

where u, v, w are process terms. No decomposable constraints are used, except for the constraint true ($\Diamond_{true} = \Diamond$). This formula means that there is a reachable state different from u, s.t. from this state the state v is reachable, but the state w is not reachable.

Let t_1, t_2 be process terms. Then the property

$$t_1 \| t_2 \models \Diamond(\neg u \ \wedge \ \Diamond(v) \ \wedge \ \neg\Diamond(w))$$

is equivalent to

$$\begin{array}{l}
(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \vee \\
(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \vee \\
\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \\
\qquad\qquad \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \ t_1' \| t_2' \models \Diamond(v) \ \wedge \ t_1' \| t_2' \models \neg\Diamond(w)
\end{array}$$

where α_1, α_2 are process terms. This is equivalent to

$$\begin{array}{l}
(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \vee \\
(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \vee \\
\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \\
\qquad\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \ \wedge \\
\qquad\qquad \neg \left(\bigvee_{\gamma_1 \| \gamma_2 = w} t_1' \models \Diamond(\gamma_1) \wedge t_2' \models \Diamond(\gamma_2) \right)
\end{array}$$

This is equivalent to

$$\begin{array}{l}
(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \vee \\
(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \vee \\
\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \wedge \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \\
\qquad\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \ \wedge \\
\qquad\qquad \bigwedge_{\gamma_1 \| \gamma_2 = w} (t_1' \models \neg\Diamond(\gamma_1) \vee t_2' \models \neg\Diamond(\gamma_2))
\end{array}$$

Now we transform this expression into disjunctive normal form. We define the set F of all functions f that assign to every pair (α_1, α_2) s.t. $\alpha_1 \| \alpha_2 = u$, a value in $\{1, 2\}$. For every $f \in F$ let $A_f^1 := \{\alpha_1 \ \mid \ f((\alpha_1, \alpha_2)) = 1\}$ and

$A_f^2 := \{\alpha_2 \mid f((\alpha_1, \alpha_2)) = 2\}$. *Then the expression is equivalent to*

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \wedge \bigvee_{f \in F} (t_1' \notin A_f^1 \ \wedge \ t_2' \notin A_f^2) \wedge \\
&\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \wedge \\
&\qquad \bigwedge_{\gamma_1 \| \gamma_2 = w} (t_1' \models \neg\Diamond(\gamma_1) \vee t_2' \models \neg\Diamond(\gamma_2))
\end{aligned}
$$

In the same way we define the set G of all functions g that assign to every pair (γ_1, γ_2) s.t. $\gamma_1 \| \gamma_2 = w$, a value in $\{1, 2\}$. For every $g \in G$ let $B_g^1 := \{\gamma_1 \mid g((\gamma_1, \gamma_2)) = 1\}$ and $B_g^2 := \{\gamma_2 \mid g((\gamma_1, \gamma_2)) = 2\}$.

Then the expression is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \bigvee_{f \in F} (t_1' \models -A_f^1 \ \wedge \ t_2' \models -A_f^2) \wedge \\
&\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \wedge \\
&\qquad \bigvee_{g \in G} \left(\bigwedge_{\gamma_1 \in B_g^1} t_1' \models \neg\Diamond(\gamma_1) \ \wedge \ \bigwedge_{\gamma_2 \in B_g^2} t_2' \models \neg\Diamond(\gamma_2) \right)
\end{aligned}
$$

This is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \\
&\bigvee_{f \in F, \beta_1 \| \beta_2 = v, g \in G} t_1' \models -A_f^1 \wedge t_1' \models \Diamond(\beta_1) \wedge \bigwedge_{\gamma_1 \in B_g^1} t_1' \models \neg\Diamond(\gamma_1) \wedge \\
&\qquad t_2' \models -A_f^2 \wedge t_2' \models \Diamond(\beta_2) \wedge \bigwedge_{\gamma_2 \in B_g^2} t_2' \models \neg\Diamond(\gamma_2)
\end{aligned}
$$

Finally, this is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\bigvee_{f \in F, \beta_1 \| \beta_2 = v, g \in G} t_1 \models \Diamond(-A_f^1 \wedge \Diamond(\beta_1) \wedge \bigwedge_{\gamma_1 \in B_g^1} \neg\Diamond(\gamma_1)) \ \wedge \\
&\qquad\qquad\qquad\qquad t_2 \models \Diamond(-A_f^2 \wedge \Diamond(\beta_2) \wedge \bigwedge_{\gamma_2 \in B_g^2} \neg\Diamond(\gamma_2))
\end{aligned}
$$

This is a boolean combination of properties of t_1 and properties of t_2.

Now we show how the decomposition is done in the general case. In order to simplify the presentation, we define the following sets of expressions. Let $\mathcal{DC}$ be a set of decomposable constraints, $T \subset \mathcal{T}$ a finite set of process terms and $d \in \mathbb{N}$.

$$
\begin{aligned}
\mathit{Cform}(d, T, \mathcal{DC}) &:= \left\{ \begin{array}{l} \left(\bigwedge_{i \in I} t_i \models \Diamond_{C_i} \Phi_i \ \wedge \ \bigwedge_{j \in J} t'_j \models \neg\Diamond_{D_j} \Psi_j \right) \ \mid \\ \forall i, j.\ t_i, t'_j \in T, C_i, D_j \in \mathcal{DC}, \Phi_i \in \mathcal{F}_d^c, \Psi_j \in \mathcal{F}_{d-1}^c \end{array} \right\} \\
\mathit{Cform}'(d, T, \mathcal{DC}) &:= \text{like } \mathit{Cform}(d, T, \mathcal{DC}), \text{ except that } \Psi_j \in \mathcal{F}_d^c \\
\mathit{Dform}(d, T, \mathcal{DC}) &:= \left\{ \bigvee_{i \in I} F_i \ \mid \ F_i \in \mathit{Cform}(d, T, \mathcal{DC}) \right\}
\end{aligned}
$$

The next two lemmas show the decomposition of properties for sequential composition. The general idea is that properties of the form $t_1.t_2 \models \Diamond_C \Phi$ are decomposed into properties of t_1 and properties of t_2. However, the details are more complex. It does not always suffice to use properties of t_1 and properties of t_2, but sometimes also properties of other terms are needed. These other terms are the terms that occur in Φ as atomic propositions and the terms that occur in the rules of the PAD-process. Fortunately, these are only finitely many.

We defined that sequential composition is left-associative, so if we write $t_1.t_2$, then the term t_2 is either a single constant or a parallel composition. The following lemma describes the decomposition for the case that t_2 is a single constant.

Lemma 21 *Let t be a process term, X a process constant, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t, X\} \cup \mathit{terms}(\Phi) \cup \{r \mid (l \xrightarrow{a} r) \in \Delta\}$*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t.X \models \Diamond_C \Phi \iff F$$

PROOF. *by induction on d.*

$\Phi = (T^+ \wedge -T^- \wedge \bigwedge_{i \in I} \Diamond_{C_i} \Phi_i \wedge \bigwedge_{j \in J} \neg \Diamond_{D_j} \Psi_j)$ *for some $T^+, T^- \subset \mathcal{T}$, $\Phi_i, \Psi_j \in \mathcal{F}^c_{d-1}$ and $C_i, D_j \in \mathcal{DC}$ decomposable constraints. In the base case $d = 0$ the sets I and J are empty and we solve the problem without referring to the induction hypothesis.*

If $|T^+| \geq 2$ then $t.X \models \Diamond_C \Phi$ is equivalent to false.

If $|T^+| = 1$ s.t. the term in T^+ is not $t'.X$ for some t', then $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i} \Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j} \Phi)$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 3 and 5). This expression is the F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{u.X\}$ for some term u. If $u.X \in T^-$ then $t.X \models \Diamond_C \Phi$ is equivalent to false. Otherwise $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i} \Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j} \Phi)$$

$$\vee$$

$$t \models \Diamond_C(u) \wedge \bigwedge_{i \in I} u.X \models \Diamond_{C_i} \Phi_i \wedge \bigwedge_{j \in J} u.X \models \neg \Diamond_{D_j} \Psi_j$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 3 and 5). This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{\}$. Then $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i} \Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j} \Phi)$$

$$\vee$$

$$\exists \sigma, t'. \left(\begin{array}{l} t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall (\alpha.X) \in T^-.\ t' \neq \alpha) \wedge \\ \bigwedge_{i \in I} t'.X \models \Diamond_{C_i} \Phi_i \wedge \bigwedge_{j \in J} t'.X \models \neg \Diamond_{D_j} \Psi_j \end{array} \right)$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 4 and 5).

If $d = 0$ then $I = J = \{\}$ and the induction hypothesis is not needed. If $d > 0$ then by induction hypothesis there are expressions $F_i, G_j \in Dform(d-1, (T - \{t\}) \cup \{t'\}, \mathcal{DC})$ s.t. the above expression is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i} \Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j} \Phi)$$

$$\vee$$

$$\exists \sigma, t'.\ t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall (\alpha.X) \in T^-.\ t' \neq \alpha)\ \wedge \bigwedge_{i \in I} F_i \wedge \bigwedge_{j \in J} \neg G_j$$

By transformation to disjunctive normal form there are finite index sets K, N_k, N'_k, M_k and formulae $\Phi'_i, \Psi'_i \in \mathcal{F}^c_{d-1}$ and decomposable constraints $E_i, E'_i \in \mathcal{DC}$ and $H_j \in Cform'(d-1, T - \{t\}, \mathcal{DC})$ s.t. the above expression is equivalent

to

$$\bigvee_{i\in I}(t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\exists\sigma, t'.\left(\begin{array}{l} t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall(\alpha.X) \in T^-.\ t' \neq \alpha) \wedge \\ \bigvee_{k\in K}\left[\bigwedge_{i\in N_k} t' \models \Diamond_{E_i}\Phi_i' \bigwedge_{i\in N_k'} t' \models \neg\Diamond_{E_i'}\Psi_i' \bigwedge_{j\in M_k} H_j\right] \end{array}\right)$$

Note that the expressions H_j do not contain the terms t or t'. This is equivalent to

$$\bigvee_{i\in I}(t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\bigvee_{k\in K}\left[t \models \Diamond_C\left(-\{\alpha \mid \alpha.X \in T^-\} \wedge \bigwedge_{i\in N_k} \Diamond_{E_i}\Phi_i' \bigwedge_{i\in N_k'} \neg\Diamond_{E_i'}\Psi_i'\right) \wedge \bigwedge_{j\in M_k} H_j\right]$$

This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

The following lemma does the same decomposition for the case that the second component in the sequential composition is itself a parallel composition.

Lemma 22 *Let t_1, t_2, t_3 be process terms, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t_1, t_2 \| t_3\} \cup terms(\Phi) \cup \{r \mid (l \xrightarrow{a} r) \in \Delta\}$*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t_1.(t_2 \| t_3) \models \Diamond_C\Phi \iff F$$

PROOF. *The proof is similar to Lemma 21 with only the following differences:*

(1) Leave out the part

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi$$

of the disjunction, and
(2) Substitute $(t_2\|t_3)$ for X everywhere. $\square$

Now we show an analogous property for parallel composition.

Lemma 23 *Let t_1, t_2 be process terms, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t_1, t_2\} \cup terms(\Phi)$.*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t_1\|t_2 \models \Diamond_C\Phi \iff F$$

PROOF. *by induction on d.*
Φ has the form $(T^+ \wedge -T^- \wedge \bigwedge_{i\in I} \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} \neg\Diamond_{D_j}\Psi_j)$ for some $T^+, T^- \subset \mathcal{T}$ and $\Phi_i, \Psi_j \in \mathcal{F}_{d-1}^c$.

If $|T^+| \geq 2$ then $\Diamond_C\Phi$ is equivalent to false.

Now we consider the case that $T^+ = \{t\}$ for some term t. If $t \in T^-$ then $t_1\|t_2 \models \Diamond_C\Phi$ is equivalent to false. Otherwise it is equivalent to

$$\bigvee_{k\in K} \left(\begin{array}{l} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi) \end{array} \right)$$

$$\vee$$

$$\bigvee_{l\in L} \bigvee_{\alpha_1\|\alpha_2=t} \left(\begin{array}{l} t_1 \models \Diamond_{D'_l}(\alpha_1) \wedge t_2 \models \Diamond_{D''_l}(\alpha_2) \wedge \\ \bigwedge_{i\in I} t \models \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} t \models \neg\Diamond_{D_j}\Psi_j \end{array} \right)$$

where the C'_k, C''_k, D'_l, D''_l are the decompositions of C as defined in Def. 10 (case 6). This is the F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{\}$. *Then* $t_1 \| t_2 \models \Diamond_C \Phi$ *is equivalent to*

$$\bigvee_{k \in K} \left(\begin{array}{l} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k} \Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k} \Phi) \end{array} \right)$$
$$\vee$$
$$\bigvee_{l \in L} \exists \sigma_1, \sigma_2, t'_1, t'_2. \left(\begin{array}{l} t_1 \xrightarrow{\sigma_1} t'_1 \wedge t_2 \xrightarrow{\sigma_2} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigwedge_{\alpha_1 \| \alpha_2 \in T^-} (t'_1 \neq \alpha_1 \vee t'_2 \neq \alpha_2) \wedge \\ \bigwedge_{i \in I} t'_1 \| t'_2 \models \Diamond_{C_i} \Phi_i \wedge \bigwedge_{j \in J} t'_1 \| t'_2 \models \neg \Diamond_{D_j} \Psi_j \end{array} \right)$$

where the C'_k, C''_k, D'_l, D''_l *are the decompositions of* C *as defined in Def. 10 (case 6). In the base case* $d = 0$ *we have* $I = J = \{\}$ *and don't need the induction hypothesis. For* $d > 0$, *by induction hypothesis, there are formulae* $F_i, G_j \in Dform(d-1, \{t'_1, t'_2\} \cup terms(\Phi), \mathcal{DC})$ *such that* $t'_1 \| t'_2 \models \Diamond_{C_i} \Phi_i \iff F_i$ *and* $t'_1 \| t'_2 \models \Diamond_{D_j} \Psi_j \iff G_j$. *Now we transform the expression into disjunctive normal form. We define the set Func of all functions*

$$f : \{(\alpha_1, \alpha_2) \mid \alpha_1 \| \alpha_2 \in T^-\} \mapsto \{1, 2\}$$

that assign to every pair (α_1, α_2) *s.t.* $\alpha_1 \| \alpha_2 \in T^-$, *a value in* $\{1, 2\}$. *For every* $f \in Func$ *let* $A^1_f := \{\alpha_1 \mid f((\alpha_1, \alpha_2)) = 1\}$ *and* $A^2_f := \{\alpha_2 \mid f((\alpha_1, \alpha_2)) = 2\}$. *Then the expression is equivalent to*

$$\bigvee_{k \in K} \left(\begin{array}{l} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k} \Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k} \Phi) \end{array} \right)$$
$$\vee$$
$$\bigvee_{l \in L} \exists \sigma_1, \sigma_2, t'_1, t'_2. \left(\begin{array}{l} t_1 \xrightarrow{\sigma_1} t'_1 \wedge t_2 \xrightarrow{\sigma_2} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigvee_{f \in Func} (t'_1 \notin A^1_f \wedge t'_2 \notin A^2_f) \wedge \\ \bigwedge_{i \in I} F_i \wedge \bigwedge_{j \in J} \neg G_j \end{array} \right)$$

By transformation to disjunctive normal form there must be finite index sets O *and* $M(o), M'(o), N(o), N'(o)$ *for every* $o \in O$ *and formulae* $\Phi'_n, \Psi'_{n'}, \Phi''_m, \Psi''_{m'} \in \mathcal{F}^c_{d-1}$ *and decomposable constraints* $E_n, E'_{n'}, F_m, F'_{m'} \in \mathcal{DC}$ *s.t. the condition is*

equivalent to

$$\bigvee_{k\in K}\left(\begin{array}{l}(t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi)\end{array}\right)$$

$$\vee$$

$$\bigvee_{l\in L} \exists\sigma_1,\sigma_2,t'_1,t'_2.\left[\begin{array}{l} t_1 \xrightarrow{\sigma_1} t'_1 \wedge t_2 \xrightarrow{\sigma_2} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigvee_{f\in Func}\ (t'_1 \notin A^1_f \wedge t'_2 \notin A^2_f) \wedge \\ \bigvee_{o\in O}\left(\begin{array}{l}\bigwedge_{n\in N(o)} t'_1 \models \Diamond_{E_n}\Phi'_n \bigwedge_{n'\in N'(o)} t'_1 \models \neg\Diamond_{E'_{n'}}\Psi'_{n'} \\ \bigwedge_{m\in M(o)} t'_2 \models \Diamond_{F_m}\Phi''_m \bigwedge_{m'\in M'(o)} t'_2 \models \neg\Diamond_{F'_{m'}}\Psi''_{m'}\end{array}\right)\end{array}\right]$$

This is equivalent to

$$\bigvee_{k\in K}\left(\begin{array}{l}(t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi)\end{array}\right)$$

$$\vee$$

$$\bigvee_{l\in L, f\in Func, o\in O}\left[\begin{array}{l} t_1 \models \Diamond_{D'_l}\left(-A^1_f \wedge \bigwedge_{n\in N(o)} \Diamond_{E_n}\Phi'_n \bigwedge_{n'\in N'(o)} \neg\Diamond_{E'_{n'}}\Psi'_{n'}\right) \wedge \\ t_2 \models \Diamond_{D''_l}\left(-A^2_f \wedge \bigwedge_{m\in M(o)} \Diamond_{F_m}\Phi''_m \bigwedge_{m'\in M'(o)} \neg\Diamond_{F'_{m'}}\Psi''_{m'}\right)\end{array}\right]$$

This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

4.2 The Tableau System

We show the decidability of the model checking problem for PAD and $EF^{=}_{DC}$ by induction on the nesting-depth d of the formula. We describe a tableau system that solves the model checking problem for formulae $\Diamond_C\Phi$ with $\Phi \in \mathcal{F}^c_d$ under the condition that we can already solve the problem for formulae $\Diamond_C\Psi$ with $\Psi \in \mathcal{F}^c_{d-1}$. This is because we use properties of the form $t' \models \Diamond_C\Psi$ for $\Psi \in \mathcal{F}^c_{d-1}$ as side conditions in the construction of the tableau. By induction hypothesis we can assume this. In the base case of $d = 0$ the condition is trivially satisfied, as $\mathcal{F}^c_{-1} = \{\}$.

Every node in the tableau is a set of expressions of the form $t \vdash \Phi$, where t is a process term and Φ an $EF^{=}_{DC}$-formula. We use the symbol $\vdash$ in the tableau instead of $\models$. The expression $t \vdash \Phi$ means that one attempts to prove the property $t \models \Phi$. The meaning of $t \models \Phi$ is defined semantically (Def. 8). The sets of expressions that form the tableau nodes are denoted by Γ and interpreted as sets of subgoals that should be proved. These subgoals are interpreted conjunctively. The branches in the tableau are interpreted

disjunctively, so the tableau is successful iff there is at least one successful branch. Every branch in the tableau can be seen as an attempt to construct a proof.

The following tableau rules are meant to be applied to a problem of the form $t \models \Diamond_C \Phi$ with $\Phi \in \mathcal{F}_d^c$. In the rules Induct1–Induct4 we apply the induction hypothesis that we can already solve the problem for formulae of a smaller nesting-depth.

SEQ1 $$\frac{\{t.X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$$ where F is from Lemma 21

SEQ2 $$\frac{\{t_1.(t_2 \| t_3) \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$$ where F is from Lemma 22

PAR $$\frac{\{t_1 \| t_2 \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$$ where F is from Lemma 23

STEP1 $$\frac{\{X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{X \vdash \Phi\} \cup \Gamma \quad \{\bigvee_{i \in I_1} t_1 \vdash \Diamond_{D_1^i} \Phi\} \cup \Gamma \ \ldots \ \{\bigvee_{i \in I_n} t_n \vdash \Diamond_{D_n^i} \Phi\} \cup \Gamma}$$
if $C(\lambda)$, where λ is the empty sequence, $(X \xrightarrow{a_k} t_k) \in \Delta$, $k = 1, \ldots, n$ and the D_k^i are the decompositions of C (Def. 10 (case 4)).

STEP2 $$\frac{\{X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{\bigvee_{i \in I_1} t_1 \vdash \Diamond_{D_1^i} \Phi\} \cup \Gamma \quad \ldots \quad \{\bigvee_{i \in I_n} t_n \vdash \Diamond_{D_n^i} \Phi\} \cup \Gamma}$$
if not $C(\lambda)$, $(X \xrightarrow{a_k} t_k) \in \Delta$, $k = 1, \ldots, n$
and the D_k^i are decompositions of C (Def. 10 (case 4)).

Unsat $$\frac{\{t \vdash \Diamond_C \Phi\} \cup \Gamma}{\{\mathit{false}\}}$$ if C is unsatisfiable

conj1 $$\frac{\{t \vdash \Phi \wedge \Psi\} \cup \Gamma}{\{t \vdash \Phi, t \vdash \Psi\} \cup \Gamma}$$

conj2 $$\frac{\{F \wedge G\} \cup \Gamma}{\{F, G\} \cup \Gamma}$$

disj1 $$\frac{\{t \vdash \Phi \vee \Psi\} \cup \Gamma}{\{t \vdash \Phi\} \cup \Gamma \quad \{t \vdash \Psi\} \cup \Gamma}$$

disj2 $\quad \dfrac{\{F \vee G\} \cup \Gamma}{\{F\} \cup \Gamma \quad \{G\} \cup \Gamma}$

Induct1 $\quad \dfrac{\{t \vdash \Diamond_C \Psi\} \cup \Gamma}{\Gamma}$ $\quad$ if $\Psi \in \mathcal{F}^c_{d-1}$ and $t \models \Diamond_C \Psi$

Induct2 $\quad \dfrac{\{t \vdash \Diamond_C \Psi\} \cup \Gamma}{\{false\}}$ $\quad$ if $\Psi \in \mathcal{F}^c_{d-1}$ and not $t \models \Diamond_C \Psi$

Induct3 $\quad \dfrac{\{t \vdash \neg\Diamond_C \Psi\} \cup \Gamma}{\Gamma}$ $\quad$ if $\Psi \in \mathcal{F}^c_{d-1}$ and not $t \models \Diamond_C \Psi$

Induct4 $\quad \dfrac{\{t \vdash \neg\Diamond_C \Psi\} \cup \Gamma}{\{false\}}$ $\quad$ if $\Psi \in \mathcal{F}^c_{d-1}$ and $t \models \Diamond_C \Psi$

Term1 $\quad \dfrac{\{t \vdash T^+\} \cup \Gamma}{\Gamma}$ $\quad$ if $T^+ = \{t\}$ or $T^+ = \{\}$

Term2 $\quad \dfrac{\{t \vdash T^+\} \cup \Gamma}{\{false\}}$ $\quad$ if $T^+ \neq \{\} \wedge T^+ \neq \{t\}$

Term3 $\quad \dfrac{\{t \vdash -T^-\} \cup \Gamma}{\Gamma}$ $\quad$ if $t \notin T^-$

Term4 $\quad \dfrac{\{t \vdash -T^-\} \cup \Gamma}{\{false\}}$ $\quad$ if $t \in T^-$

In order to avoid any unnecessary growth of the proof tree, we define that the rules with names in capital letters (PAR, SEQ1, SEQ2, STEP1 and STEP2) have a **lower** precedence than the other rules. So in the construction of a branch of the proof tree we only use such a rule if none of the others is applicable.

Lemma 24 *For any instance of a tableau-rule, the antecedent is true iff at least one of the succedents is true.*

PROOF. *This follows immediately from the definition of the tableau-rules and Lemma 21, Lemma 22 and Lemma 23.*

Definition 25 (Termination conditions)
A node in the tableau consisting of a set of formulae Γ *is a* terminal node *if one of the following conditions is satisfied:*

(1) $\Gamma = \{\}$
(2) *false* $\in \Gamma$.
(3) *There is a previous node in the same branch that is marked with the same set* Γ.

Terminal nodes of type 1 are successful, while terminal nodes of types 2,3 are unsuccessful.

The construction of a branch of the tableau stops when a terminal node is reached. The branch is successful if this terminal node is successful. The tableau is successful if there is at least one successful branch.

The intuition is that every branch in the tableau is an attempt to construct a proof. A terminal node of type 1 means that all subgoals have been solved. A terminal node of type 2 means that this attempt to construct a proof failed. A terminal node of type 3 means that the proof is 'running in circles'. If there is a proof, then it can be found elsewhere in the tableau by a shorter branch.

The construction of the tableau starts with a root-node of the form $\{t \vdash \Diamond_C \Phi\}$ where t is a process term and $\Phi \in \mathcal{F}_d^c$. The tableau for a given root is not unique, because the sequents are sets of expressions and the element to which a rule is applied is chosen nondeterministically. However, all tableaux are equivalent semantically, because the order in which subgoals are solved does not matter.

4.3 Decidability

In this section we show that the tableau system of the previous section is sound and complete and produces only finite tableaux for any given root. Thus it yields a decision procedure for the model checking problem for PAD and $EF_{DC}^{=}$.

Lemma 26 *If the root node has the form* $\{t \vdash \Diamond_C \Phi\}$*, for* $\Phi \in \mathcal{F}_d^c$*, then for every node in a tableau with this root at least one of the following conditions is satisfied:*

(1) A tableau rule is applicable
(2) The node is a terminal node.

PROOF. *The only problematic cases are the expressions of the form* $t \vdash \neg\Diamond_C \Phi$*. If such an expression occurs, then it must be due to the rules SEQ1, SEQ2 or STEP1. By definition of these rules and Lemma 21 and Lemma 22 we know that* $\Phi \in \mathcal{F}_{d-1}^c$*. Then the rules Induct3 or Induct4 are applicable,*

because we assumed (by induction hypothesis) that we can already solve the problem for formulae of a smaller nesting depth.

Lemma 27 *The tableau is finite for every instance of the model checking problem.*

PROOF. *If only process terms of a bounded size are used as atomic propositions, then there are only finitely many formulae in $\mathcal{F}_d^c$ for any fixed d. The tableau rules and the proofs of Lemmas 21, 22 and 23 show that this precondition is satisfied. Any set $\mathcal{DC}$ of decomposable constraints is finite. There are only finitely many rules $(t_1 \xrightarrow{a} t_2) \in \Delta$ with only finitely many subterms of the terms t_2. So there are only finitely many different sets of expressions of the form $t \vdash \Phi$ in the tableau. Therefore the branches of the tableau can only have finite length, because of termination condition 3. Since the tableau is finitely branching, the result follows.*

Now we prove the soundness and completeness of the tableau. The following lemma shows the soundness.

Lemma 28 *Let $\Phi \in \mathcal{F}_d^c$ and $C \in \mathcal{DC}$. If there is a successful tableau with root $\{t \vdash \Diamond_C \Phi\}$, then $t \models \Diamond_C \Phi$.*

PROOF. *A successful tableau has a successful branch ending with a node marked by the empty set of expressions. Since these sets are interpreted conjunctively this node is true. By repeated application of Lemma 24 all its ancestor-nodes must be true and thus the root-node must be true.*

We need some new definitions to show the completeness of the tableau system.

Definition 29 *A* valid sequent Γ *in a tableau is a set of expressions which evaluate to true.*

For example if $(t \vdash \Diamond_C \Phi) \in \Gamma$ then $t \models \Diamond_C \Phi$. If $(F \wedge G) \in \Gamma$ then F and G evaluate to true.

It follows from the construction of the tableau system that every expression in a valid sequent is a disjunction of conjunctions of expressions of the form $t \vdash \Diamond_C \Phi$ or $t \vdash \neg\Diamond_C \Phi$.

Now we define a total order on valid sequents.

Definition 30 *For an expression $t \vdash \Diamond_C \Phi$ with $t \models \Diamond_C \Phi$ we define*

$$xnorm(t \vdash \Diamond_C \Phi) := min\{length(\sigma) \mid t \xrightarrow{\sigma} t' \in [\![\Phi]\!] \wedge C(\sigma)\}$$

and

$$ynorm(t \vdash \Diamond_C \Phi) := size(t)$$

For an expression F in a valid sequent we define

$xnorm(F) := max\{xnorm(t \vdash \Diamond_C \Phi) \mid t \vdash \Diamond_C \Phi$ *is subterm of* F, $nd(\Phi) = d\}$

and

$ynorm(F) := max\{ynorm(t \vdash \Diamond_C \Phi) \mid t \vdash \Diamond_C \Phi$ *is subterm of* F, $nd(\Phi) = d\}$

and

$$znorm(F) := size(F)$$

where $size(F)$ is just the number of letters/symbols needed to write F. The norm of F is a triple, which is defined by

$$norm(F) := (xnorm(F), ynorm(F), znorm(F))$$

These norms are ordered lexicographically. The order is well-founded.

For a valid sequent Γ let

$$\gamma_{x,y,z} := |\{F \in \Gamma \mid norm(\Gamma) = (x, y, z)\}|$$

Since Γ is valid and finite, there is a largest x s.t. $\gamma_{x,y,z} \neq 0$ for some y, z. This largest x will be called x_{max}. It depends on Γ. Also for every $x \leq x_{max}$ there is a largest y (called $y(x)$) s.t. $\gamma_{x,y,z} \neq 0$ for some z. Finally, for every x, y there is a largest $z(x, y)$ s.t. $\gamma_{x,y,z} \neq 0$.

We define a well-founded ordering on valid sequents. Let Γ and Γ' be two valid sequents and $\gamma_{x,y,z}$ and $\gamma'_{x,y,z}$ be defined as above. Then

$$\Gamma < \Gamma' \;:\Leftrightarrow\; \exists(x,y,z).\gamma_{x,y,z} < \gamma'_{x,y,z} \;\wedge\; \forall(x',y',z') \geq_{lex} (x,y,z).\ \gamma_{x',y',z'} = \gamma'_{x',y',z'}$$

The intuition is that if a tableau-rule is applied to a valid sequent Γ, then there is at least one valid succedent sequent that is smaller. This is because an expression $F \in \Gamma$ is replaced with several others with a lower norm. Since the ordering is well-founded, the process must eventually terminate.

Note that these definitions do not apply to non-valid sequents.

Lemma 31 *Let Γ be a valid sequent. Then every tableau with root Γ has at least one successful branch that ends with the empty sequent.*

PROOF. *By Lemma 24 every tableau with root Γ has at least one branch that only contains valid sequents. Choose one such branch of minimal length. We show that the order of the sequents on this branch must strictly decrease. We do this by showing that every application of a tableau rule to a valid sequent yields a smaller sequent.*

SEQ1,SEQ2 *It follows from the construction of the expressions in Lemma 21 and Lemma 22 that in these expressions one of two cases holds:*
(1) The remaining sequence is shorter (lower xnorm) or
(2) The remaining sequence has the same length and the terms are smaller (lower ynorm).
Thus the succedent sequent is smaller.

PAR *It follows from the construction of the expression in Lemma 23 that the terms are always smaller (since t_1, t_2 are smaller than $t_1||t_2$). The xnorm is the same or smaller and the ynorm is smaller. Thus the succedent is smaller.*

STEP1,STEP2 *Here we have two sub-cases:*
- *In the first branch of the rule STEP1 the sequence has length 0. In the succedent the xnorm and ynorm are the same, but the znorm is smaller.*
- *In the other branches of STEP1 and all branches of STEP2 we choose the valid succedent that corresponds to the shortest sequence that leads to a state that satisfies Φ. In this succedent the sequence is shorter and thus the xnorm is smaller.*

In both cases the succedent is smaller.

Unsat *This rule is never applied in this branch, because all sequents are valid.*

conj1,conj2 *For these rules the succedent is smaller, because the znorm decreases.*

disj1,disj2 *For these rules the succedent is smaller, because the znorm decreases.*

Induct,Term *For the rules Induct1,Induct3 and Term1,Term3 the succedent must be smaller, because expressions are removed from the sequent. The rules Induct2,Induct4,Term2,Term4 are never applied in this branch, because all sequents are valid.*

The construction of this branch cannot be stopped by termination condition 3, because the order strictly decreases. Since the order of the sequents strictly decreases on this branch, it must eventually end with the empty sequent and thus it is successful.

Corollary 32 *If $t \models \Diamond_C \Phi$ for $\Phi \in \mathcal{F}_d^c$ and $C \in \mathcal{DC}$ then every tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*

PROOF. *The root-sequent is valid. By Lemma 31 every tableau must have a branch that ends with the empty sequent. This branch is successful and thus the tableau is successful.*

Lemma 33 *Let t be a process term, Δ a PAD, $\Phi \in \mathcal{F}^c_d$, $\mathcal{DC}$ a set of decomposable constraints and $C \in \mathcal{DC}$. Then the following conditions are equivalent:*

- $t \models \Diamond_C \Phi$
- *A tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*
- *Every tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*

PROOF. *Directly from Lemma 28 and Corollary 32.*

Theorem 34 *The model checking problem for $EF^{=}_{DC}$ and PAD is decidable.*

PROOF. *By Lemma 19 it suffices to prove decidability for formulae of the form $\Diamond_C \Phi$ with Φ in $\mathcal{F}^c_d$ for any d. We prove this by induction on d. By Lemma 33 and Lemma 27 it suffices to construct a finite tableau. During the construction we must decide problems of the form $t' \models \Diamond_C \Psi$ for $\Psi \in \mathcal{F}^c_{d-1}$. In the base case $d = 0$ this is trivial, since $\mathcal{F}^c_{-1} = \emptyset$. For $d > 0$ this is possible by induction hypothesis.*

Since EF is weaker than $EF^{=}_{DC}$, we get the following corollary.

Corollary 35 *Model checking PAD with EF is decidable.*

5 Example

In this section we describe a small example of the model checking problem for EF and PAD. The PAD-process is described by the following set of rules Δ:

$$\begin{aligned} X &\xrightarrow{a} (Y \| X).Z \\ Y &\xrightarrow{b} \epsilon \\ X.Z &\xrightarrow{c} X \end{aligned}$$

The initial state is X. By using the algorithm derived from the tableau system described in Section 4 we can show a property of the process X.

$$X \models \Box\Diamond\langle a\rangle true \ \wedge \ \Box\Diamond\langle c\rangle true \ \wedge \ \neg\Diamond\langle a\rangle\langle c\rangle true$$

This means that process X can always get back into states where it can do action "a" or action "c", but never a "c" directly after an "a".

6 Parallel Pushdown Automata

Parallel Pushdown Automata (PPDA) are defined as the pushdown extension of BPP. They are the class of systems that can be described by a synchronization of a BPP with a finite-state system. In the framework of PRS they can be described as follows: Let $R := \{X_1, \ldots, X_k\} \subset \mathit{Const}$ be the process constants that represent the states in the finite state system. Then a PPDA is a PRS where all rules in Δ have the form

$$X_i \| Y \xrightarrow{a} X_j \| t$$

where $X_i, X_j \in R$, $Y \in \mathit{Const}$ and $t \in P$ is a parallel composition of constants that does not contain any constants from R. This is a subclass of Petri net (type (P, P)) rules. In the case of sequential composition the same construction yields pushdown automata, which are equivalent to (S, S)-PRS [8]. However, PPDA are slightly weaker than Petri nets w.r.t. bisimulation.

We prove that model checking with EF is undecidable for PPDA by showing that the proof of undecidability for Petri nets carries over to PPDA. Undecidability of model checking Petri nets with EF was first proved by Esparza in [10]. The proof there contains a slight error, which was corrected in [11]. The idea is to prove undecidability by reduction from the *reachability set containment problem*.

Reachability set containment

Instance: Two Petri nets N_1 and N_2 having the same number of places and a bijection f between the sets of places of N_1 and N_2. f can be extended to a bijection on markings in the obvious way.

Question: Is it true that for every reachable marking M of N_1, $f(M)$ is a reachable marking of N_2 ?

Rabin showed that this problem is undecidable by reduction of Hilbert's 10th problem. Later Jančar [13,14] gave a more direct proof by a reduction from the halting problem for counter machines.

We sketch the reduction of the reachability set containment problem to the model checking problem. It is similar to the one in [11], but slightly simpler. We assume that the transitions in the Petri nets N_1, N_2 are not labeled with atomic actions.

Figure 2 illustrates the following construction.

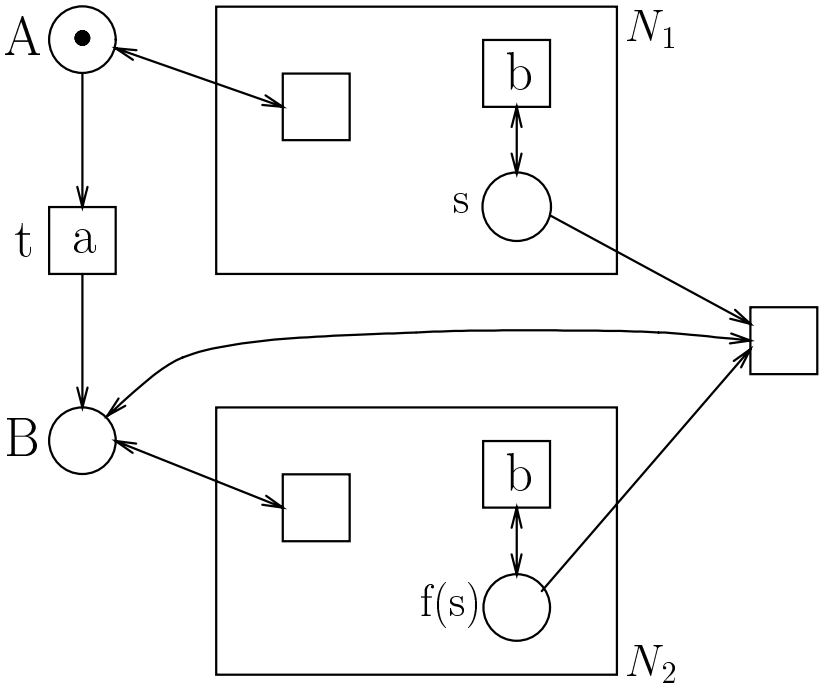


Fig. 2. Reducing reachability set containment to model checking with EF

(1) Put N_1 and N_2 side by side.
(2) Add a place A and arcs from A to every transition in N_1 and back. Put one token on A.
(3) Add a new transition t and a place B and arcs from A to t and from t to B. The transition t is labeled with the atomic action a. Place B is initially unmarked.
(4) Add arcs from B to every transition in N_2 and back.
(5) For every pair of places $(s, f(s))$ add a transition t_s and arcs from s to t_s, $f(s)$ to t_s, B to t_s and t_s to B.
(6) For every place s in N_1 add a transition t'_s labeled with action b and arcs from s to t'_s and back. Do the same for N_2.

Proposition 36 *An instance of the reachability set containment problem has answer 'yes' iff the newly constructed Petri net satisfies the EF-formula*

$$\Box(\neg a \ \vee \ \Diamond(\neg a \ \wedge \ \neg b))$$

Now we construct a PPDA that weakly simulates the Petri net of Figure 2. Assign a unique process constant to every place in this net. Every transition t in the net then corresponds to a rule $Y_1 \| \ldots \| Y_n \xrightarrow{a} Y'_1 \| \ldots Y'_m$. (These constants need not be pairwise different.) Now replace every such rule by the following rules:

$$
\begin{aligned}
X\|Y_1 &\xrightarrow{\tau} X_1 \\
X_i\|Y_{i+1} &\xrightarrow{\tau} X_{i+1} \quad \text{for i=1, } \ldots \text{ ,n-2} \\
X_{n-1}\|Y_n &\xrightarrow{a} X\|Y_1'\| \ldots Y_m'
\end{aligned}
$$

where τ is a new action. The X_j are states of the finite control and are specific for every transition t. X is also a state of the finite control, but it is global and only exists once. The intuition is that these rules simulate the original transition in n steps. The initial state of the PPDA is $X\|I$, where I is the initial state of the Petri net.

Finally, we add one rule $X \xrightarrow{\rho} X$, where ρ is a new action. So action ρ is enabled iff the finite control is in state X. This simulation can get stuck, in case there were not enough tokens there to fire the transition in the first place. We call a state in the simulation 'faithful' if it is not forced to get stuck, i.e., it can get back to a state where ρ is enabled again. This can be expressed by the formula $\Diamond(\rho)$.

Theorem 37 *Model checking with EF is undecidable for PPDA.*

PROOF. *An instance of the containment problem has answer 'yes' iff the PPDA simulation of the net in Figure 2 satisfies*

$$\Box(\neg\Diamond(\rho) \vee \neg a \ \vee \ \Diamond(\rho \ \wedge \ \neg a \ \wedge \ \neg b))$$

7 Conclusion

We have shown decidability of the model checking problem for the branching-time temporal logic EF and the process model PAD. The exact complexity of the problem is an open question. The problem is known to be *PSPACE*-complete for the special cases of BPP [23,19] and BPA [2,24]. Model checking pushdown processes with EF is decidable in *EXPTIME* and *PSPACE*-hard [2,29]. It is even *PSPACE*-hard in the size of the system for a small fixed EF-formula. The complexity for PA and PAD is an open question. The two completely different algorithms for PA by Mayr [21] and by Lugiez and Schnoebelen [17] both have the same extremely high complexity of $\mathcal{O}(tower(n))$. The algorithm for PAD described in this paper is a generalization of the one in [21], but not a generalization of the algorithm for BPP in [19]. The *PSPACE*-algorithm for BPP in [23,19] uses a bounded search, while the algorithm for PAD works by decomposition. For a formula of nesting-depth d the complexity of the algorithm derived from the tableau system is d-times exponential. This is because the tableau has a branching degree that is d-times exponential for EF-formulae of nesting depth d. Also there are d-times exponentially many

different EF-formulae of nesting depth d. So the overall complexity of the algorithm is $O(tower(n))$, where $tower(0) := 0$ and $tower(i+1) := 2^{tower(i)}$.

The best known lower bound for both PAD and PA is *PSPACE*-hardness, but there is a slight difference. For PAD the problem is *PSPACE*-hard in the size of the system for a fixed formula, because this holds for pushdown processes [2] and PAD subsumes pushdown processes. PA does not subsume pushdown processes and the best known lower bound is the same as for BPP: The problem is Σ_d^p-hard for formulae of nesting depth $\leq d$ [19].

Finally, model checking PPDA with EF is undecidable, as shown in Section 6. This implies undecidability for all models in the PRS-hierarchy that are more general than PPDA, i.e., Petri nets, PAN and PRS.

Model	**Complexity of model checking with EF**
finite-state systems	polynomial
BPA	*PSPACE*-complete
pushdown processes	$\in$ *EXPTIME*, *PSPACE*-hard
BPP	*PSPACE*-complete
PA	decidable, *PSPACE*-hard
PAD	decidable, *PSPACE*-hard
PPDA (and higher)	undecidable

As EF is a fragment of CTL and the modal μ-calculus, it is interesting to compare the limits of decidability for these logics. There is another fragment of CTL (and modal μ-calculus) called EG. EG is like EF, except that the diamond operator EF (for some path eventually in the future) is replaced by the operator EG (for some path always in the future). EG is also a fragment of CTL. Model checking with EG is undecidable even for BPP [12]. On the other hand model checking with the modal μ-calculus is decidable (and *EXPTIME*-complete) for pushdown processes [29] and BPA [24]. Thus in the PRS-hierarchy decidability of the weak logic EG coincides with decidability of the much more expressive modal μ-calculus. In Figure 3 we draw the border of decidability of several branching-time logics in the PRS-hierarchy. Model checking is decidable for all models below the border and undecidable for all those above it. Note that almost all branching-time logics have the same decidability border. EF is the only exception. So EF is 'much more decidable' than all other branching-time logics.

It is interesting to compare the decidability results for branching-time logics with the results for linear-time logics like LTL [27] and the linear-time μ-

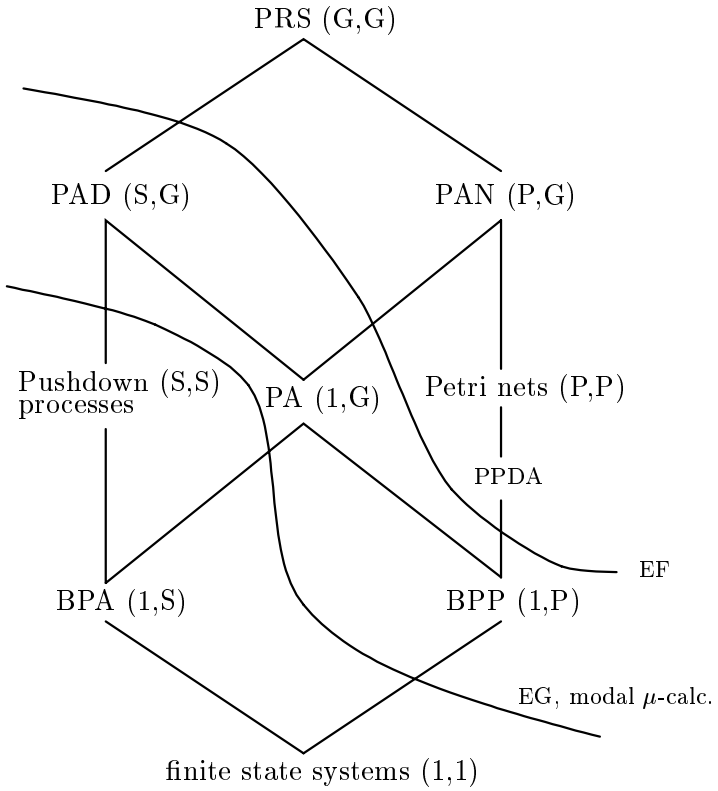


Fig. 3. Limits of the decidability of branching-time logics.

calculus. While model checking PA-processes with EF is decidable, it is undecidable for LTL and the linear-time μ-calculus [3]. For Petri nets the situation is just the opposite. While model checking Petri nets with EF is undecidable, it is decidable for LTL and the linear-time μ-calculus [4,11,7]. In Figure 4 we draw the border of decidability of LTL in the diagram of the PRS-hierarchy.

Acknowledgment: Thanks to Javier Esparza for helpful discussions.

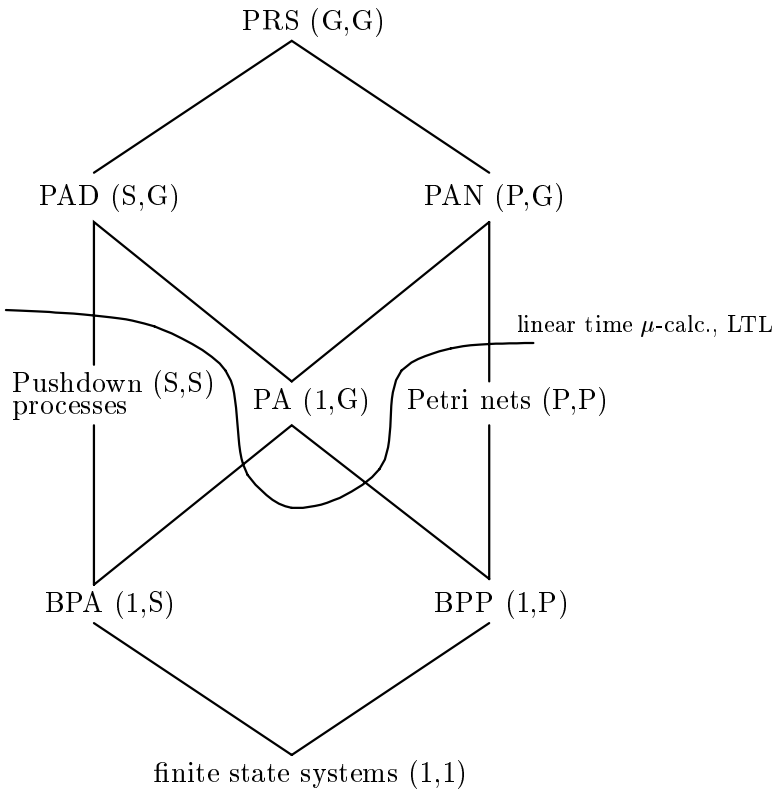


Fig. 4. Limits of the decidability of linear-time logics.

References

[1] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[2] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[3] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[4] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. of STACS'99*, volume 1563 of *LNCS*. Springer Verlag, 1999.

[5] J. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, 1992.

[6] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[7] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 5, 1997.

[8] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[9] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, Edinburgh University, 1993.

[10] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Trees in Algebra and Programming – CAAP'94*, volume 787 of *LNCS*. Springer Verlag, 1994.

[11] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[12] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[13] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proceedings of STACS'94*, volume 775 of *LNCS*. Springer Verlag, 1994.

[14] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[15] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[16] A. Kucera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[17] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*. Springer Verlag, 1998.

[18] R. Mayr. Process rewrite systems. *Information and Computation.* To appear.

[19] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[20] R. Mayr. Combining Petri nets and PA-processes. In Martin Abadi and Takayasu Ito, editors, *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*. Springer Verlag, 1997.

[21] R. Mayr. Model checking PA-processes. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[22] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 7, 1997. Proceedings of Expressiveness in Concurrency (EXPRESS'97).

[23] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-München, 1998.

[24] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.

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

[26] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[27] A. Pnueli. The temporal logic of programs. In *FOCS'77*. IEEE, 1977.

[28] Ph. Schnoebelen. Decomposable and regular languages and the shuffle op erator. In *EATCS Bulletin*, volume 67. European Association of Theoretical Computer Science , February 1999.

[29] I. Walukiewicz. Pushdown processes: games and model checking. In *International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*. Springer Verlag, 1996.

[30] I. Walukiewicz. Pushdown processes: games and model checking. Technical Report RS-96-54, BRICS, Aarhus, Denmark, 1996. Longer version of a CAV'96 paper.

Deciding Bisimulation-Like Equivalences with Finite-State Processes

Petr Jančar [a,1], Antonín Kučera [b,2], Richard Mayr [c]

[a] *Dept. of Computer Science, FEI, Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava, Czech Republic. E-mail:* Petr.Jancar@vsb.cz

[b] *Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic. E-mail:* tony@fi.muni.cz

[c] *Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80290 München, Germany. E-mail:* mayrri@informatik.tu-muenchen.de

Abstract

We show that characteristic formulae for finite-state systems up to bisimulation-like equivalences (e.g., strong and weak bisimilarity) can be given in the simple branching-time temporal logic *EF*. Since *EF* is a very weak fragment of the modal μ-calculus, model checking with *EF* is decidable for many more classes of infinite-state systems. This yields a general method for proving decidability of bisimulation-like equivalences between infinite-state processes and finite-state ones. We apply this method to the class of PAD processes, which strictly subsumes PA and pushdown (PDA) processes, showing that a large class of bisimulation-like equivalences (including, e.g., strong and weak bisimilarity) is decidable between PAD and finite-state processes. On the other hand, we also demonstrate that no 'reasonable' bisimulation-like equivalence is decidable between state-extended PA processes and finite-state ones. Furthermore, weak bisimilarity with finite-state processes is shown to be undecidable even for state-extended BPP (which are also known as 'parallel pushdown processes').

Key words: concurrency, bisimulation, characteristic formulae, infinite-state systems

[1] Supported by the Grant Agency of the Czech Republic, grant number 201/97/0456.

[2] Supported by a Research Fellowship granted by the Alexander von Humboldt Foundation and by a Post-Doc grant GA ČR No. 201/98/P046.

1 Introduction

We study the decidability of bisimulation-like equivalences between infinite-state processes and finite-state ones. The motivation is that the intended behavior of a process is often easy to specify (by a finite-state system), but a 'real' implementation can contain components which are essentially infinite-state (e.g., counters, buffers). The aim of formal verification is to check if the finite-state specification and the infinite-state implementation are semantically equivalent (i.e., bisimilar). First we examine this problem in a general setting, extracting its core in a form of two rather special subproblems (which are naturally not decidable in general). A special variant of this method which works for strong bisimilarity has been described in [14]; here we extend and generalize the concept, obtaining a universal mechanism for proving decidability of bisimulation-like equivalences between infinite-state and finite-state processes. We show that finite-state processes can be encoded up to bisimilarity in formulae of the temporal logic *EF* (more precisely, in a slightly extended version of *EF* which can also express constraints on sequences of atomic actions). Such a formula is called a *characteristic formula* for the given finite-state process. The characteristic formula Θ_f of a finite-state process f has the property that for any (general) process g whose set of actions is contained in the one of f we have that g is bisimilar to f if and only if g satisfies Θ_f. Previous works used the modal μ-calculus to construct characteristic formulae [33]. We show that the much simpler logic *EF* (a fragment of CTL and the modal μ-calculus) suffices. This is significant, because model checking with *EF* is decidable for many more classes of infinite-state systems than with the modal μ-calculus [10,20,24].

Then we apply the designed method to the class of PAD processes (defined in [21]), which properly subsumes all PA and pushdown processes. We prove that a large class of bisimulation-like equivalences (including, e.g., strong and weak bisimilarity) is decidable between PAD and finite-state processes, utilizing previously established results on decidability of the model-checking problem for the logic *EF* [23,20,24,19]. We also provide several undecidability results to complete the picture—we show that any 'reasonable' bisimulation-like equivalence is undecidable between state-extended PA processes and finite-state ones. Moreover, even in the case of state-extended BPP processes (which form a natural subclass of Petri nets) the problem of weak bisimilarity with finite-state processes is undecidable.

Decidability of bisimulation-like equivalences has been intensively studied for various process classes (see [28] for a survey). The majority of the results are about the decidability of strong bisimilarity, e.g., [3,9,8,34,7,16,12].

Strong bisimilarity with finite-state processes is known to be decidable for

(labeled) Petri nets [15], PA, and pushdown processes [14]. Another positive result of this kind is presented in [22], where it is shown that weak bisimilarity is decidable between BPP and finite-state processes. However, weak bisimilarity with finite-state processes is undecidable for Petri nets [13]. In this paper we obtain original positive results for PAD (and hence also PA and PDA) processes, and an undecidability result for state-extended BPP processes. Moreover, all positive results are proved using the same general strategy which can also be adapted to the previously established ones.

In Section 2 we define process rewrite systems, the formalism we use to describe infinite-state systems. In Section 3 we describe the general method for deciding bisimilarity between infinite-state systems and finite-state systems. In Section 4 we use this method to construct characteristic formulae and apply them to prove the main positive decidability result. In Section 5 we prove several undecidability results for strong and weak bisimilarity. In the last section we summarize the results and outline possible future work.

2 Definitions

Transition systems are widely accepted as structures which can exactly define the operational semantics of processes. In the rest of this paper we understand processes as (being associated with) nodes in transition systems of certain types.

Definition 1 *A transition system (TS) $\mathcal{T}$ is a triple $(S, Act, \rightarrow)$ where S is a set of* states*, Act is a finite set of* actions *(or* labels*), and $\rightarrow \subseteq S \times A \times S$ is a* transition relation*.*

We defined Act as a finite set; it is somewhat nonstandard, but we can allow this as all classes of process descriptions we consider generate transition systems of this kind. As usual, we write $s \stackrel{a}{\rightarrow} t$ instead of $(s, a, t) \in \rightarrow$ and we extend this notation to elements of Act^* in an obvious way (we sometimes write $s \rightarrow^* t$ instead of $s \stackrel{w}{\rightarrow} t$ if $w \in Act^*$ is irrelevant). A state t is *reachable* from a state s iff $s \rightarrow^* t$.

Let $Const = \{X, Y, Z, \ldots\}$ be a countably infinite set of *process constants*. The set of (general) *process expressions*, denoted G, is defined by the following abstract syntax equation:

$$E \ ::= \ \varepsilon \mid X \mid E \| E \mid E.E$$

Here X ranges over $Const$ and ε is a special constant that denotes the empty expression. Intuitively, the '.' operator corresponds to a sequential composi-

tion, while the '||' operator models a simple form of parallelism.

In the rest of this paper we do not distinguish between expressions related by *structural congruence* which is the smallest congruence relation over process expressions such that the following laws hold:

- associativity for '.' and '||'
- commutativity for '||'
- 'ε' as a unit for '.' and '||'.

A *process rewrite system* [21] is specified by a finite set Δ of *rules* which are of the form $E \stackrel{a}{\rightarrow} F$, where E, F are process expressions and a is an element of a finite set Act. The sets of process constants which are used in the rules of Δ is denoted by $Const(\Delta)$, and the set of all process expressions built over $Const(\Delta)$ is denoted by $G(\Delta)$.

Each process rewrite system Δ determines a unique transition system where states are process expressions of $G(\Delta)$, the set of labels is Act, and transitions are determined by Δ and the following inference rules (remember that '||' is commutative):

$$\frac{(E \stackrel{a}{\rightarrow} F) \in \Delta}{E \stackrel{a}{\rightarrow} F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E.F \stackrel{a}{\rightarrow} E'.F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E \| F \stackrel{a}{\rightarrow} E' \| F}$$

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of the rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '||' and the '.' operator, respectively. For short, we also use '1' to denote the set of process constants. A hierarchy of process rewrite systems is presented in Figure 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. The set of states of a system Δ which belongs to the subclass determined by (A, B) is then formed by all process expressions of $B \cap G(\Delta)$. It is important to realize that, e.g., every BPA system Δ can also be seen as a PA system, but the sets of states (processes) of Δ are *different* in the two respective cases.

The hierarchy of Figure 1 contains almost all classes of infinite-state systems which have been studied so far; BPA, BPP, and PA processes are well-known [4], PDA correspond to pushdown processes (as proved by Caucal in [6]), PN correspond to Petri nets (see, e.g., [31]), etc. This hierarchy is strict w.r.t. strong bisimulation, i.e., 'higher' classes are strictly more expressive [21].

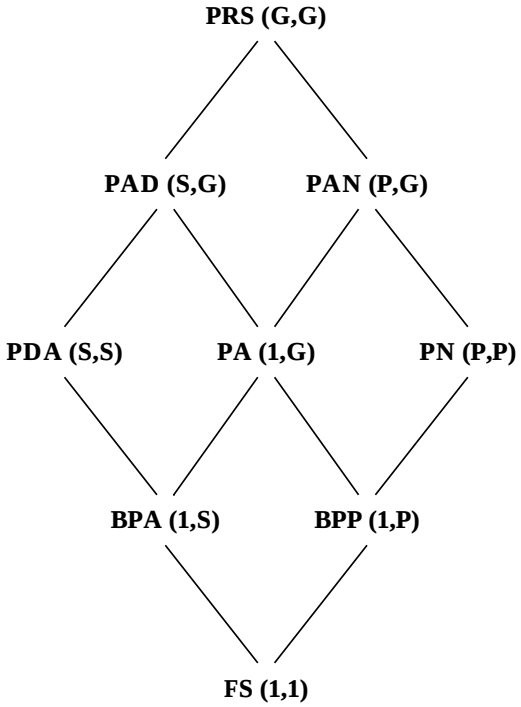


Fig. 1. A hierarchy of process rewrite systems

A convenient way how to extend expressibility of process rewrite systems is to equip them with a finite-state control unit. In order to do that, we first need to introduce the notion of *Step*. Let Δ be a PRS. Observe that each transition $E \stackrel{a}{\rightarrow} F$ is due to some rule $H \stackrel{a}{\rightarrow} K$ of Δ (i.e., H is rewritten to K within E, yielding the expression F). Generally, there can be more than one rule of Δ with this property—if, e.g., $\Delta = \{X \stackrel{a}{\rightarrow} X \| Y, Y \stackrel{a}{\rightarrow} Y \| Y\}$, then the transition $X \| Y \stackrel{a}{\rightarrow} X \| Y \| Y$ can be derived in one step in two different ways. For each transition $E \stackrel{a}{\rightarrow} F$ we denote the set of all rules of Δ which allow to derive the transition in one step by $\mathit{Step}(E \stackrel{a}{\rightarrow} F)$.

A *state-extended PRS* (StExt(PRS)) is a triple (Δ, Q, BT) where Δ is a PRS, Q is a finite set of *control states*, and $BT \subseteq \Delta \times Q \times Q$ is a set of *basic transitions*. The transition system generated by a state-extended PRS (Δ, Q, BT) has $Q \times G(\Delta)$ as the set of states (its elements are called *state-extended PRS processes*, or StExt(PRS) processes for short), Act is the set of labels, and the transition relation is determined by the following rule: $(p, E) \stackrel{a}{\rightarrow} (q, F)$ iff $E \stackrel{a}{\rightarrow} F$ and there is $H \stackrel{a}{\rightarrow} K \in \mathit{Step}(E \stackrel{a}{\rightarrow} F)$ such that $(H \stackrel{a}{\rightarrow} K, p, q) \in BT$.

This construction also applies to the aforementioned subclasses of PRS. It can (but does not have to) increase the expressive power of a given subclass. For example, if we add a finite-state control to a FS, PDA, or PN process, we obtain a process which can be equivalently described by another FS, PDA, or PN process, respectively (here the word 'equivalent' means 'the same up to isomorphism'). In the other cases, the mentioned extension brings strictly more power—StExt(BPA) are in fact PDA processes, StExt(BPP) form a

proper subclass of PN processes (which is also a proper superclass of BPP), and if we add finite-state control to PA (or to any of its superclasses), we obtain systems with full Turing power. The last fact will be demonstrated in Section 5. Let us note that PRS themselves are not Turing-powerful, because the reachability problem is decidable for them—see [21].

3 A General Method for Bisimulation-Like Equivalences

In this section we design a general method for proving decidability of bisimulation-like equivalences between infinite-state processes and finite-state ones.

Definition 2 *Let $R : Act \to 2^{Act^*}$ be a (total) function, assigning to each action its corresponding set of* responses. *We say that R is* closed under substitution *if the following conditions hold:*

- $a \in R(a)$ *for each* $a \in Act$.
- *If* $b_1 b_2 \ldots b_n \in R(a)$ *and* $w_1 \in R(b_1), w_2 \in R(b_2), \ldots, w_n \in R(b_n)$, *then also* $w_1 w_2 \ldots w_n \in R(a)$.

In order to simplify our notation, we adopt the following conventions in this section:

- $\mathcal{G} = (G, Act, \to)$ always denotes a (general) transition system.
- $\mathcal{F} = (F, Act, \to)$ always denotes a finite-state transition system with k states.
- R always denotes a function from Act to 2^{Act^*} which is closed under substitution.
- N always denotes a decidable binary predicate defined for pairs (s, t) of nodes in transition systems (which will be clear from the context). Moreover, N is reflexive, symmetric, and transitive.

Note that $\mathcal{G}$ and $\mathcal{F}$ have the same set of actions Act. All definitions and propositions which are formulated for $\mathcal{G}$ should be considered as general; if we want to state some specific property of finite-state transition systems, we refer to $\mathcal{F}$. We also assume that $\mathcal{G}$, $\mathcal{F}$, R, and N are defined in a 'reasonable' way so that we can allow natural decidability assumptions on them (e.g., it is decidable whether $g \stackrel{a}{\to} g'$ for all $g, g' \in G$ and $a \in Act$, or whether $w \in R(a)$ for a given $w \in Act^*$, etc.)

Definition 3 *The* extended *transition relation $\Rightarrow \subseteq G \times Act \times G$ is defined as follows: $s \stackrel{a}{\Rightarrow} t$ iff $s \stackrel{w}{\to} t$ for some $w \in R(a)$.*

Definition 4 *A relation $P \subseteq G \times G$ is an* R-N-bisimulation *if whenever*

$(s,t) \in P$, then $N(s,t)$ is true and for each $a \in Act$:

- *If $s \xrightarrow{a} s'$, then $t \stackrel{a}{\Rightarrow} t'$ for some $t' \in G$ such that $(s',t') \in P$.*
- *If $t \xrightarrow{a} t'$, then $s \stackrel{a}{\Rightarrow} s'$ for some $s' \in G$ such that $(s',t') \in P$.*

States $s,t \in G$ are R-N-bisimilar*, written $s \stackrel{RN}{\sim} t$, if there is an R-N-bisimulation relating them.*

Various special versions of R-N-bisimilarity appeared in the literature, e.g., *strong* and *weak* bisimilarity (see [30,26]). The corresponding versions of R (denoted by S and W, respectively) are defined as follows ($\mathbb{N}_0$ denotes the set of all nonnegative integers):

- $S(a) = \{a\}$ for each $a \in Act$.
- $W(a) = \begin{cases} \{\tau^i \mid i \in \mathbb{N}_0\} & \text{if } a = \tau; \\ \{\tau^i a \tau^j \mid i,j \in \mathbb{N}_0\} & \text{otherwise.} \end{cases}$

The 'τ' is a special (silent) action, usually used to model an internal communication. As the predicate N is not used in the definitions of strong and weak bisimilarity, we can assume it is always true (we use T to denote this special case of N in the rest of this paper). One can also argue that the N predicate could be omitted from the definition of R-N-bisimilarity, as it is not employed by any known bisimulation-like equivalence. This is not completely true, as, e.g., the version of strong bisimilarity introduced in [28] uses such a predicate to distinguish between 'terminal' and 'final' states of pushdown processes (in this way it is possible to distinguish between a 'successful' termination caused by emptying the stack, and an 'unsuccessful' one (deadlock) caused by entering a state (p,E), where $E \neq \varepsilon$, from which there are no transitions).

Generally, every R-N-bisimilarity is a refinement of R-T-bisimilarity and this fact also suggests the way how to use the predicate N; its basic purpose is to impose some additional conditions on pairs of states which cannot be specified by R, but which should be satisfied by (pairs of) equivalent states. We illustrate this approach by designing a natural refinement of weak bisimilarity.

Example 5 *It is a well-known fact that weak bisimilarity does not distinguish between a state which cannot emit any action (deadlock), and a state which can emit only an infinite number of silent 'τ' actions (livelock). However, these two behaviors are considered to be* different *in many situations; for example, there are very good reasons to distinguish between deadlock and livelock in the context of operating systems. Therefore, it is natural to ask whether there is some refinement of weak bisimilarity which preserves most of its properties but eliminates the mentioned drawback at the same time. A simple solution is to*

define the D predicate in the following way:

$$D(s,t) \textit{ is true iff } (\mathit{Init}(s) = \emptyset \iff \mathit{Init}(t) = \emptyset)$$

Here $\mathit{Init}(s)$ denotes the set of initial actions*, defined as follows: $\mathit{Init}(s) = \{a \in Act \mid s \xrightarrow{a} s' \textit{ for some } s'\}$. Now W-D-bisimilarity is a good candidate for the equivalence we are looking for; it is very similar to weak bisimilarity, but it distinguishes between deadlock and livelock. As we shall see, W-D-bisimilarity is also* decidable *between PAD processes and finite-state ones.*

The concept of R-N-bisimilarity covers many equivalences which have not been explicitly investigated so far; for example, we can define the function R like this:

- $K(a) = \{a^i \mid i \in \mathbb{N}_0\}$ for each $a \in Act$.
- $L(a) = \{w \in Act^* \mid w \text{ begins with } a\}$.
- $M(a) = \begin{cases} Act^* & \text{if } a = \tau; \\ \{w \in Act^* \mid w \text{ contains at least one } a\} & \text{otherwise.} \end{cases}$

The predicate N can also have various forms. We have already mentioned the 'T' (always true) and 'D' (deadlock equivalence). Another natural example is the 'I' predicate: $I(s,t)$ is true iff $\mathit{Init}(s) = \mathit{Init}(t)$. It is easy to see that, e.g., $\overset{ST}{\sim}$ coincides with $\overset{SI}{\sim}$, while $\overset{WI}{\sim}$ refines $\overset{WD}{\sim}$.

An important example of a bisimulation-like equivalence which cannot be seen as R-N-bisimilarity is *branching bisimilarity* (introduced in [35]). This relation places additional requirements on 'intermediate' nodes that extended transitions pass through, and this brings further difficulties. Therefore, we do not consider branching bisimilarity in our paper.

R-N-bisimilarity can also be defined in terms of the so-called *R-N-bisimulation game*. Imagine that there are two tokens initially placed in states s and t such that $N(s,t)$ is true. Two players, Al and Ex, now start to play a game consisting of a (possibly infinite) sequence of rounds, where each round is performed as follows:

1. Al chooses one of the two tokens and moves it along an arbitrary (but single!) transition, labeled by some $a \in Act$.
2. Ex has to respond by moving the other token along a finite sequence of transitions in such a way that the corresponding sequence of labels belongs to $R(a)$ and the predicate N is true for the pair of states where the tokens lie after Ex finishes his move.

Al wins the R-N-bisimulation game, if after a finite number of rounds Ex cannot respond to Al's final attack. Now it is easy to see that the states s and

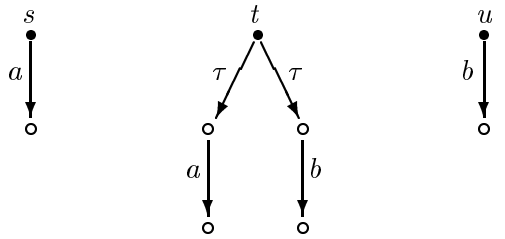


Fig. 2. A transition system considered in Example 6

t are R-N-bisimilar iff Ex has a universal defending strategy (i.e., Ex can play in such a way that Al cannot win).

A natural way how to approximate R-N-bisimilarity is to define the family of relations $\overset{RN}{\sim}_i \subseteq G \times G$, $i \in \mathbb{N}_0$, as follows: $s \overset{RN}{\sim}_i t$ iff $N(s,t)$ is true and Ex has a defending strategy within the first i rounds in the R-N-bisimulation game. However, $\overset{RN}{\sim}_i$ does not have to be an equivalence relation. Moreover, it is not necessarily true that $s \overset{RN}{\sim} t \iff \forall i \in \mathbb{N}_0 : s \overset{RN}{\sim}_i t$.

Example 6 *It is a well-known fact that in the case of weak bisimilarity (i.e., W-T-bisimilarity) the equivalence*

$$s \overset{WT}{\sim} t \iff \forall i \in \mathbb{N}_0 : s \overset{RN}{\sim}_i t$$

does not hold in general ('$\Longleftarrow$' does not have to be valid). Moreover, $\overset{WT}{\sim}_i$ is not transitive for $i \geq 1$. To see this, consider the states s, t, u in the transition system of Figure 2; we have $s \overset{WT}{\sim}_1 t$ and $t \overset{WT}{\sim}_1 u$, but $s \overset{WT}{\not\sim}_1 u$.

Now we show how to overcome those difficulties; to do this, we first introduce the *extended* R-N-bisimulation relation:

Definition 7 *A relation $P \subseteq G \times G$ is an* extended *R-N-bisimulation if whenever $(s,t) \in P$, then $N(s,t)$ is true and for each $a \in Act$:*

- *If $s \overset{a}{\Rightarrow} s'$, then $t \overset{a}{\Rightarrow} t'$ for some $t' \in G$ such that $(s',t') \in P$.*
- *If $t \overset{a}{\Rightarrow} t'$, then $s \overset{a}{\Rightarrow} s'$ for some $s' \in G$ such that $(s',t') \in P$.*

States $s, t \in G$ are extended R-N-bisimilar *if there is an extended R-N-bisimulation relating them.*

Naturally, we can also define the extended R-N-bisimilarity by means of the extended R-N-bisimulation game; we simply allow Al to use the 'long' moves (i.e., Al can play the same kind of moves as Ex). Moreover, we can define the family of approximations of extended R-N-bisimilarity in the same way as in the case of R-N-bisimilarity—for each $i \in \mathbb{N}_0$ we define the relation $\overset{RN}{\simeq}_i \subseteq G \times G$ as follows: $s \overset{RN}{\simeq}_i t$ iff $N(s,t)$ is true and Ex has a defending strategy within the first i rounds in the extended R-N-bisimulation game where tokens are

initially placed in s and t.

Lemma 8 *Two states s, t of $\mathcal{G}$ are R-N-bisimilar iff s and t are extended R-N-bisimilar.*

PROOF. Every extended R-N-bisimulation is also an R-N-bisimulation; here we need that $a \in R(a)$ for each $a \in Act$. Conversely, every R-N-bisimulation is also an extended R-N-bisimulation; each extended transition is a finite sequence of transitions, hence we can concatenate 'responses' to those individual transitions, obtaining a valid response to the original extended transition. Here we need the second requirement of Definition 2, that the relation R is closed under substitution. □

Lemma 9 *The following properties hold:*

(1) $\overset{RN}{\simeq}_i$ is an equivalence relation for each $i \in \mathbb{N}_0$.
(2) Let s, t be states of $\mathcal{G}$. Then $\forall i \in \mathbb{N}_0 : s \overset{RN}{\sim}_i t$ iff $\forall i \in \mathbb{N}_0 : s \overset{RN}{\simeq}_i t$.

PROOF.

(1) For the first part, reflexivity and symmetry are obvious. Transitivity follows from the condition that the relation R is closed under substitution.
(2) It follows from the definition of $\overset{RN}{\simeq}$ that $s \overset{RN}{\simeq}_i t \implies s \overset{RN}{\sim}_i t$. Hence, it suffices to realize that if $s \overset{RN}{\not\simeq}_i t$, then $s \overset{RN}{\not\sim}_j t$ for some $j \in \mathbb{N}_0$—as Al can force his win using i 'long' moves and each of those moves consists of a finite number of 'short' moves, Al could actually 'decompose' his attacks, playing only (a finite number of) short moves. □

Remark 10 *For all states s, t of $\mathcal{G}$ and $i \in \mathbb{N}_0$ we have that if $s \overset{RN}{\simeq}_i t$ then also $s \overset{RN}{\sim}_i t$. However, there is no 'reverse correspondence'—it can be easily shown that for arbitrarily large j the implication $s \overset{RN}{\sim}_j t \implies s \overset{RN}{\simeq}_1 t$ is generally invalid (the implication is invalid even in the case when t is a state in a one-state TS). See Section 5 for details.*

Now we examine some special properties of R-N-bisimilarity on finite-state transition systems (remember that $\mathcal{F}$ is a finite-state TS with k states).

Lemma 11 *Two states s, t of $\mathcal{F}$ are R-N-bisimilar iff $s \overset{RN}{\simeq}_{k-1} t$.*

PROOF. As $\mathcal{F}$ has k states and $\overset{RN}{\simeq}_{i+1}$ refines $\overset{RN}{\simeq}_i$ for each $i \in \mathbb{N}_0$, we have that $\overset{RN}{\simeq}_{k-1} = \overset{RN}{\simeq}_k$, hence $\overset{RN}{\simeq}_{k-1} = \overset{RN}{\simeq}$. □

Theorem 12 *States $g \in G$ and $f \in F$ are R-N-bisimilar iff the following conditions hold:*

1. $g \overset{RN}{\simeq}_k f$.
2. *For each state g' which is reachable from g there is a state $f' \in F$ such that $g' \overset{RN}{\simeq}_k f'$.*

PROOF.
'$\Longrightarrow$': Obvious.
'$\Longleftarrow$': We prove that the relation

$$P = \{(g', f') \mid g \to^* g' \text{ and } g' \overset{RN}{\simeq}_k f'\}$$

is an extended *R-N*-bisimulation. Let $(g', f') \in P$ and let $g' \overset{a}{\Rightarrow} g''$ for some $a \in Act$ (the case when $f' \overset{a}{\Rightarrow} f''$ is handled in the same way). By definition of $\overset{RN}{\simeq}_k$, there is an f'' such that $f' \overset{a}{\Rightarrow} f''$ and $g'' \overset{RN}{\simeq}_{k-1} f''$. It suffices to show that $g'' \overset{RN}{\simeq}_k f''$; as $g \to^* g''$, there is a state $\overline{f}$ of $\mathcal{F}$ such that $g'' \overset{RN}{\simeq}_k \overline{f}$. By transitivity of $\overset{RN}{\simeq}_{k-1}$ we have $\overline{f} \overset{RN}{\simeq}_{k-1} f''$, hence $\overline{f} \overset{RN}{\simeq}_k f''$ (due to Lemma 11). Now $g'' \overset{RN}{\simeq}_k \overline{f} \overset{RN}{\simeq}_k f''$ and thus $g'' \overset{RN}{\simeq}_k f''$ as required. Clearly $(g, f) \in P$ and the proof is finished. □

Remark 13 *We have already mentioned that the equivalence*

$$s \overset{RN}{\sim} t \iff \forall i \in \mathbb{N}_0 : s \overset{RN}{\simeq}_i t$$

is generally invalid (e.g., in the case of weak bisimilarity). However, as soon as we assume that t is a state in a finite-state transition system, the equivalence holds. This is an immediate consequence of the previous theorem. Moreover, the second part of Lemma 9 says that we could also use the $\overset{RN}{\sim}_i$ approximations on the right-hand side of the equivalence.

The previous theorem in fact says that one can use the following strategy to decide whether $g \overset{RN}{\sim} f$:

1. Decide whether $g \overset{RN}{\simeq}_k f$ (if not, then $g \overset{RN}{\not\sim} f$).
2. Check whether g can reach a state g' such that $g' \overset{RN}{\not\simeq}_k f'$ for *every* state f' of $\mathcal{F}$ (if there is such a g' then $g \overset{RN}{\not\sim} f$; otherwise $g \overset{RN}{\sim} f$).

However, none of these tasks is easy in general. Our aim is to examine both subproblems in detail, keeping the general setting. Hence, we cannot expect any 'universal' (semi)decidability result, because even the problems $g \overset{WT}{\simeq}_1 f$ and $g \overset{WT}{\not\simeq}_1 f$ are not semidecidable in general (see Section 5).

As $\mathcal{F}$ has finitely many states, the extended transition relation $\Rightarrow$ is finite and effectively constructible. Therefore, we can effectively replace the transition relation of $\mathcal{F}$ with its corresponding extended transition relation. Al and Ex can now play only 'short' moves consisting of exactly one transition whenever playing within the modified system $\mathcal{F}$—each such move corresponds to some extended transition of the original system $\mathcal{F}$ and vice versa. This observation leads to the notion of *branching tree*, which allows to 'extract' from $\mathcal{F}$ the information which is relevant for the first k moves in the extended R-N-bisimulation game. The aim of the following definition is to describe all such trees up to isomorphism (remember that Act is a *finite* set).

Definition 14 *For each $i \in \mathbb{N}_0$ we define the set of* Trees *with* depth *at most i (denoted $Tree_i$) inductively as follows:*

- *A Tree with depth* 0 *is any tree with no arcs and a single node (the root) which is labeled by an element of $F \cup \{\bot\}$.*
- *A Tree with depth at most $i+1$ is any directed tree with root r whose nodes are labeled by elements of $F \cup \{\bot\}$, arcs are labeled by elements of Act, which satisfies the following conditions:*
  - *If $r \xrightarrow{a} s$, then the subtree rooted by s is a Tree with depth at most i.*
  - *If $r \xrightarrow{a} s$ and $r \xrightarrow{a} s'$ for $s \neq s'$, then the subtrees rooted by s and s' are not isomorphic.*

It is clear that the set $Tree_j$ is finite for every $j \in \mathbb{N}_0$. More precisely, its cardinality (denoted $NT(j)$) is given by:

- $NT(0) = k + 1$
- $NT(i+1) = (k+1) \cdot 2^{n \cdot NT(i)}$, where $n = card(Act)$

The set $Tree_j$ is effectively constructible for every $j \in \mathbb{N}_0$. As each Tree can be seen as a transition system, we can also speak about *Tree-processes* which are associated with roots of Trees (we do not distinguish between Trees and Tree-processes in the rest of this paper).

Now we introduce special rules which replace the standard ones whenever we consider an extended R-N-bisimulation game with initial state (g, p), where $g \in G$ and p is a Tree process (formally, this is a *different* game—however, it does not deserve a special name in our opinion).

- Al and Ex are allowed to play only 'short' moves consisting of exactly one transition whenever playing within the Tree process p (transitions of Trees correspond to the extended transitions of $\mathcal{F}$).
- The predicate $N(g', p')$, where $g' \in G$ and p' is a state of the Tree process

p, is evaluated as follows:

$$N(g',p') = \begin{cases} \text{true} & \text{if } \mathit{label}(p') = \bot \text{ and} \\ & N(g',f) = \text{false for every } f \in F \\ \text{false} & \text{if } \mathit{label}(p') = \bot \text{ and} \\ & N(g',f) = \text{true for some } f \in F \\ N(g', \mathit{label}(p')) & \text{otherwise} \end{cases}$$

Whenever we write $g \overset{RN}{\simeq}_i p$, where $g \in G$ and p is a Tree process, we mean that Ex has a defending strategy within the first i rounds in the 'modified' extended R-N-bisimulation game. The importance of Tree processes is clarified by the two lemmas below:

Lemma 15 *Let g be a state of $\mathcal{G}$, $j \in \mathbb{N}_0$. Then $g \overset{RN}{\simeq}_j p$ for some $p \in \mathit{Tree}_j$.*

PROOF. We proceed by induction on j:

- $\mathbf{j = 0}$: Then p is a Tree with no arcs and just one node labeled by some $f \in F$ such that $N(g,f)$ is true; if there is no such f, then it is labeled by $\bot$. Clearly $g \overset{RN}{\simeq}_0 p$.
- **Induction step:** We need to construct a Tree p such that $g \overset{RN}{\simeq}_{j+1} p$. The Tree p has a root r whose label is determined in the same way as in the case when $j = 0$. The successors of r are defined by

$$r \overset{a}{\rightarrow} s \text{ iff } g \overset{a}{\Rightarrow} g' \text{ and } g' \overset{RN}{\simeq}_j s$$

  Note that for each g' there is $s \in \mathit{Tree}_j$ such that $g' \overset{RN}{\simeq}_j s$ by induction hypothesis. Thus, we have $g \overset{RN}{\simeq}_{j+1} p$ as required. □

Lemma 16 *Let f be a state of $\mathcal{F}$, $j \in \mathbb{N}_0$, and $p \in \mathit{Tree}_j$ such that $f \overset{RN}{\simeq}_j p$. Then for every state g of $\mathcal{G}$ we have that $g \overset{RN}{\simeq}_j f$ iff $g \overset{RN}{\simeq}_j p$.*

PROOF.
'$\Longrightarrow$': By induction on j:

- $\mathbf{j = 0}$: As $f \overset{RN}{\simeq}_0 p$ and $g \overset{RN}{\simeq}_0 f$, we have that $N(g,f)$ is true and (the root of) p is labeled by some f' such that $N(f,f')$ is true. Hence, $N(g,f')$ is true and $g \overset{RN}{\simeq}_0 p$.
- **Induction step:** Let $f \overset{RN}{\simeq}_{j+1} p$ and $g \overset{RN}{\simeq}_{j+1} f$. We prove that $g \overset{RN}{\simeq}_{j+1} p$. Clearly $N(g, \mathit{label}(p))$ is true (see above). Let $g \overset{a}{\Rightarrow} g'$ (the case when $p \overset{a}{\rightarrow} p'$ can be done similarly). We need to show that $p \overset{a}{\rightarrow} p'$ for some p' with $g' \overset{RN}{\simeq}_j p'$. As $g \overset{RN}{\simeq}_{j+1} f$, there is $f' \in F$ such that $f \overset{a}{\Rightarrow} f'$ and $g' \overset{RN}{\simeq}_j f'$.

Furthermore, as $f \overset{RN}{\simeq}_{j+1} p$ and $f \overset{a}{\Rightarrow} f'$, there is p' such that $p \overset{a}{\rightarrow} p'$ and $f' \overset{RN}{\simeq}_j p'$. To sum up, we have $f' \overset{RN}{\simeq}_j p'$ and $g' \overset{RN}{\simeq}_j f'$, hence $g' \overset{RN}{\simeq}_j p'$ by induction hypotheses.

'$\Longleftarrow$': In a similar way. □

Now we can extract the core of both subproblems which appeared in the previously mentioned general strategy in a (hopefully) nice way by defining two new and rather special problems—the Step-problem and the Reach-problem:

The Step-problem
Instance: (g, a, j, p) where g is a state of $\mathcal{G}$, $a \in Act$, $0 \leq j < k$, and $p \in Tree_j$.
Question: Is there a state g' of $\mathcal{G}$ such that $g \overset{a}{\Rightarrow} g'$ and $g' \overset{RN}{\simeq}_j p$?
A decision algorithm may use an oracle which for any state g'' of $\mathcal{G}$ answers whether $g'' \overset{RN}{\simeq}_j p$.

The Reach-problem
Instance: (g, p) where g is a state of $\mathcal{G}$ and p is a Tree-process of depth $\leq k$.
Question: Is there a state g' of $\mathcal{G}$ such that $g \rightarrow^* g'$ and $g' \overset{RN}{\simeq}_k p$?
A decision algorithm may use an oracle which for any state g'' of $\mathcal{G}$ answers whether $g'' \overset{RN}{\simeq}_k p$.

Formally, the transition system $\mathcal{F}$ should also be given in the instances of the aforementioned problems, as it determines the sets $Tree_j$ and the constant k; we prefer the simplified form to make the following proofs more readable.

Theorem 17 *If the Step-problem is decidable (possibly using the mentioned oracle), then $\overset{RN}{\simeq}_k$ is decidable between all states g and f of $\mathcal{G}$ and $\mathcal{F}$, respectively.*

PROOF. We prove by induction on j that $\overset{RN}{\simeq}_j$ is decidable for every $0 \leq j \leq k$. First, $\overset{RN}{\simeq}_0$ is decidable because the predicate N is decidable. Let us assume that $\overset{RN}{\simeq}_j$ is decidable (hence the mentioned oracle can be used). It remains to prove that if the Step-problem is decidable, then $\overset{RN}{\simeq}_{j+1}$ is decidable as well. We need to introduce two auxiliary finite sets:

- The set of **C**ompatible **S**teps, denoted CS_j^f, is composed exactly of all pairs of the form (a, p), where $a \in Act$ and $p \in Tree_j$, such that $f \overset{a}{\Rightarrow} f'$ for some f' with $f' \overset{RN}{\simeq}_j p$.

- The set of **INC**ompatible **S**teps, denoted $INCS_j^f$, is a complement of CS_j^f w.r.t. $Act \times Tree_j$.

The sets CS_j^f and $INCS_j^f$ are effectively constructible. By definition, $g \stackrel{RN}{\simeq}_{j+1} f$ iff $N(g,f)$ is true and the following conditions hold:

1. If $f \stackrel{a}{\Rightarrow} f'$, then $g \stackrel{a}{\Rightarrow} g'$ for some g' with $g' \stackrel{RN}{\simeq}_j f'$.
2. If $g \stackrel{a}{\Rightarrow} g'$, then $f \stackrel{a}{\Rightarrow} f'$ for some f' with $g' \stackrel{RN}{\simeq}_j f'$.

The first condition in fact says that (g,a,j,p) is a positive instance of the Step-problem for every $(a,p) \in CS_j^f$ (see Lemma 15 and 16). It can be checked effectively due to the decidability of the Step-problem.

The second condition does *not* hold iff $g \stackrel{a}{\Rightarrow} g'$ for some g' such that $g' \stackrel{RN}{\simeq}_j p$ where (a,p) is an element of $INCS_j^f$ (due to Lemma 15 and 16). This is clearly decidable due to the decidability of the Step-problem again. □

It is worth mentioning that the Step-problem is generally semidecidable (provided it is possible to enumerate all finite paths starting in g). However, it does not suffice for semidecidability of $\stackrel{RN}{\simeq}_i$ or $\stackrel{RN}{\not\simeq}_i$ between states of $\mathcal{G}$ and $\mathcal{F}$.

Theorem 18 *Decidability of the Step-problem and the Reach-problem (possibly using the indicated oracles) implies decidability of the problem whether for each g' which is reachable from a given state g of $\mathcal{G}$ there is a state f' of $\mathcal{F}$ with $g' \stackrel{RN}{\simeq}_k f'$.*

PROOF. First, the oracle indicated in the definition of Reach-problem can be used because we already know that decidability of the Step-problem implies decidability of $\stackrel{RN}{\simeq}_k$ between states of $\mathcal{G}$ and $\mathcal{F}$ (see the previous theorem). To finish the proof, we need to define one auxiliary set:

- The set of **INC**ompatible **T**rees, denoted $INCT$, is composed of all $p \in Tree_k$ such that $f \stackrel{RN}{\not\simeq}_k p$ for every state f of $\mathcal{F}$.

The set $INCT$ is finite and effectively constructible. The state g can reach a state g' such that $g' \stackrel{RN}{\not\simeq}_k f$ for every state f of $\mathcal{F}$ (i.e., g is a *negative* instance of the problem specified in the second part of this theorem) iff (g,p) is a positive instance of the Reach problem for some $p \in INCT$ (due to Lemma 15 and 16). □

4 Characteristic Formulae

In this section we show how to apply the previously designed general method to construct characteristic formulae for finite-state systems in the temporal logic $EF_{\mathcal{C}}$ (we show that the Step-problem as well as the Reach-problem can be encoded by $EF_{\mathcal{C}}$ formulae). Consequently, we reduce the problem of R-N-bisimilarity between infinite-state processes and finite-state ones to the model checking problem for $EF_{\mathcal{C}}$. Therefore it is possible to apply decidability results from this area. In this way we prove that a large class of R-N-bisimulation equivalences is decidable between PAD processes and finite-state ones (the class includes all versions of R-N-bisimulation equivalences we defined in this paper and many others). First we define the logic $EF_{\mathcal{C}}$ (it is an extended version of the logic EF [10] with constraints on sequences of actions). Let $\mathcal{C}$ be a finite set of unary predicates on sequences of atomic actions. The formulae of $EF_{\mathcal{C}}$ have the following syntax (where $a \in Act$ and $C \in \mathcal{C}$):

$$\Phi ::= true \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \langle a \rangle \Phi \mid \Diamond_C \Phi$$

Let $\mathcal{T} = (S, Act, \rightarrow)$ be a transition system. The denotation $[\![\Phi]\!]$ of a formula Φ is a set of states of $\mathcal{T}$ where the formula *holds*; it is defined as follows (sequences of atomic actions are denoted by w):

$$\begin{aligned}
[\![true]\!] &:= S \\
[\![\neg\Phi]\!] &:= S - [\![\Phi]\!] \\
[\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\
[\![\langle a \rangle \Phi]\!] &:= \{ s \in S \mid \exists s' \in S.\ s \xrightarrow{a} s' \ \wedge \ s' \in [\![\Phi]\!] \} \\
[\![\Diamond_C \Phi]\!] &:= \{ s \in S \mid \exists w, s'.\ s \xrightarrow{w} s' \ \wedge \ C(w) \ \wedge \ s' \in [\![\Phi]\!] \}
\end{aligned}$$

The predicates of $\mathcal{C}$ are used to express constraints on sequences of actions. An instance of the *model checking problem* is given by a state s in S and an $EF_{\mathcal{C}}$ formula Φ. The question is whether $s \in [\![\Phi]\!]$. This property is also denoted by $s \models \Phi$.

A *characteristic formula* Θ_f for a finite-state process f w.r.t. R-N-bisimulation has the property that for every (general) process g whose set of actions is contained in the set of actions of f we have

$$g \overset{RN}{\sim} f \iff g \models \Theta_f$$

For every R-N-bisimulation we define the set of predicates $\mathcal{R}$ as follows:

$$\mathcal{R} = \{ C_a \mid a \in Act, C_a(w) \Longleftrightarrow w \in R(a) \} \cup \{ true, false \}$$

As usual, we write $\Diamond\Phi$ instead of $\Diamond_{true}\Phi$.

Let us fix a general TS $\mathcal{G} = (G, Act, \to)$ and a finite-state TS $\mathcal{F} = (F, Act, \to)$ with k states in the same way as in the previous section. We show how to encode the Step and the Reach problems by $EF_{\mathcal{R}}$ formulae. The first difficulty is the N predicate. Although it is decidable, this fact is generally of no use because we cannot make any assumptions on 'strategies' of model checking algorithms. Instead, we restrict our attention to those predicates which can be encoded by $EF_{\mathcal{R}}$ formulae in the following sense: for each $f \in F$ there is an $EF_{\mathcal{R}}$ formula Ψ_f such that for each $g \in G$ we have that $g \models \Psi_f$ iff $N(g, f)$ is true. In this case we also define the formula $\Psi_\bot := \bigwedge_{f \in F} \neg\Psi_f$.

A concrete example of a predicate which can be encoded by $EF_{\mathcal{R}}$ formulae is, e.g., the 'I' predicate defined in the previous section: For every $f \in F$ let $A_f := \{a \in Act \mid \exists f'.\, f \xrightarrow{a} f'\}$. Then

$$\Psi_f := \bigwedge_{a \in A_f} \langle a \rangle true \ \wedge \bigwedge_{a \in Act - A_f} \neg\langle a \rangle true$$

The 'D' predicate can be encoded in a similar way.

Now we design the family of $\Phi_{j,p}$ formulae, where $0 \leq j \leq k$ and $p \in Tree_j$, in such a way that for every $g \in G$ the following equivalence holds:

$$g \overset{RN}{\simeq}_j p \iff g \models \Phi_{j,p}$$

Having these formulae, the Step and the Reach problems can be encoded in a rather straightforward way:

- (g, a, j, p) is a positive instance of the Step problem iff $g \models \Diamond_{C_a}(\Phi_{j,p})$
- (g, p) is a positive instance of the Reach problem iff $g \models \Diamond(\Phi_{k,p})$

The family of $\Phi_{j,p}$ formulae is defined inductively on j as follows:

- $\Phi_{0,p} := \Psi_f$, where $f = label(p)$
- $\Phi_{j+1,p} := \Psi_f \wedge \left(\bigwedge_{a \in Act} \bigwedge_{p' \in S(p,a)} \Diamond_{C_a} \Phi_{j,p'} \right) \wedge \left(\bigwedge_{a \in Act} (\neg\Diamond_{C_a}(\bigwedge_{p' \in S(p,a)} \neg\Phi_{j,p'})) \right)$,

  where $f = label(p)$ and $S(p, a) = \{p' \mid p \xrightarrow{a} p'\}$. Empty conjunctions are equivalent to *true*.

Thus, the characteristic formula Θ_f for a process f of a finite-state system $\mathcal{F} = (F, Act, \rightarrow)$ with k states is defined by

$$\Theta_f \;\equiv\; \Phi_{k,f} \wedge \neg\Diamond\left(\bigwedge_{f' \in F} \neg\Phi_{k,f'}\right)$$

The decidability of the model checking problem for the logic $EF_{\mathcal{C}}$ depends on properties of the family of constraints $\mathcal{C}$. It has been shown in [23] that the model checking problem for PA processes and the logic $EF_{\mathcal{C}}$ is decidable for the class of *decomposable constraints* (see also [19] where the same result was proved later using a completely different technique). This result has been generalized to PAD processes in [20,24]. These constraints are called decomposable, because they can be decomposed w.r.t. sequential and parallel composition. A formal definition is as follows: a set of decomposable constraints $\mathcal{DC}$ is a finite set of unary predicates on finite sequences of actions that contains the predicates *true* and *false* and satisfies the following conditions:

1. For every $C \in \mathcal{DC}$ there is a finite index set I and a finite set of decomposable constraints $\{C_i^1, C_i^2 \in \mathcal{DC} \mid i \in I\}$ s.t.

$$\forall w, w_1, w_2.\; w_1 w_2 = w \implies \left(C(w) \iff \bigvee_{i \in I} C_i^1(w_1) \wedge C_i^2(w_2)\right)$$

2. For every $C \in \mathcal{DC}$ there is a finite index set J and a finite set of decomposable constraints $\{C_i^1, C_i^2 \in \mathcal{DC} \mid i \in J\}$ s.t.

$$\forall w_1, w_2.\left(\;(\exists w \in \mathit{interleave}(w_1, w_2).\, C(w)) \iff \bigvee_{i \in J} (C_i^1(w_1) \wedge C_i^2(w_2))\right)$$

   where $\mathit{interleave}(w_1, w_2)$ is the set of all interleavings of w_1 and w_2 defined by

$$\begin{aligned}
\mathit{interleave}(\varepsilon, w) &:= \{w\} \\
\mathit{interleave}(w, \varepsilon) &:= \{w\} \\
\mathit{interleave}(a_1 w_1, a_2 w_2) &:= \{a_1 w \mid w \in \mathit{interleave}(w_1, a_2 w_2)\} \;\cup \\
&\qquad\quad \{a_2 w \mid w \in \mathit{interleave}(a_1 w_1, w_2)\}
\end{aligned}$$

It is easy to see that the closure of a set of decomposable constraints under disjunction is again a set of decomposable constraints (see [19,32] for more on decomposable constraints and decomposable languages). All the previously mentioned examples of relations R can be expressed by decomposable constraints. Consider the relation W for weak bisimulation. There we have the following constraints:

$$W_\tau(w) := (w = \tau^i \text{ for some } i \in \mathbb{N}_0)$$

$$W_a(w) := (w = \tau^i a \tau^j \text{ for some } i, j \in \mathbb{N}_0)$$

These constraints can be decomposed w.r.t. sequential and parallel composition. For W_τ this is trivial. For W_a we have

$$\begin{aligned} W_a(w_1 w_2) &\Longleftrightarrow (W_a(w_1) \wedge W_\tau(w_2)) \vee (W_\tau(w_1) \wedge W_a(w_2)) \\ (\exists w \in \mathit{interleave}(w_1, w_2).\, W_a(w)) &\Longleftrightarrow (W_a(w_1) \wedge W_\tau(w_2)) \vee (W_\tau(w_1) \wedge W_a(w_2)) \end{aligned}$$

Now we show decomposability for some other (nonstandard) relations that were defined in Section 3. For the relation K the decomposition is trivial. For the relation L we have the constraint

$$L_a(w) := w \text{ begins with } a$$

The decomposition is

$$\begin{aligned} L_a(w_1 w_2) &\iff L_a(w_1) \\ (\exists w \in \mathit{interleave}(w_1, w_2).\, L_a(w)) &\iff L_a(w_1) \vee L_a(w_2) \end{aligned}$$

For the relation M we have the constraints

$$\begin{aligned} &M_\tau(w) := \mathit{true} \\ &M_a(w) := w \text{ contains at least one } a \end{aligned}$$

The decomposition of M_τ is trivial. The decomposition of M_a is

$$\begin{aligned} M_a(w_1 w_2) &\iff M_a(w_1) \vee M_a(w_2) \\ (\exists w \in \mathit{interleave}(w_1, w_2).\, M_a(w)) &\iff M_a(w_1) \vee M_a(w_2) \end{aligned}$$

However, there are also relations R that are closed under substitution, but which yield non-decomposable constraints. For example, let $Act = \{a, b\}$ and $R(a) := \{w \mid \#_a w > \#_b w\}$ and $R(b) := \{b\}$, where $\#_a w$ is the number of actions a in w. The function R is obviously closed under substitution, but the corresponding set of constraints is not decomposable. On the other hand, there are decomposable constraints that are not closed under substitution like, e.g., $R(a) := \{a^i \mid 1 \leq i \leq 5\}$. Now we can formulate a general decidability theorem:

Theorem 19 *The problem $g \overset{RN}{\sim} f$, where R yields a set of constraints $\mathcal{R}$ contained in a set $\mathcal{DC}$ of decomposable constraints, N is expressible in $EF_{\mathcal{R}}$, g is a PAD processes, and f is a finite-state process, is decidable.*

Corollary 20 *Weak bisimilarity between PAD processes and finite-state ones is decidable.*

Remark 21 (Complexity of the problem)
The complexity of our algorithm for the problem $g \overset{RN}{\sim} f$ depends on the complexity of the model checking problem for $EF_{\mathcal{C}}$ and PAD, which is not known exactly yet. The algorithm for PAD in [20,24] and the different algorithms for PA in [23] and [19] all have non-elementary complexity. For BPP, model checking with $EF_{\mathcal{C}}$ is PSPACE-complete [22,24] (see also Section 6). The $EF_{\mathcal{R}}$ formulae that are constructed for a finite-state system $\mathcal{F}$ with k states have exponential size in k, but a nesting-depth of the operator $\Diamond$ that is only polynomial in k. Model checking can be done 'on-the-fly' while these formulae are constructed and thus polynomial space suffices. Hence, the problem $g \overset{RN}{\sim} f$ is in PSPACE for BPP.

For BPA and PDA, model checking with $EF_{\mathcal{C}}$ is known to be in EXPTIME [36,5]. It was claimed in [5] that it is even in PSPACE, but the given proof contains an error (it assumed that an accepting polynomial space-bounded Turing machine always has an accepting computation of polynomial length; however, there are cases where the shortest accepting computation has an exponential length). Thus the question about the complexity of model checking pushdown systems with $EF_{\mathcal{C}}$ is open again. Still we conjecture PSPACE-completeness to be most likely, because the number of alternations between conjunction and disjunction in the model checking problem is bounded by the size of the formula and thus polynomial. So far, our construction yields an EXPTIME algorithm for the problem $g \overset{RN}{\sim} f$ for BPA and PDA.

The known lower bounds for the model checking problem are PSPACE-hardness for BPP [10] and BPA [25] (and thus also for for PDA, PA and PAD). However, unlike the upper bounds, the lower bounds for the model checking problem do not carry over to the bisimulation problem $g \overset{RN}{\sim} f$. For example, it has recently been shown that weak bisimilarity between BPA and finite-state systems is decidable in polynomial time [18], while model checking BPA with $EF_{\mathcal{C}}$ is PSPACE-hard [25].

Decidability of the model checking problem for the $EF_{\mathcal{R}}$ logic in a certain class of transition systems $\mathcal{K}$ is a sufficient but not necessary condition for decidability of R-N-bisimilarity between processes of $\mathcal{K}$ and finite-state ones. For example, model checking the 'pure' *EF* (without any constraints) is undecidable for Petri nets, but the Step and the Reach problems are decidable for S-T-bisimilarity [15]. In fact, strong bisimilarity is the simplest form of R-N-bisimilarity and the *EF* formulae which encode the two problems are therefore very simple as well. An exact formulation of this observation is given in the following theorem:

Theorem 22 *An EF formula is* simple *iff it is of the form $\Diamond\Phi$ where the subformula Φ does not contain any $\Diamond$-operator (i.e., Φ is a formula of Hennessy-Milner logic [26]). If the model checking problem for simple EF formulae is*

decidable in a class $\mathcal{K}$ of transition systems, then strong bisimilarity is decidable between processes of $\mathcal{K}$ and finite-state ones.

PROOF. The $\overset{ST}{\simeq}_j$ equivalence with a given Tree process p can be encoded by a formula of Hennessy-Milner logic for every $j \in \mathbb{N}_0$. Consequently, the Step problem can also be encoded by a formula of Hennessy-Milner logic, and the Reach problem is encoded by a formula of the form $\Diamond\Phi$ where Φ is a formula of Hennessy-Milner logic. □

The model checking problem for simple *EF* formulae is essentially a kind of generalized reachability problem (one checks whether there is a reachable state that satisfies a given formula of Hennessy-Milner logic). Of course, it is much easier than the general model checking problem for *EF*. Thus, decidability issues can be different—we have already mentioned that model checking *EF* logic is undecidable for Petri nets; however, model checking simple *EF* is decidable (due to the decidability of the Reach problem—see below). For example, in the case of Petri nets we can observe that the markings which satisfy some formula of H.M. logic can be characterized by boolean combinations of constraints of the form $p \geq k$ or $p \leq k$, meaning that there are at least/at most k tokens in place p. This leads to a generalized reachability problem which is decidable [11].

Now we show that the model checking problem for simple *EF* formulae can be seen as a reformulation of the Step and the Reach problems in the case of strong bisimilarity (the Step problem is trivially decidable, and the Reach problem is 'equivalently hard' to the model checking problem for the simple *EF* logic). This shows the essence of the whole problem in a new light.

Theorem 23 *The model checking problem for simple EF formulae and the special variant of the Reach problem for strong bisimilarity are inter-reducible in the Turing sense (i.e., decidability of one of the two problems implies decidability of the other one).*

PROOF. Decidability of the model checking problem for simple *EF* formulae implies decidability of the Reach problem, as shown in Theorem 22. We prove the other direction; let $\Diamond\Phi$ be a simple *EF* formula. First, let us realize that the sub-formula Φ cannot distinguish between states related by $\overset{ST}{\simeq}_n$, where $n = \mathit{length}(\Phi)$. Due to Lemma 15 we know that for every state g of the transition system $\mathcal{G}$ there is a $p \in \mathit{Tree}_n$ such that $g \overset{ST}{\simeq}_n p$ (as the predicate T is trivial, we do not have to label the nodes of Trees; hence the construction of Tree_n does not depend on the transition system $\mathcal{F}$—see Definition 14). For

each $p \in Tree_n$ we check whether $p \models \Phi$. Now it is easy to see that $g' \models \Diamond\Phi$ iff $Reach(g', p) = true$ for some $p \in Tree_n$ such that $p \models \Phi$. □

5 Undecidability Results

In this section we provide several negative (undecidability) results which help to clarify the decidability/undecidability border in the area of comparing infinite-state processes with finite-state ones.

Intuitively, any 'nontrivial' equivalence with finite-state processes should be undecidable for a class of processes having 'full Turing power', which can be formally expressed as, e.g., the ability to simulate Minsky counter machines.

Definition 24 A counter machine $\mathcal{M}$ *with nonnegative counters* $c_1, c_2, ..., c_m$ *is a sequence of instructions*

$$\begin{array}{ll} 1: & INS_1 \\ 2: & INS_2 \\ \vdots & \\ n-1: & INS_{n-1} \\ n: & \texttt{halt} \end{array}$$

where each INS_i *(*$i = 1, 2, ..., n-1$*) is in one of the following two forms (assuming* $1 \leq k, k_1, k_2 \leq n$, $1 \leq j \leq m$*)*

- $c_j := c_j + 1$; $\texttt{goto}\ k$
- $\texttt{if}\ c_j = 0\ \texttt{then goto}\ k_1\ \texttt{else}\ (c_j := c_j - 1;\ \texttt{goto}\ k_2)$

The halting problem is undecidable even for Minsky machines with two counters initialized to zero values [27]. Any such machine $\mathcal{M}$ can be easily 'mimicked' by a StExt(PA) process $P(\mathcal{M}) = (\Delta, Q, BT)$ where

- Δ contains the following rules:
  · $Z_j \stackrel{a}{\rightarrow} I_j.Z_j$, $Z_j \stackrel{a}{\rightarrow} Z_j$
  · $I_j \stackrel{a}{\rightarrow} I_j.I_j$, $I_j \stackrel{a}{\rightarrow} \varepsilon$
  where $j \in \{1, 2\}$.
- $Q = \{q_1, \ldots, q_n\}$, where n is the number of instructions of $\mathcal{M}$.
- BT is determined by the following rules:

(1) If the program of $\mathcal{M}$ contains an instruction of the form

$$l:\ c_j := c_j + 1;\quad \texttt{goto}\ k$$

then BT contains the elements $(Z_j \stackrel{a}{\rightarrow} I_j.Z_j, q_l, q_k)$ and $(I_j \stackrel{a}{\rightarrow} I_j.I_j, q_l, q_k)$.

(2) If the program of $\mathcal{M}$ contains an instruction of the form

$$l : \texttt{if } c_j = 0 \texttt{ then goto } k_1 \texttt{ else } (c_j := c_j - 1; \texttt{ goto } k_2)$$

then BT contains the elements $(Z_j \stackrel{a}{\rightarrow} Z_j, q_l, q_{k_1})$ and $(I_j \stackrel{a}{\rightarrow} \varepsilon, q_l, q_{k_2})$.
(3) Each element of BT can be derived using the rule 1 or 2.

Intuitively, the (two) counters of the machine $\mathcal{M}$ are modeled by a simple PA process $(I_1.I_1 \ldots I_1.Z_1)\|(I_2.I_2 \ldots I_2.Z_2)$ where the number of I_j's means the current value of the counter c_j, $j \in \{1, 2\}$ (the starting zero point being modeled by $Z_1\|Z_2$). The control states $q_1, \ldots, q_n$ correspond to the instructions of $\mathcal{M}$ (more precisely, to their labels). Each state determines the unique transition to be performed next with the exception of q_n which is the 'halting state'. The process $(q_1, Z_1\|Z_2)$ is able either to perform the action a boundedly many times and to stop (its behavior can be defined as a^m for some $m \in \mathbb{N}_0$) or to do a forever (its behavior being a^ω); this depends on whether the machine $\mathcal{M}$ halts or not. Notice that a^ω is the behavior of the one-state process f where $\{f \stackrel{a}{\rightarrow} f\}$ is the underlying PRS. When we declare as *reasonable* any equivalence which distinguishes between (processes with) behaviors a^ω and a^m, we can conclude:

Theorem 25 *Any reasonable equivalence between StExt(PA) processes and finite-state ones is undecidable.*

It is obvious that (almost) any R-N-bisimilarity is reasonable in the above sense, except for some trivial cases. For weak bisimilarity, we can even show that none of the problems $g \stackrel{WT}{\simeq}_1 f$, $g \stackrel{WT}{\not\simeq}_1 f$ is semidecidable when g is a StExt(PA) process. It suffices to realize that we can label all transitions in $P(\mathcal{M})$ by τ and add a special a-transition enabled in the (halting) state q_n. Now $q_1(Z_1\|Z_2) \stackrel{WT}{\simeq}_1 \tau^\omega$ iff the machine $\mathcal{M}$ does not halt, and similarly $q_1(Z_1\|Z_2) \stackrel{WT}{\simeq}_1 f$ where $\{f \stackrel{\tau}{\rightarrow} f, f \stackrel{a}{\rightarrow} g\}$ iff the machine $\mathcal{M}$ halts.

Now, the claim of Remark 10 is also easy to see; if we take the modified process $P(\mathcal{M})$ of the previous paragraph, we can observe that $q_1(Z_1\|Z_2) \stackrel{WT}{\sim}_j \tau^\omega$ for every j which is less than the number of computational steps of $\mathcal{M}$. On the other hand, if $\mathcal{M}$ halts then $q_1(Z_1\|Z_2) \stackrel{WT}{\not\simeq}_1 \tau^\omega$. Therefore, the implication $q_1(Z_1\|Z_2) \stackrel{WT}{\sim}_j \tau^\omega \implies q_1(Z_1\|Z_2) \stackrel{WT}{\simeq}_1 \tau^\omega$ is invalid for any $j \in \mathbb{N}$, because for each such j there is a machine with more then j computational steps which halts.

Once seeing that StExt(PA) are strong enough to make our equivalences undecidable, it is natural to ask what happens when we add finite-state control parts to processes from subclasses of PA, namely to BPA and BPP.

We have already shown that every R-N-bisimilarity such that R yields decomposable constraints and N is expressible within $EF_{\mathcal{R}}$ is decidable be-

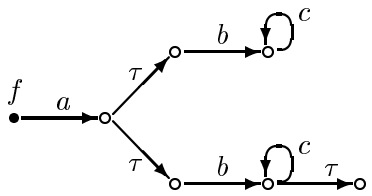


Fig. 3. A finite-state system used in the proof of Theorem 26

tween StExt(BPA) (i.e., PDA) processes and finite-state ones. In the case of StExt(BPP), strong bisimilarity with finite-state processes is decidable [15]. Here we demonstrate that the problem for weak bisimilarity is undecidable. Our proof is obtained by modifying the construction which has been used in [13] to show the undecidability of weak bisimilarity between Petri nets and finite-state systems. To make this paper self-contained, we now give a concise description of this modified construction.

Theorem 26 *Weak bisimilarity is undecidable between StExt(BPP) processes and finite-state ones.*

PROOF. Consider a Minsky machine $\mathcal{M}$ as in Definition 24 with just two counters ($m = 2$). In a stepwise manner, we show how to construct a StExt(BPP) process $P(\mathcal{M})$ such that $P(\mathcal{M})$ is weakly bisimilar to the process f of Figure 3 iff $\mathcal{M}$ does not halt.

We begin with using just the action τ, the process constants Z, I_1, I_2, D, S and the control states $p_1, \ldots, p_n, q_1, \ldots, q_n$. The states (p_1, Z), (q_1, Z) are considered as two possible starting ones. Basic transitions of $P(\mathcal{M})$ are determined as follows: for every machine instruction

$$l: \ c_j := c_j + 1; \quad \texttt{goto} \ k$$

we have $(Z \xrightarrow{\tau} I_j \| S \| Z, p_l, p_k)$ and $(Z \xrightarrow{\tau} I_j \| S \| Z, q_l, q_k)$. For every machine instruction

$$l: \ \texttt{if} \ c_j = 0 \ \texttt{then goto} \ k_1 \ \texttt{else} \ (c_j := c_j - 1; \ \texttt{goto} \ k_2)$$

we have

$$(I_j \xrightarrow{\tau} D \| S, p_l, p_{k_2}), (Z \xrightarrow{\tau} S \| Z, p_l, p_{k_1}), (I_j \xrightarrow{\tau} I_j \| S, p_l, q_{k_1}),$$

and

$$(I_j \xrightarrow{\tau} D \| S, q_l, q_{k_2}), (Z \xrightarrow{\tau} S \| Z, q_l, q_{k_1}).$$

We observe that, for every reachable state (r, E), exactly one occurrence of Z appears in the expression E; the constant Z only serves as an auxiliary symbol which is used to model the 'empty left-hand side' of rules. The number of I_1 (I_2) in E is meant to correspond to the current value of counter c_1 (c_2). By (the occurrences of) S we count the number of steps, and by D the number of 'decreasing steps'.

Thus both (p_1, Z) and (q_1, Z) can simulate the computation of $\mathcal{M}$ (with counters initialized to zero). Nevertheless, also 'cheating steps' (performing a 'zero step' instead of a decreasing one) are possible; it reflects the inability of StExt(BPP) (more generally, Petri nets) to test for zero. Note that by (and only by) a cheating step we can go from the 'p-domain' to the 'q-domain'.

Now we shall refine the transitions mentioned so far. The idea is to view the sequence of steps as a string of 0's (non-decreasing steps) and 1's (decreasing steps), and to enable D to 'count' the respective binary number. We introduce an additional auxiliary constant C, and replace every transition $(E_1 \stackrel{\tau}{\to} E_2, r_1, r_2)$ by the set

$$(E_1 \stackrel{\tau}{\to} E_2, r_1, r'),\ (D \stackrel{\tau}{\to} C \| C, r', r'),\ (Z \stackrel{\tau}{\to} Z, r', r''),$$
$$(C \stackrel{\tau}{\to} D, r'', r''),\ (Z \stackrel{\tau}{\to} Z, r'', r_2)$$

where r', r'' are newly added control states. It allows (though does not force) to double the number of D's in each step (after adding 1 in the case of a decreasing step). Now we add a control state h and the basic transition $(Z \stackrel{\tau}{\to} D \| Z, q_n, h)$.

Defining $vec(E)$ as the 5-dimensional vector giving the numbers of (occurrences of) I_1, I_2, C, D, S in E, we can easily derive (similarly as in [12]) that the set $\{\ vec(E) \mid \exists r$ such that (r, E) is reachable from $(p_1, Z)\ \}$ is a subset of $\{\ vec(E) \mid \exists r$ such that (r, E) is reachable from $(q_1, Z)\ \}$; moreover, the two sets are equal iff $\mathcal{M}$ does not halt.

To proceed with the construction of our desired $P(\mathcal{M})$, we now take a disjoint union of the so far constructed StExt(BPP) system with its isomorphic duplicate. For a control state r (or a process constant X), we denote the respective duplicate by $\overline{r}$ (or $\overline{X}$).

We now introduce new control states s_1, s_2, s_3; moreover, the pairs (s, r), $(s, \overline{r})$—where $s \in \{s_1, s_2, s_3\}$ and r is 'old'—will also serve as control states. The process $P(\mathcal{M})$ is defined as $((s_1, q_1), Z \| \overline{Z})$ when we also include the following basic transitions (adding actions a, b, c): for every $(E \stackrel{\tau}{\to} E', r, r')$ we add

$$(E \stackrel{\tau}{\to} E', (s_1, r), (s_1, r')), (E \stackrel{\tau}{\to} E', (s_2, \overline{r}), (s_2, \overline{r'})).$$

For every r, we add

$$(Z \xrightarrow{\tau} Z, (s_1, r), s_1), (Z \xrightarrow{\tau} Z, (s_2, \overline{r}), s_2).$$

We also add $(Z \xrightarrow{a} Z, s_1, (s_2, \overline{p_1}))$ and $(Z \xrightarrow{b} Z, s_2, s_3)$. Finally, for every $X \in \{I_1, I_2, C, D, S\}$ we add a new control state s_X and

$$(X \xrightarrow{c} X, s_3, s_3),\ (\overline{X} \xrightarrow{c} \overline{X}, s_3, s_3),\ (X \xrightarrow{\tau} \varepsilon, s_3, s_X),$$
$$(\overline{X} \xrightarrow{\tau} \varepsilon, s_X, s_3),\ (Z \xrightarrow{c} Z, s_X, s_X)$$

Checking that $P(\mathcal{M})$ is weakly bisimilar to f iff $\mathcal{M}$ does not halt can be done analogously to [13]. □

6 Conclusions, Future Work

We designed a general method for proving decidability of R-N-bisimilarity between infinite-state processes and finite-state ones (Theorem 12) by reducing this problem to two other problems—the Step and the Reach problem (Theorem 17 and 18). We also showed how to encode these special problems by formulae of $EF_{\mathcal{R}}$ logic. In this way we constructed characteristic formulae for finite-state systems up to bisimulation in the logic $EF_{\mathcal{C}}$. As this logic is decidable for PAD (and hence also PA and PDA) processes, we obtained a general decidability theorem (Theorem 19), which says that every R-N-bisimilarity such that R yields decomposable constrains on sequences of actions and N can be expressed by $EF_{\mathcal{R}}$ formulae is decidable between PAD and finite-state processes. This class of R-N-bisimilarities includes all versions of R-N-bisimulation equivalences mentioned in this paper. Examples are the relations $\overset{KI}{\sim}$, $\overset{LT}{\sim}$, $\overset{MI}{\sim}$, or $\overset{WD}{\sim}$, but most importantly $\overset{ST}{\sim}$ and $\overset{WT}{\sim}$ (i.e., strong and weak bisimilarity).

Then we demonstrated that each 'reasonable' R-N-bisimilarity is undecidable between StExt(PA) processes and finite-state ones (Theorem 25); this is caused by the fact that StExt(PA) processes have full Turing power. Moreover, even if we restrict our attention to StExt(BPP), we get an undecidability result for weak bisimilarity (Theorem 26). This proof is obtained by a modification of the one which has been used for Petri nets.

A complete summary of the results on decidability of bisimulation-like equivalences with finite-state processes is given in the table below. As we want to clarify what results have been previously obtained by other researchers, our table contains more rows than it is necessary (e.g., the positive result for

PAD and $\overset{RN}{\sim}$, where R and N have the above indicated properties, 'covers' all positive results for BPA, BPP, PA, and PDA).

We also add a special column which indicates decidability of the model-checking problem for the logic *EF*. The decidability of *EF* for pushdown processes (PDA) and BPA follows from a much stronger result by Muller and Schupp [29] who showed the decidability of monadic second order logic for pushdown automata. Later, model checking PDA with *EF* was shown to be in *EXPTIME*[36,5] (see also Remark 21). Model checking BPP with *EF* was shown to be decidable by Esparza [10] and *PSPACE*-complete by Mayr [22,24]. Decidability of *EF* for PA was shown by Mayr [23] and later by Lugiez and Schnoebelen [19], who used a completely different method. The decidability for PAD was shown in [20,24]. The undecidability of *EF* for Petri nets was shown by Esparza in [10]. The undecidability of *EF* for StExt(BPP) and StExt(PA) follows directly from the undecidability results on bisimilarity in this paper.

	$\overset{ST}{\sim}$	$\overset{WT}{\sim}$	$\overset{RN}{\sim}$	*EF*
BPA	Yes [9]	**YES**	**YES**	Yes [29,5]
BPP	Yes [8]	Yes [22]	**YES**	Yes [10,22,24]
PA	Yes [14]	**YES**	**YES**	Yes [23,19]
StExt(BPA), i.e., PDA	Yes [14]	**YES**	**YES**	Yes [29,5]
StExt(BPP), i.e., PPDA	Yes [15]	**NO**	**NO**	**NO**
StExt(PA)	No [14]	No [14]	No [14]	**NO**
PAD	**YES**	**YES**	**YES**	Yes [20,24]
Petri nets	Yes [15]	No [13]	No [13]	No [10]

The results obtained in this paper are in boldface. Note that although model-checking *EF* logic is undecidable for StExt(BPP) processes and Petri nets, strong bisimilarity with finite-state systems is decidable. The original proof in [15] in fact demonstrates decidability of the Reach problem (the Step problem is trivially decidable), hence our general strategy applies also in this case.

A unifying concept similar to R-N-bisimulation can also be used for simulation-like equivalences—we can define the R-N-simulation relation in the very same way as R-N-bisimulation (which can be then seen as a special case of R-N-simulation with the property that its inverse is also an R-N-simulation). The predicate N becomes more important in this context, as it allows to define some of the known and studied simulation-like equivalences (e.g., the ready simulation equivalence). An interesting open problem is whether it is possible to design a general strategy for deciding R-N-simulation equivalence between infinite-state and finite-state processes in a similar way as for R-N-bisimilarity

(recently, the decidability/tractability border for strong simulation (i.e., S-T-simulation) with finite-state systems has been established in [17]). Another set of open problems is the decidability of branching bisimilarity with finite-state processes.

References

[1] *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*. Springer, 1996.

[2] *Proceedings of CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*. Springer, 1997.

[3] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the Association for Computing Machinery*, 40:653–682, 1993.

[4] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[5] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *Proceedings of CONCUR'97* [2], pages 135–150.

[6] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[7] I. Černá, M. Křetínský, and A. Kučera. Comparing expressibility of normed BPA and normed BPP processes. *Acta Informatica*, 36(3):233–256, 1999.

[8] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 1993.

[9] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.

[10] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[11] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74:71–93, 1990.

[12] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[13] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimilarity. In *Proceedings of ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 1996.

[14] P. Jančar and A. Kučera. Bisimilarity of processes with finite-state systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.

[15] P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 1995.

[16] A. Kučera. On effective decomposability of sequential behaviours. *Theoretical Computer Science*. To appear.

[17] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In *Proceedings of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 503–512. Springer, 1999.

[18] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proceedings of CONCUR'99*, Lecture Notes in Computer Science. Springer, 1999. To appear.

[19] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proceedings of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 1998.

[20] R. Mayr. Decidability of Model Checking with the Temporal Logic EF. *Theoretical Computer Science*. To appear.

[21] R. Mayr. Process rewrite systems. *Information and Computation*. To appear.

[22] R. Mayr. Weak bisimulation and model checking for basic parallel processes. In *Proceedings of FST&TCS'96*, volume 1180 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 1996.

[23] R. Mayr. Model checking PA-processes. In *Proceedings of CONCUR'97* [2], pages 332–346.

[24] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.

[25] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

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

[27] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[28] F. Moller. Infinite results. In *Proceedings of CONCUR'96* [1], pages 195–216.

[29] D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second order logic. *Theoretical Computer Science*, 37(1):51–75, 1985.

[30] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings 5^{th} GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[31] W. Reisig. *Petri Nets—An Introduction*. Springer, 1985.

[32] Ph. Schnoebelen. Decomposable regular languages and the shuffle operator. *EATCS Bulletin*, 67:283–289, February 1999.

[33] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[34] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, 195:113–131, 1998.

[35] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the Association for Computing Machinery*, 43(3):555–600, 1996.

[36] I. Walukiewicz. Pushdown processes: games and model checking. In *International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*. Springer Verlag, 1996.

Weak Bisimilarity between Finite-State Systems and BPA or Normed BPP is Decidable in Polynomial Time

Antonín Kučera [a,1], Richard Mayr [b,2]

[a] *Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic. E-mail:* tony@fi.muni.cz

[b] *LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu, F-75251 Paris Cedex 05. France. E-mail:* mayr@liafa.jussieu.fr

Abstract

We prove that weak bisimilarity is decidable in polynomial time between finite-state systems and several classes of infinite-state systems: context-free processes (BPA) and normed Basic Parallel Processes (normed BPP). To the best of our knowledge, these are the first polynomial algorithms for weak bisimilarity problems involving infinite-state systems.

Key words: concurrency, infinite-state systems, process algebras, verification, bisimulation

1 Introduction

Recently, a lot of attention has been devoted to the study of decidability and complexity of verification problems for infinite-state systems [33,12,5]. We consider the problem of weak bisimilarity between certain infinite-state processes and finite-state ones. The motivation is that the intended behavior of a process is often easy to specify (by a finite-state system), but a 'real' implementation can contain components which are essentially infinite-state (e.g., counters, buffers, recursion, creation of new parallel subprocesses). The aim is

[1] On leave at the Institute for Informatics, Technical University Munich. Supported by the Alexander von Humboldt Foundation and by the Grant Agency of the Czech Republic, grant No. 201/00/0400.

[2] Supported by DAAD Post-Doc grant D/98/28804.

to check if the finite-state specification and the infinite-state implementation are semantically equivalent, i.e., weakly bisimilar.

We concentrate on the classes of infinite-state processes definable by the syntax of BPA (Basic Process Algebra) and normed BPP (Basic Parallel Processes) systems. BPA processes (also known as context-free processes) can be seen as simple sequential programs (due to the binary operator of sequential composition). They have recently been used to solve problems of data-flow analysis in optimizing compilers [13]. BPP [8] model simple parallel systems (due to the binary operator of parallel composition). They are equivalent to communication-free nets, the subclass of Petri nets [36] where every transition has exactly one input-place [11]. A process is *normed* iff at every reachable state it can terminate via a finite sequence of computational steps.

Although the syntax of BPA and BPP allows to define simple infinite-state systems, from the practical point of view it is also important that they can give very compact definitions of finite-state processes (i.e., the size of a BPA/BPP definition of a finite-state process F can be exponentially smaller than the number of states of F—see the next section). As our verification algorithms are polynomial in the size of the BPA/BPP definition, we can (potentially) verify *very* large processes. Thus, our results can be also seen as a way how to overcome the well-known problem of state-space explosion.

The state of the art. Baeten, Bergstra, and Klop [1] proved that *strong bisimilarity* [35] is decidable for normed BPA processes. Simpler proofs have been given later in [20,14], and there is even a polynomial-time algorithm [17]. The decidability result has later been extended to the class of all (not necessarily normed) BPA processes in [10], but the best known algorithm is doubly exponential [4]. Decidability of strong bisimilarity for BPP processes has been established in [9], but the associated complexity analysis does not yield an elementary upper bound (although some deeper examination might in principle show that the algorithm *is* elementary). Strong bisimilarity of BPP has been shown to be co-$\mathcal{NP}$-hard in [28]. However, there is a polynomial-time algorithm for the subclass of normed BPP [18]. Strong bisimilarity between normed BPA and normed BPP is also decidable [7]. This result even holds for parallel compositions of normed BPA and normed BPP processes [22]. Recently, this has even been generalized to the class of all normed PA-processes [16].

For weak bisimilarity, much less is known. Semidecidability of weak bisimilarity for BPP has been shown in [11]. In [15] it is shown that weak bisimilarity is decidable for those BPA and BPP processes which are 'totally normed' (a process is totally normed if it can terminate at any moment via a finite sequence of computational steps, but at least one of those steps must be 'visible', i.e., non-internal). Decidability of weak bisimilarity for general BPA and BPP is

open; those problems might be decidable, but they are surely intractable (assuming $\mathcal{P} \neq \mathcal{NP}$). Weak bisimilarity of (normed) BPA is *PSPACE*-hard [38]. An $\mathcal{NP}$ lower bound for weak bisimilarity of BPP has been shown by Stříbrná in [38]. This result has been improved to Π_2^p-hardness by Mayr [28] and very recently to *PSPACE*-hardness by Srba in [37]. Moreover, the *PSPACE* lower bound for weak bisimilarity of BPP in [37] holds even for normed BPP.

The situation is dramatically different if we consider weak bisimilarity between certain infinite-state processes and finite-state ones. This study is motivated by the fact that the intended behavior of a process is often easy to specify (by a finite-state system), but a 'real' implementation can contain components which are infinite-state (e.g., counters, buffers, recursion, creation of new parallel subprocesses). It has been shown in [26] that weak bisimilarity between BPP and finite-state processes is decidable. A more general result has recently been obtained in [21], where it is shown that many bisimulation-like equivalences (including the strong and weak ones) are decidable between PAD and finite-state processes. The class PAD [31,30] strictly subsumes not only BPA and BPP, but also PA [2] and pushdown processes. The result in [21] is obtained by a general reduction to the model-checking problem for the simple branching-time temporal logic *EF*, which is decidable for PAD [30]. As the model-checking problem for *EF* is hard (for example, it is known to be *PSPACE*-complete for BPP [26] and *PSPACE*-complete for BPA [39,27]), this does not yield an efficient algorithm.

Our contribution. We show that weak (and hence also strong) bisimilarity is decidable in polynomial time between BPA and finite-state processes, and between normed BPP and finite-state processes. To the best of our knowledge, these are the first polynomial algorithms for weak bisimilarity with infinite-state systems. Moreover, the algorithm for BPA is the first example of an efficient decision procedure for a class of *unnormed* infinite-state systems (the polynomial algorithms for strong bisimilarity of [17,18] only work for the normed subclasses of BPA and BPP, respectively). Due to the aforementioned hardness results for the 'symmetric case' (when we compare two BPA or two (normed) BPP processes) we know that our results cannot be extended in this direction. A recent work [29] shows that strong bisimilarity between pushdown processes (a proper superclass of BPA) and finite-state ones is already *PSPACE*-hard. Furthermore, weak bisimilarity remains computationally intractable (*DP*-hard) even between processes of one-counter nets and finite-state processes [23] (one-counter nets are computationally equivalent to the subclass of Petri nets with at most one unbounded place and can be thus also seen as very simple pushdown automata). Hence, our result for BPA is rather tight. The question whether the result for normed BPP can be extended to the class of all (not necessarily normed) BPP processes is left open. It should also be noted that *simulation equivalence* with a finite-state process is co-$\mathcal{NP}$-hard for BPA/BPP processes [25], *EXPTIME*-complete for

pushdown processes [24], but polynomial for one-counter nets [24].

The basic scheme of our constructions for BPA and normed BPP processes is the same. The main idea is that weak bisimilarity between BPA (or normed BPP) processes and finite-state ones can be generated from a finite base of 'small' size and that certain infinite subsets of BPA and BPP state-space can be 'symbolically' described by finite automata and context-free grammars, respectively. A more detailed intuition is given in Section 3. An interesting point about this construction is that it works although weak bisimulation is not a congruence w.r.t. sequential composition, but only a *left congruence*. In Section 4, we propose a natural refinement of weak bisimilarity called termination-sensitive bisimilarity which *is* a congruence and which is also decidable between BPA and finite-state processes in polynomial time. The result demonstrates that the technique which has been used for weak bisimilarity actually has a wider applicability—it can be adapted to many 'bisimulation-like' equivalences. Finally, we should note that our aim is just to show that the mentioned problems are in $\mathcal{P}$; although we do compute the degrees of bounding polynomials explicitly, our analysis is quite simple and rough. Moreover, both presented algorithms could be easily improved by employing standard techniques. See the final section for further comments.

2 Definitions

We use process rewrite systems [31] as a formal model for processes. Let $Act = \{a, b, c, \ldots\}$ and $Const = \{X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *process expressions* $\mathcal{E}$ is defined by

$$E \ ::= \ \varepsilon \ \mid \ X \ \mid \ E \| E \ \mid \ E.E$$

where $X \in Const$ and ε is a special constant that denotes the empty expression. Intuitively, '.' is sequential composition and '$\|$' is parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '$\|$' are associative, '$\|$' is commutative, and 'ε' is a unit for '.' and '$\|$'.

A *process rewrite system* [31] is specified by a finite set of *rules* Δ which have the form $E \stackrel{a}{\rightarrow} F$, where $E, F \in \mathcal{E}$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$, $Act(\Delta)$ is the set of labels, and transitions are determined by Δ

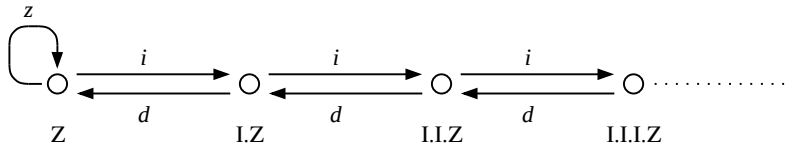
and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\to} F) \in \Delta}{E \stackrel{a}{\to} F} \qquad \frac{E \stackrel{a}{\to} E'}{E.F \stackrel{a}{\to} E'.F} \qquad \frac{E \stackrel{a}{\to} E'}{E\|F \stackrel{a}{\to} E'\|F}$$

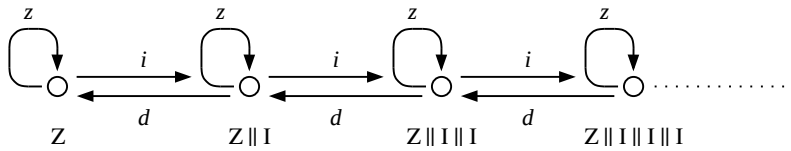
We extend the notation $E \stackrel{a}{\to} F$ to elements of Act^* in the standard way. F is *reachable* from E if $E \stackrel{w}{\to} F$ for some $w \in Act^*$.

Sequential and *parallel* expressions are those process expressions which do not contain the '$\|$' and the '.' operator, respectively. Finite-state, BPA, and BPP systems are subclasses of process rewrite systems obtained by putting certain restrictions on the form of the rules. Finite-state, BPA, and BPP allow only a single constant on the left-hand side of rules, and a single constant, sequential expression, and parallel expression on the right-hand side, respectively. The set of states of a transition system which is generated by a finite-state, BPA, or BPP process Δ is restricted to $Const(\Delta)$, the set of all sequential expressions over $Const(\Delta)$, or the set of all parallel expressions over $Const(\Delta)$, respectively.

Example 1 *Let $\Delta = \{Z \stackrel{z}{\to} Z, Z \stackrel{i}{\to} I.Z, I \stackrel{i}{\to} I.I, I \stackrel{d}{\to} \varepsilon\}$ be a process rewrite system. We see that Δ is a BPA system; a part of the transition system associated to Δ which is reachable from Z looks as follows:*



If we replace each occurrence of the '.' operator with the '$\|$' operator, we obtain a BPP system which generates the following transition system (again, we only draw the part reachable from Z):



A process is *normed* iff at every reachable state it can (successfully) terminate via a finite sequence of computational steps. For a BPA or BPP process, this is equivalent to the condition that for each constant $X \in Const(\Delta)$ of its underlying system Δ there is some $w \in Act^*$ such that $X \stackrel{w}{\to} \varepsilon$. We call such constants X with this property *normed*.

The semantical equivalence we are interested in here is *weak bisimilarity* [32]. This relation distinguishes between 'observable' and 'internal' moves (compu-

tational steps); the internal moves are modeled by a special action which is denoted 'τ' by convention. In what follows we consider process expressions over $Const(\Delta)$ where Δ is some fixed process rewrite system.

Definition 2 *The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by* $E \stackrel{a}{\Rightarrow} F$ *iff either* $E = F$ *and* $a = \tau$, *or* $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ *for some* $i, j \in \mathbb{N}_0$, $E', E'' \in \mathcal{E}$.

A binary relation R *over process expressions is a* weak bisimulation *iff whenever* $(E, F) \in R$ *then for each* $a \in Act$:

- *if* $E \stackrel{a}{\rightarrow} E'$ *then there is* $F \stackrel{a}{\Rightarrow} F'$ *such that* $(E', F') \in R$, *and*
- *if* $F \stackrel{a}{\rightarrow} F'$ *then there is* $E \stackrel{a}{\Rightarrow} E'$ *such that* $(E', F') \in R$.

Processes E, F *are* weakly bisimilar, *written* $E \approx F$, *iff there is a weak bisimulation relating them.*

Weak bisimilarity can be approximated by the family of $\approx_i$ relations, which are defined as follows:

- $E \approx_0 F$ for every E, F
- $E \approx_{i+1} F$ iff $E \approx_i F$ and the following conditions hold:
  - if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ such that $E' \approx_i F'$
  - if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ such that $E' \approx_i F'$

It is worth noting that $\approx_i$ is not an equivalence for $i \geq 1$, as it is not transitive. It is possible to approximate weak bisimilarity in a different way so that the approximations *are* equivalences (see [21]). However, we do not need this for our purposes.

Let Γ be a finite-state system with n states, $f, g \in Const(\Gamma)$. It is easy to show that the problem whether $f \approx g$ is decidable in $\mathcal{O}(n^3)$ time. First we compute in $\mathcal{O}(n^3)$ time the transitive closure of the transition system w.r.t. the $\stackrel{\tau}{\rightarrow}$ transitions and thus obtain a new system in which $\stackrel{a}{\rightarrow}$ is the same as $\stackrel{a}{\Rightarrow}$ in the old system. Then it suffices to decide strong bisimilarity of f and g in the new system. This can be done in $\mathcal{O}(n^2 \log n)$ time, using partition refinement techniques from [34].

Sometimes we also consider weak bisimilarity between processes of *different* process rewrite systems, say Δ and Γ. Formally, Δ and Γ can be considered as a *single* system by taking their disjoint union.

3 BPA Processes

In this section we prove that weak bisimilarity is decidable between BPA and finite-state processes in polynomial time.

Let E be a BPA process with the underlying system Δ, F a finite-state process with the underlying system Γ such that $Const(\Delta) \cap Const(\Gamma) = \emptyset$. We assume (w.l.o.g.) that $E \in Const(\Delta)$. Moreover, we also assume that for all $f, g \in Const(\Gamma)$, $a \in Act$ such that $f \neq g$ or $a \neq \tau$ we have that $f \stackrel{a}{\Rightarrow} g$ implies $f \stackrel{a}{\rightarrow} g \in \Gamma$. If those '$\stackrel{a}{\rightarrow}$' transitions are missing in Γ, we can add them safely. Adding these transitions does not change the weak bisimilarity relation among the states. In order to do this it suffices to compute (in cubic time) the transitive closure of Γ w.r.t. the τ transitions. These extra transitions do not influence our complexity estimations, as we always consider the worst case when Γ has all possible transitions. The condition that $a \neq \tau$ is there because we do not want to add new transitions of the form $f \stackrel{\tau}{\rightarrow} f$, because then our proof for weak bisimilarity would not immediately work for termination-sensitive bisimilarity (which is defined at the end of this section).

We use upper-case letters $X, Y, \ldots$ to denote elements of $Const(\Delta)$, and lower-case letters $f, g, \ldots$ to denote elements of $Const(\Gamma)$. Greek letters $\alpha, \beta, \ldots$ are used to denote elements of $Const(\Delta)^*$. The size of Δ is denoted by n, and the size of Γ by m (we measure the complexity of our algorithm in (n, m)).

The set $Const(\Delta)$ can be divided into two disjoint subsets of *normed* and *unnormed* constants (remember that $X \in Const(\Delta)$ is normed iff $X \stackrel{w}{\rightarrow} \varepsilon$ for some $w \in Act^*$). Note that it is decidable in $\mathcal{O}(n^2)$ time if a constant is normed. The set of all normed constants of Δ is denoted $Normed(\Delta)$. In our constructions we also use processes of the form αf; they should be seen as BPA processes with the underlying system $\Delta \cup \Gamma$.

Intuition: Our proof can be divided into two parts: first we show that the greatest weak bisimulation between processes of Δ and Γ is finitely representable. There is a finite relation $\mathcal{B}$ of size $\mathcal{O}(n\,m^2)$ (called *bisimulation base*) such that each pair of weakly bisimilar processes can be generated from that base (a technique first used by Caucal [6]). Then we show that the bisimulation base can be computed in polynomial time. To do that, we take a sufficiently large relation $\mathcal{G}$ which surely subsumes the base and 'refine' it (this refinement technique has been used in [17,18]). The size of $\mathcal{G}$ is still $\mathcal{O}(n\,m^2)$, and each step of the refinement procedure possibly deletes some of the elements of $\mathcal{G}$. If nothing is deleted, we have found the base (hence we need at most $\mathcal{O}(n\,m^2)$ steps). The refinement step is formally introduced in Definition 9 (we compute the *expansion* of the currently computed approximation of the base). Intuitively, a pair of processes belongs to the expansion iff for each $\stackrel{a}{\rightarrow}$ move of one component there is a $\stackrel{a}{\Rightarrow}$ move of the other component such that the

resulting pair of processes can be generated from the current approximation of $\mathcal{B}$. We have to overcome two problems:

1. The set of pairs which can be generated from $\mathcal{B}$ (and its approximations) is infinite.
2. The set of states which are reachable from a given BPA state in one '$\stackrel{a}{\Rightarrow}$' move is infinite.

We employ a 'symbolic' technique to represent those infinite sets (similar to the one used in [3]), taking advantage of the fact that they have a simple (regular) structure which can be encoded by finite-state automata (see Theorem 6 and 12). This allows to compute the expansion in polynomial time.

Definition 3 *A relation K is* well-formed *iff it is a subset of the relation $\mathcal{G}$ defined by*

$$\begin{aligned}\mathcal{G} = &((\mathit{Normed}(\Delta) \cdot \mathit{Const}(\Gamma)) \times \mathit{Const}(\Gamma)) \\ &\cup (\mathit{Const}(\Delta) \times \mathit{Const}(\Gamma)) \\ &\cup (\mathit{Const}(\Gamma) \times \mathit{Const}(\Gamma)) \\ &\cup (\{\varepsilon\} \times \mathit{Const}(\Gamma))\end{aligned}$$

Note that the size of any well-formed relation is $\mathcal{O}(n\,m^2)$ and that $\mathcal{G}$ is the greatest well-formed relation.

One of the well-formed relations is of special importance.

Definition 4 *The* bisimulation base *for Δ and Γ, denoted $\mathcal{B}$, is defined as follows:*

$$\begin{aligned}\mathcal{B} = &\{(Y\,f, g) \mid Y\,f \approx g,\, Y \in \mathit{Normed}(\Delta)\} \\ &\cup \{(X, g) \mid X \approx g\} \\ &\cup \{(f, g) \mid f \approx g\} \\ &\cup \{(\varepsilon, g) \mid \varepsilon \approx g\}\end{aligned}$$

As weak bisimilarity is a left congruence w.r.t. sequential composition, we can 'generate' from $\mathcal{B}$ new pairs of weakly bisimilar processes by substitution (it is worth noting that weak bisimilarity is *not* a right congruence w.r.t. sequencing—to see this, it suffices to define $X \stackrel{\tau}{\to} X$, $Y \stackrel{\tau}{\to} \varepsilon$, $Z \stackrel{a}{\to} Z$. Now $X \approx Y$, but $XZ \not\approx YZ$). This generation procedure can be defined for any well-formed relation as follows:

Definition 5 *Let K be a well-formed relation. The* closure *of K, denoted*

$Cl(K)$, is the least relation M which satisfies the following conditions:

(1) $K \subseteq M$,
(2) if $(f, g) \in K$ *and* $(\alpha, f) \in M$, *then* $(\alpha, g) \in M$,
(3) if $(f, g) \in K$ *and* $(\alpha h, f) \in M$, *then* $(\alpha h, g) \in M$,
(4) if $(Yf, g) \in K$ *and* $(\alpha, f) \in M$, *then* $(Y\alpha, g) \in M$,
(5) if $(Yf, g) \in K$ *and* $(\alpha h, f) \in M$, *then* $(Y\alpha h, g) \in M$,
(6) if $(\alpha, g) \in M$ *and* α *contains an unnormed constant, then* $(\alpha\beta, g), (\alpha\beta h, g) \in M$ *for every* $\beta \in Const(\Delta)^*$ *and* $h \in Const(\Gamma)$.

Note that $Cl(K)$ contains elements of just two forms – (α, g) and $(\alpha f, g)$. Clearly $Cl(K) = \bigcup_{i=0}^{\infty} Cl(K)^i$ where $Cl(K)^0 = K$ and $Cl(K)^{i+1}$ consists of $Cl(K)^i$ and the pairs which can be immediately derived from $Cl(K)^i$ by the rules 2–6 of Definition 5.

Although the closure of a well-formed relation can be infinite, its structure is in some sense regular. This fact is precisely formulated in the following theorem:

Theorem 6 *Let K be a well-formed relation. For each $g \in Const(\Gamma)$ there is a finite-state automaton $\mathcal{A}_g$ of size $\mathcal{O}(n\,m^2)$ constructible in $\mathcal{O}(n\,m^2)$ time such that $L(\mathcal{A}_g) = \{\alpha \mid (\alpha, g) \in Cl(K)\} \cup \{\alpha f \mid (\alpha f, g) \in Cl(K)\}$.*

PROOF. We construct a regular grammar of size $\mathcal{O}(n\,m^2)$ which generates the mentioned language. Let $G_g = (N, \Sigma, \delta, \overline{g})$ where

- $N = \{\overline{f} \mid f \in Const(\Gamma)\} \cup \{U\}$
- $\Sigma = Const(\Delta) \cup Const(\Gamma)$
- δ is defined as follows:
  - for each $(\varepsilon, h) \in K$ we add the rule $\overline{h} \to \varepsilon$.
  - for each $(f, h) \in K$ we add the rules $\overline{h} \to \overline{f}$, $\overline{h} \to f$.
  - for each $(Yf, h) \in K$ we add the rules $\overline{h} \to Yf$, $\overline{h} \to Y\overline{f}$.
  - for each $(X, h) \in K$ we add the rule $\overline{h} \to X$ and if X is unnormed, then we also add the rule $\overline{h} \to XU$.
  - for each $X \in Const(\Delta)$, $f \in Const(\Gamma)$ we add the rules $U \to XU, U \to X$, $U \to f$.

A proof that G_g indeed generates the mentioned language is routine. Now we translate G_g to $\mathcal{A}_g$ (see, e.g., [19]). Note that the size of $\mathcal{A}_g$ is essentially the same as the size of G_g; $\mathcal{A}_g$ is non-deterministic and can contain ε-rules.

It follows immediately that for any well-formed relation K, the membership problem for $Cl(K)$ is decidable in polynomial time. Another property of $Cl(K)$ is specified in the lemma below.

Lemma 7 *Let* $(\alpha f, g) \in Cl(K)$. *If* $(\beta h, f) \in Cl(K)$, *then also* $(\alpha\beta h, g) \in Cl(K)$. *Similarly, if* $(\beta, f) \in Cl(K)$, *then also* $(\alpha\beta, g) \in Cl(K)$.

PROOF. We just give a proof for the first claim (the second one is similar). Let $(\alpha f, g) \in Cl(K)^i$. By induction on i.

- $i = 0$. Then $(\alpha f, g) \in K$ and we can immediately apply the rule 3 or 5 of Definition 5 (remember that α can be ε).
- **Induction step.** Let $(\alpha f, g) \in Cl(K)^{i+1}$. There are three possibilities (cf. Definition 5).

I. There is r such that $(\alpha f, r) \in Cl(K)^i$, $(r, g) \in K$. By induction hypothesis we know $(\alpha\beta h, r) \in Cl(K)$, hence $(\alpha\beta h, g) \in Cl(K)$ due to the rule 3 of Definition 5.

II. $\alpha = Y\gamma$ and there is r such that $(Yr, g) \in K$, $(\gamma f, r) \in Cl(K)^i$. By induction hypothesis we have $(\gamma\beta h, r) \in Cl(K)$, and hence also $(Y\gamma\beta h, r) \in Cl(K)$ by the rule 5 of Definition 5.

III. $\alpha = \gamma\delta$ where $(\gamma, g) \in Cl(K)^i$ and γ contains an unnormed constant. Then $(\gamma\delta\beta h, g) \in Cl(K)$ by the last rule of Definition 5.

The importance of the bisimulation base is clarified by the following theorem. It says that $Cl(\mathcal{B})$ subsumes the greatest weak bisimulation between processes of Δ and Γ.

Theorem 8 *For all* α, f, g *we have* $\alpha \approx g$ *iff* $(\alpha, g) \in Cl(\mathcal{B})$, *and* $\alpha f \approx g$ *iff* $(\alpha f, g) \in Cl(\mathcal{B})$.

PROOF. The 'if' part is obvious in both cases, as $\mathcal{B}$ contains only weakly bisimilar pairs and all the rules of Definition 5 produce pairs which are again weakly bisimilar. The 'only if' part can, in both cases, be easily proved by induction on the length of α (we just show the first proof; the second one is similar).

- $\alpha = \varepsilon$. Then $(\varepsilon, g) \in \mathcal{B}$, hence $(\varepsilon, g) \in Cl(\mathcal{B})$.
- $\alpha = Y\beta$. If Y is unnormed, then $Y \approx g$ and $(Y, g) \in \mathcal{B}$. By the rule 6 of Definition 5 we obtain $(Y\beta, g) \in Cl(\mathcal{B})$. If Y is normed, then $Y\beta \stackrel{w}{\rightarrow} \beta$ for some $w \in Act^*$ and g must be able to match the sequence w by some $g \stackrel{w}{\Rightarrow} g'$ such that $\beta \approx g'$. By substitution we now obtain that $Yg' \approx g$. Clearly $(Yg', g) \in \mathcal{B}$, and $(\beta, g') \in Cl(\mathcal{B})$ by induction hypothesis. Hence $(\alpha, g) \in Cl(\mathcal{B})$ due to the rule 4 of Definition 5.

The next definition formalizes one step of the 'refinement procedure' which is applied to $\mathcal{G}$ to compute $\mathcal{B}$. The intuition is that we start with $\mathcal{G}$ as an

approximation to $\mathcal{B}$. In each refinement step some pairs are deleted from the current approximation. If in a refinement step no pairs are deleted any more then we have found $\mathcal{B}$. The next definition specifies the condition on which a given pair is *not* deleted in a refinement step from the currently computed approximation of $\mathcal{B}$.

Definition 9 *Let K be a well-formed relation. We say that a pair (X, g) of K* expands in K *iff the following two conditions hold:*

- *for each $X \stackrel{a}{\rightarrow} \alpha$ there is some $g \stackrel{a}{\Rightarrow} g'$ such that $(\alpha, g') \in Cl(K)$*
- *for each $g \stackrel{a}{\rightarrow} g'$ there is some $X \stackrel{a}{\Rightarrow} \alpha$ such that $(\alpha, g') \in Cl(K)$*

The expansion of a pair of the form (Yf, g), (f, g), (ε, g) in K is defined in the same way—for each '$\stackrel{a}{\rightarrow}$' move of the left component there must be some '$\stackrel{a}{\Rightarrow}$' move of the right component such that the resulting pair of processes belongs to $Cl(K)$, and vice versa (note that $\varepsilon \stackrel{\tau}{\Rightarrow} \varepsilon$). The set of all pairs of K which expand in K is denoted by $Exp(K)$.

The notion of expansion is in some sense 'compatible' with the definition of weak bisimulation. This intuition is formalized in the following lemma.

Lemma 10 *Let K be a well-formed relation such that $Exp(K) = K$. Then $Cl(K)$ is a weak bisimulation.*

PROOF. We prove that every pair (α, g), $(\alpha f, g)$ of $Cl(K)^i$ has the property that for each '$\stackrel{a}{\rightarrow}$' move of one component there is a '$\stackrel{a}{\Rightarrow}$' move of the other component such that the resulting pair of processes belongs to $Cl(K)$ (we consider just pairs of the form $(\alpha f, g)$; the other case is similar). By induction on i.

- $i = 0$. Then $(\alpha f, g) \in K$; as $K = Exp(K)$, the claim follows directly from the definitions.
- **Induction step.** Let $(\alpha f, g) \in Cl(K)^{i+1}$. There are three possibilities:

I. There is an h such that $(\alpha f, h) \in Cl(K)^i$, $(h, g) \in K$.

Let $\alpha f \stackrel{a}{\rightarrow} \gamma f$ (note that α can be empty; in this case we have to consider moves of the form $f \stackrel{a}{\rightarrow} f'$. It is done in a similar way as below). As $(\alpha f, h) \in Cl(K)^i$, we can use the induction hypothesis and conclude that there is $h \stackrel{a}{\Rightarrow} h'$ such that $(\gamma f, h') \in Cl(K)$. We distinguish two cases:
1) $a = \tau$ and $h' = h$. Then $(\gamma f, h) \in Cl(K)$ and as $(h, g) \in K$, we obtain $(\gamma f, g) \in Cl(K)$ due to Lemma 7. Hence g can use the move $g \stackrel{\tau}{\Rightarrow} g$.
2) $a \neq \tau$ or $h \neq h'$. Then there is a transition $h \stackrel{a}{\rightarrow} h'$ (see the beginning of this section) and as $(h, g) \in K$, by induction hypothesis we know that there is some $g \stackrel{a}{\Rightarrow} g'$ such that $(h', g') \in Cl(K)$. Hence, $(\gamma f, g') \in Cl(K)$ due to Lemma 7.

Now let $g \stackrel{a}{\rightarrow} g'$. As $(h, g) \in K$, there is $h \stackrel{a}{\Rightarrow} h'$ such that $(h', g') \in$

$Cl(K)$. We distinguish two possibilities again:
1) $a = \tau$ and $h' = h$. Then αf can use the move $\alpha f \stackrel{\tau}{\Rightarrow} \alpha f$; we have $(h, g') \in Cl(K)$ and $(\alpha f, h) \in Cl(K)$, hence also $(\alpha f, g') \in Cl(K)$.
2) $a \neq \tau$ or $h \neq h'$. Then $h \stackrel{a}{\rightarrow} h'$ and as $(\alpha f, h) \in Cl(K)^i$, there is $\alpha f \stackrel{a}{\Rightarrow} \gamma f$ (or $\alpha f \stackrel{a}{\Rightarrow} f'$; it is handled in the same way) such that $(\gamma f, h') \in Cl(K)$. Hence also $(\gamma f, g') \in Cl(K)$ by Lemma 7.

II. $\alpha = Y\beta$ and there is h such that $(Yh, g) \in K$, $(\beta f, h) \in Cl(K)^i$.
Let $Y\beta f \stackrel{a}{\rightarrow} \gamma\beta f$. As $(Yh, g) \in K$, we can use induction hypothesis and conclude that there is $g \stackrel{a}{\Rightarrow} g'$ such that $(\gamma h, g') \in Cl(K)$. As $(\beta f, h) \in Cl(K)$, we obtain $(\gamma\beta f, g') \in Cl(K)$ by Lemma 7.
Let $g \stackrel{a}{\rightarrow} g'$. As $(Yh, g) \in K$, by induction hypothesis we know that Yh can match the move $g \stackrel{a}{\rightarrow} g'$; there are two possibilities:
1) $Yh \stackrel{a}{\Rightarrow} \gamma h$ such that $(\gamma h, g') \in Cl(K)$. Then also $Y\beta f \stackrel{a}{\Rightarrow} \gamma\beta f$. As $(\beta f, h) \in Cl(K)$, we immediately have $(\gamma\beta f, g') \in Cl(K)$ as required.
2) $Yh \stackrel{a}{\Rightarrow} h'$ such that $(h', g') \in Cl(K)$. The transition $Yh \stackrel{a}{\Rightarrow} h'$ can be 'decomposed' into $Yh \stackrel{x}{\Rightarrow} h$, $h \stackrel{y}{\Rightarrow} h'$ where $x = a \wedge y = \tau$ or $x = \tau \wedge y = a$. If $y = \tau$ and $h' = h$, we are done immediately because then $Y\beta \stackrel{a}{\Rightarrow} \beta$ and as (h, g'), $(\beta, h) \in Cl(K)$, we also have $(\beta, g') \in Cl(K)$ as needed. If $y \neq \tau$ or $h' \neq h$, there is a transition $h \stackrel{y}{\rightarrow} h'$. As $(\beta f, h) \in Cl(K)^i$, due to induction hypothesis we know that there is some $\beta f \stackrel{y}{\Rightarrow} \gamma f$ (or $\beta f \stackrel{y}{\Rightarrow} f'$; this is handled in the same way) with $(\gamma f, h') \in Cl(K)$. Clearly $Y\beta f \stackrel{a}{\Rightarrow} \gamma f$. As (h', g'), $(\gamma f, h') \in Cl(K)$, we also have $(\gamma f, g') \in Cl(K)$.

III. $\alpha = \beta\gamma$ where β contains an unnormed constant and $(\beta, g) \in Cl(K)^i$.
Let $\alpha \stackrel{a}{\rightarrow} \alpha'$. Then $\alpha' = \delta\gamma$ and $\beta \stackrel{a}{\rightarrow} \delta$. As $(\beta, g) \in Cl(K)^i$, there is $g \stackrel{a}{\Rightarrow} g'$ such that $(\delta, g') \in Cl(K)$ due to the induction hypothesis. Clearly δ contains an unnormed constant, hence $(\delta\gamma, g') \in Cl(K)$ by the last rule of Definition 5.
Let $g \stackrel{a}{\rightarrow} g'$. As $(\beta, g) \in Cl(K)^i$, there is $\beta \stackrel{a}{\Rightarrow} \delta$ such that $(\delta, g') \in Cl(K)$ and δ contains an unnormed constant. Hence $\alpha \stackrel{a}{\Rightarrow} \delta\gamma$ and $(\delta\gamma, g') \in Cl(K)$ due to the last rule of Definition 5.

The notion of expansion allows to approximate $\mathcal{B}$ in the following way: $\mathcal{B}^0 = \mathcal{G}$, $\mathcal{B}^{i+1} = Exp(\mathcal{B}^i)$.

Theorem 11 *There is a $j \in \mathbb{N}$, bounded by $\mathcal{O}(n\,m^2)$, such that $\mathcal{B}^j = \mathcal{B}^{j+1}$. Moreover, $\mathcal{B}^j = \mathcal{B}$.*

PROOF. *Exp* (viewed as a function on the complete lattice of well-formed relations) is monotonic, hence the greatest fixed-point exists and must be reached after $\mathcal{O}(n\,m^2)$ steps, as the size of $\mathcal{G}$ is $\mathcal{O}(n\,m^2)$. We prove that $\mathcal{B}^j = \mathcal{B}$.
'$\supseteq$:' First, let us realize that $\mathcal{B} = Exp(\mathcal{B})$ (it follows immediately from Definition 4, Definition 9, and Theorem 8). The inclusion $\mathcal{B} \subseteq \mathcal{B}^j$ can be proved

by a simple inductive argument; clearly $\mathcal{B} \subseteq \mathcal{B}^0$, and if $\mathcal{B} \subseteq \mathcal{B}^i$, we also have $\mathcal{B} \subseteq \mathcal{B}^{i+1}$ by definition of the expansion and the fact $\mathcal{B} = \mathit{Exp}(\mathcal{B})$.

'$\subseteq$:' As $\mathit{Exp}(\mathcal{B}^j) = \mathcal{B}^j$, we know that $\mathit{Cl}(\mathcal{B}^j)$ is a weak bisimulation due to Lemma 10. Thus, processes of every pair in $\mathcal{B}^j$ are weakly bisimilar.

In other words, $\mathcal{B}$ can be obtained from $\mathcal{G}$ in $\mathcal{O}(n\,m^2)$ refinement steps which correspond to the construction of the expansion. The only thing which remains to be shown is that $\mathit{Exp}(K)$ is effectively constructible in polynomial time. To do that, we employ a 'symbolic' technique which allows to represent infinite subsets of BPA state-space in an elegant and succinct way.

Theorem 12 *For all $X \in \mathit{Const}(\Delta)$, $a \in \mathit{Act}(\Delta)$ there is a finite-state automaton $\mathcal{A}_{(X,a)}$ of size $\mathcal{O}(n^2)$ constructible in $\mathcal{O}(n^2)$ time such that $L(\mathcal{A}_{(X,a)}) = \{\alpha \mid X \stackrel{a}{\Rightarrow} \alpha\}$*

PROOF. We define a left-linear grammar $G_{(X,a)}$ of size $\mathcal{O}(n^2)$ which generates the mentioned language. This grammar can be converted to $\mathcal{A}_{(X,a)}$ by a standard algorithm known from automata theory (see, e.g., [19]). Note that the size of $\mathcal{A}_{(X,a)}$ is essentially the same as the size of $G_{(X,a)}$. First, let us realize that we can compute in $\mathcal{O}(n^2)$ time the sets M_τ and M_a consisting of all $Y \in \mathit{Const}(\Delta)$ such that $Y \stackrel{\tau}{\Rightarrow} \varepsilon$ and $Y \stackrel{a}{\Rightarrow} \varepsilon$, respectively. Let $G_{(X,a)} = (N, \Sigma, \delta, S)$ where

- $N = \{Y^a, Y^\tau \mid Y \in \mathit{Const}(\Delta)\} \cup \{S\}$. Intuitively, the index indicates whether the action 'a' has already been emitted.
- $\Sigma = \mathit{Const}(\Delta)$
- δ is defined as follows:
  - We add the production $S \to X^a$ to δ, and if $X \stackrel{a}{\Rightarrow} \varepsilon$ then we also add the production $S \to \varepsilon$.
  - For every transition $Y \stackrel{a}{\to} Z_1 \cdots Z_k$ of Δ and every i such that $1 \leq i \leq k$ we test whether $Z_j \stackrel{\tau}{\Rightarrow} \varepsilon$ for every $1 \leq j < i$. If this is the case, we add to δ the productions
    $$Y^a \to Z_i Z_{i+1} \cdots Z_k \text{ and } Y^a \to Z_i^\tau Z_{i+1} \cdots Z_k$$
  - For every transition $Y \stackrel{\tau}{\to} Z_1 \cdots Z_k$ of Δ and every i such that $1 \leq i \leq k$ we do the following:
    We test whether $Z_j \stackrel{\tau}{\Rightarrow} \varepsilon$ for every $1 \leq j < i$. If this is the case, we add to δ the productions
    $$Y^a \to Z_i^a Z_{i+1} \cdots Z_k,\ Y^\tau \to Z_i^\tau Z_{i+1} \cdots Z_k \text{ and } Y^\tau \to Z_i Z_{i+1} \cdots Z_k$$
    We test whether there is a $t < i$ such that $Z_t \stackrel{a}{\Rightarrow} \varepsilon$ and $Z_j \stackrel{\tau}{\Rightarrow} \varepsilon$ for every $1 \leq j < i$, $j \neq t$. If this is the case, we add to δ the productions
    $$Y^a \to Z_i^\tau Z_{i+1} \cdots Z_k \text{ and } Y^a \to Z_i Z_{i+1} \cdots Z_k$$

The fact that $G_{(X,a)}$ generates the mentioned language is intuitively clear and

a formal proof of that is easy. The size of $G_{(X,a)}$ is $\mathcal{O}(n^2)$, as Δ contains $\mathcal{O}(n)$ basic transitions of length $\mathcal{O}(n)$.

The crucial part of our algorithm (the 'refinement step') is presented in the proof of the next theorem. Our complexity analysis is based on the following facts: Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a non-deterministic automaton with ε-rules, and let t be the total number of states and transitions of $\mathcal{A}$.

- The problem whether a given $w \in \Sigma^*$ belongs to $L(\mathcal{A})$ is decidable in $\mathcal{O}(|w| \cdot t)$ time.
- The problem whether $L(\mathcal{A}) = \emptyset$ is decidable in $\mathcal{O}(t)$ time.

Theorem 13 *Let K be a well-formed relation. The relation $Exp(K)$ can be effectively constructed in $\mathcal{O}(n^4\, m^5)$ time.*

PROOF. First we construct the automata $\mathcal{A}_g$ of Theorem 6 for every $g \in Const(\Gamma)$. This takes $\mathcal{O}(n\, m^3)$ time. Then we construct the automata $\mathcal{A}_{(X,a)}$ of Theorem 12 for all X, a. This takes $\mathcal{O}(n^4)$ time. Furthermore, we also compute the set of all pairs of the form $(f, g), (\varepsilon, g)$ which belong to $Cl(K)$. It can be done in $\mathcal{O}(m^2)$ time. Now we show that for each pair of K we can decide in $\mathcal{O}(n^3\, m^3)$ time whether this pair expands in K.

The pairs of the form (f, g) and (ε, g) are easy to handle; there are at most m states f' such that $f \xrightarrow{a} f'$, and at most m states g' with $g \stackrel{a}{\Rightarrow} g'$, hence we need to check only $\mathcal{O}(m^2)$ pairs to verify the first (and consequently also the second) condition of Definition 9. Each such pair can be checked in constant time, because the set of all pairs $(f, g), (\varepsilon, g)$ which belong to $Cl(K)$ has already been computed at the beginning.

Now let us consider a pair of the form (Y, g). First we need to verify that for each $Y \xrightarrow{a} \alpha$ there is some $g \stackrel{a}{\Rightarrow} h$ such that $(\alpha, h) \in Cl(K)$. This requires $\mathcal{O}(n\, m)$ tests whether $\alpha \in L(\mathcal{A}_h)$. As the length of α is $\mathcal{O}(n)$ and the size of $\mathcal{A}_h$ is $\mathcal{O}(n\, m^2)$, each such test can be done in $\mathcal{O}(n^2\, m^2)$ time, hence we need $\mathcal{O}(n^3\, m^3)$ time in total. As for the second condition of Definition 9, we need to find out whether for each $g \xrightarrow{a} h$ there is some $X \stackrel{a}{\Rightarrow} \alpha$ such that $(\alpha, h) \in Cl(K)$. To do that, we simply test the emptiness of $L(\mathcal{A}_{(X,a)}) \cap L(\mathcal{A}_h)$. The size of the product automaton is $\mathcal{O}(n^3\, m^2)$ and we need to perform only $\mathcal{O}(m)$ such tests, hence $\mathcal{O}(n^3\, m^3)$ time suffices.

Pairs of the form (Yf, g) are handled in a similar way; the first condition of Definition 9 is again no problem, as we are interested only in the '$\xrightarrow{a}$' moves of the left component. Now let $g \xrightarrow{a} g'$. An existence of a 'good' $\stackrel{a}{\Rightarrow}$ move of

$Y f$ can be verified by testing whether one of the following conditions holds:

- $L(\mathcal{A}_{(Y,a)}) \cdot \{f\} \cap L(\mathcal{A}_{g'})$ is nonempty.
- $Y \stackrel{a}{\Rightarrow} \varepsilon$ and there is some $f \stackrel{\tau}{\Rightarrow} f'$ such that $(f', g') \in Cl(K)$.
- $Y \stackrel{\tau}{\Rightarrow} \varepsilon$ and there is some $f \stackrel{a}{\Rightarrow} f'$ such that $(f', g') \in Cl(K)$.

All those conditions can be checked in $\mathcal{O}(n^3\, m^3)$ time (the required analysis has been in fact done above). As K contains $\mathcal{O}(n\, m^2)$ pairs, the total time which is needed to compute $Exp(K)$ is $\mathcal{O}(n^4\, m^5)$.

As the BPA process E (introduced at the beginning of this section) is an element of $Const(\Delta)$, we have that $E \approx F$ iff $(E, F) \in \mathcal{B}$. To compute $\mathcal{B}$, we have to perform the computation of the expansion $\mathcal{O}(n\, m^2)$ times (see Theorem 11). This gives us the following main theorem:

Theorem 14 *Weak bisimilarity is decidable between BPA and finite-state processes in $\mathcal{O}(n^5\, m^7)$ time.*

4 Termination-Sensitive Bisimilarity

As we already mentioned in the previous section, weak bisimilarity is not a congruence w.r.t. sequential composition. This is a major drawback, as any equivalence which is to be considered as 'behavioral' should have this property. We propose a solution to this problem by designing a natural refinement of weak bisimilarity called *termination-sensitive bisimilarity*. This relation respects some of the main features of sequencing which are 'overlooked' by weak bisimilarity; consequently, it is a congruence w.r.t. sequential composition. We also show that termination-sensitive bisimilarity is decidable between BPA and finite-state processes in polynomial time by adapting the method of the previous section. It should be noted right at the beginning that we do *not* aim to design any new 'fundamental' notion of the theory of sequential processes (that is why the properties of termination-sensitive bisimilarity are not studied in detail). We just want to demonstrate that our method is applicable to a larger class of bisimulation-like equivalences and the relation of termination-sensitive bisimilarity provides a (hopefully) convincing evidence that some of them might be interesting and useful.

In our opinion, any 'reasonable' model of sequential behaviors should be able to express (and distinguish) the following 'basic phenomena' of sequencing:

- *successful termination* of the process which is currently being executed. The system can then continue to execute the next process in the queue;

- *unsuccessful termination* of the executed process (deadlock). This models a severe error which causes the whole system to 'get stuck';
- *entering an infinite internal loop* (cycling).

The difference between successful and unsuccessful termination is certainly significant. The need to distinguish between termination and cycling has also been recognized in practice; major examples come, e.g., from the theory of operating systems.

BPA processes are a very natural model of recursive sequential behaviors. Successful termination is modeled by reaching 'ε'. There is also a 'hidden' syntactical tool to model deadlock—note that by the definition of BPA systems there can be an $X \in Const(\Delta)$ such that Δ does not contain any rule of the form $X \stackrel{a}{\rightarrow} \alpha$ (let us call such constants *undefined*). A state $X\beta$ models the situation when the executed process reaches a deadlock—there is no transition (no computational step) from $X\beta$, the process is 'stuck'. It is easy to see that we can safely assume that Δ contains at most *one* undefined constant (the other ones can be simply renamed to X), which is denoted δ by convention [2]. Note that δ is *unnormed* by definition. States of the form $\delta\alpha$ are called *deadlocked*.

In the case of finite-state systems, we can distinguish between successful and unsuccessful termination in a similar way. Deadlock is modeled by a distinguished undefined constant δ, and the other undefined constants model successful termination.

Note that $\delta \approx \varepsilon$ by definition of weak bisimilarity. As 'ε' represents a successful termination, this is definitely not what we want. Before we define the promised relation of termination-sensitive bisimilarity, we need to clarify what is meant by cycling; intuitively, it is the situation when a process enters an infinite internal loop. In other words, it can do 'τ' forever without a possibility to do anything else or to terminate (either successfully or unsuccessfully).

Definition 15 *The set of* initial actions *of a process E, denoted $I(E)$, is defined by $I(E) = \{a \in Act \mid E \stackrel{a}{\rightarrow} F \text{ for some } F\}$. A process E is* cycling *iff every state F which is reachable from E satisfies $I(F) = \{\tau\}$.*

Note that it is easily decidable in quadratic time whether a given BPA process is cycling; in the case of finite-state systems we only need linear time.

Definition 16 *We say that an expression E is* normal *iff E is not cycling, deadlocked, or successfully terminated.*

A binary relation R over process expressions is a termination-sensitive bisi-

mulation *iff whenever* $(E, F) \in R$ *then the following conditions hold:*

- *if one of the expressions* E, F *is cycling then the other is also cycling;*
- *if one of the expressions* E, F *is deadlocked then the other is either normal or it is also deadlocked;*
- *if one of the expressions* E, F *is successfully terminated then the other is either normal or it is also successfully terminated;*
- *if* $E \stackrel{a}{\rightarrow} E'$ *then there is* $F \stackrel{a}{\Rightarrow} F'$ *such that* $(E', F') \in R$*;*
- *if* $F \stackrel{a}{\rightarrow} F'$ *then there is* $E \stackrel{a}{\Rightarrow} E'$ *such that* $(E', F') \in R$*.*

Processes E, F *are* termination-sensitive bisimilar*, written* $E \simeq F$*, iff there is a termination-sensitive bisimulation relating them.*

Termination-sensitive bisimilarity seems to be a natural refinement of weak bisimilarity which better captures an intuitive understanding of 'sameness' of sequential processes. It distinguishes among the phenomena mentioned at the beginning of this section, but it still allows to ignore internal computational steps to a large extent. For example, a deadlocked process is still equivalent to a process which is not deadlocked yet but which *necessarily* deadlocks after a finite number of τ transitions (this example also explains why the first three conditions of Definition 16 are stated so carefully).

The family of $\simeq_i$ approximations is defined in the same way as in case of weak bisimilarity; the only difference is that $\simeq_0$ relates exactly those processes which satisfy the first three conditions of Definition 16. The following theorem follows immediately from this definition.

Theorem 17 *Termination-sensitive bisimilarity is a congruence w.r.t. sequential composition.*

The technique which has been used in the previous section also works for termination-sensitive bisimilarity.

Theorem 18 *Termination-sensitive bisimilarity is decidable between BPA and finite-state processes in* $\mathcal{O}(n^5\, m^7)$ *time.*

PROOF. First, all assumptions about Δ and Γ which were mentioned at the beginning of Section 3 are also safe w.r.t. termination-sensitive bisimilarity; note that it would not be true if we also assumed the existence of a τ-loop $f \stackrel{\tau}{\rightarrow} f$ for every $f \in \mathit{Const}(\Gamma)$. Now we see why the assumptions about Γ are formulated so carefully. The only thing which has to be modified is the notion of well-formed relation; it is defined in the same way, but in addition we require that processes of every pair which is contained in a well-formed relation K are related by $\simeq_0$. It can be easily shown that processes of pairs contained in $\mathit{Cl}(K)$ are then also related by $\simeq_0$. In other words, we do not

have to take care about the first two requirements of Definition 16 in our constructions anymore; everything works without a single change.

The previous proof indicates that the 'method' of Section 3 can be adapted to other bisimulation-like equivalences. See the final section for further comments.

5 Normed BPP Processes

In this section we prove that weak bisimilarity is decidable in polynomial time between normed BPP and finite-state processes. The basic structure of our proof is similar to the one for BPA. The key is that the weak bisimulation problem can be decomposed into problems about the single constants and their interaction with each other. In particular, a normed BPP process is finite w.r.t. weak bisimilarity iff every single reachable process constant is finite w.r.t. weak bisimilarity. This does not hold for general BPP and thus our construction does not carry over to general BPP.

Example 19 *Consider the unnormed BPP that is defined by the following rules.*

$$X_i \stackrel{a_{i+1}}{\longrightarrow} X_i \| Y_i, \quad Y_i \stackrel{a_i}{\longrightarrow} \varepsilon \ \textit{for}\ 1 \leq i \leq n-1$$
$$X_n \stackrel{a_1}{\longrightarrow} X_n \| Y_n,\ Y_n \stackrel{a_n}{\longrightarrow} \varepsilon$$

Then the process $X_1 \| X_2 \| \ldots \| X_n$ is finite w.r.t. bisimilarity, but every subprocess (e.g. $X_3 \| X_4 \| X_7$ or every single constant X_i) is infinite w.r.t. bisimilarity.

Even for normed BPP, we have to solve some additional problems. The bisimulation base and its closure are simpler due to the normedness assumption, but the 'symbolic' representation of BPP state-space is more problematic (see below). The set of states which are reachable from a given BPP state in one '$\stackrel{a}{\Rightarrow}$' move is no longer regular, but it can be in some sense represented by a CF-grammar. In our algorithm we use the facts that emptiness of a CF language is decidable in polynomial time, and that CF languages are closed under intersection with regular languages.

Let E be a BPP process and F a finite-state process with the underlying systems Δ and Γ, respectively. We can assume w.l.o.g. that $E \in \mathit{Const}(\Delta)$. Elements of $\mathit{Const}(\Delta)$ are denoted by $X, Y, Z, \ldots$, elements of $\mathit{Const}(\Gamma)$ by $f, g, h, \ldots$ The set of all parallel expressions over $\mathit{Const}(\Delta)$ is denoted by $\mathit{Const}(\Delta)^{\otimes}$ and its elements by Greek letters $\alpha, \beta, \ldots$ The size of Δ is denoted by n, and the size of Γ by m.

In our constructions we represent certain subsets of $Const(\Delta)^{\otimes}$ by finite automata and CF grammars. The problem is that elements of $Const(\Delta)^{\otimes}$ are considered modulo commutativity; however, finite automata and CF grammars of course distinguish between different 'permutations' of the same word. As the classes of regular and CF languages are not closed under permutation, this problem is important. As we want to clarify the distinction between α and its possible 'linear representations', we define for each α the set $Lin(\alpha)$ as follows:

$$Lin(X_1 \| \cdots \| X_k) = \{X_{p(1)} \cdots X_{p(k)} \mid p \text{ is a permutation of the set } \{1, \cdots, k\}\}$$

For example, $Lin(X\|Y\|Z) = \{XYZ, XZY, YXZ, YZX, ZXY, ZYX\}$. We also assume that each $Lin(\alpha)$ contains some (unique) element called *canonical form* of $Lin(\alpha)$. It is not important how the canonical form is chosen; we need it just to make some constructions deterministic (for example, we can fix some linear order on process constants and let the canonical form of $Lin(\alpha)$ be the sorted order of constants of α).

Definition 20 *A relation K is* well-formed *iff it is a subset of $\mathcal{G} = (Const(\Delta) \cup \{\varepsilon\}) \times Const(\Gamma)$. The* bisimulation base *for Δ and Γ, denoted $\mathcal{B}$, is defined as follows:*

$$\mathcal{B} = \{(X, f) \mid X \approx f\} \cup \{(\varepsilon, f) \mid \varepsilon \approx f\}$$

Definition 21 *Let K be a well-formed relation. The* closure *of K, denoted $Cl(K)$, is the least relation M which satisfies*

(1) $K \subseteq M$,
(2) if $(X, g) \in K$, $(\beta, h) \in M$, *and* $f \approx g\|h$, *then* $(\beta\|X, f) \in M$,
(3) if $(\varepsilon, g) \in K$, $(\beta, h) \in M$, *and* $f \approx g\|h$, *then* $(\beta, f) \in M$.

The family of $Cl(K)^i$ approximations is defined in the same way as in Section 3.

Lemma 22 *Let $(\alpha, f) \in Cl(K)$, $(\beta, g) \in Cl(K)$, $f\|g \approx h$. Then $(\alpha\|\beta, h) \in Cl(K)$.*

PROOF. Let $(\alpha, f) \in Cl(K)^i$. By induction on i.

- $i = 0$. Then $(\alpha, f) \in K$ and we can immediately apply the rule 2 or 3 of Definition 21.
- **Induction step.** Let $(\alpha, f) \in Cl(K)^{i+1}$. There are two possibilities.

  I. $\alpha = X\|\gamma$ and there are r, s such that $(X, r) \in K$, $(\gamma, s) \in Cl(K)^i$, and $r\|s \approx f$. Clearly $r\|s\|g \approx h$, hence also $s\|g \approx t$ for some t. By induction

hypothesis we have $(\gamma\|\beta, t) \in Cl(K)$. Now $(X\|\gamma\|\beta, h) \in Cl(K)$ due to the second rule of Definition 21 (note that $r\|t \approx h$).

II. $(\alpha, r) \in Cl(K)^i$ and there is some s such that $(\varepsilon, s) \in K$ and $r\|s \approx f$. As $r\|s\|g \approx h$, there is some t such that $r\|g \approx t$. By induction hypothesis we obtain $(\alpha\|\beta, t) \in Cl(K)$, and hence $(\alpha\|\beta, h) \in Cl(K)$ due to the third rule of Definition 21.

Again, the closure of the bisimulation base is the greatest weak bisimulation between processes of Δ and Γ.

Theorem 23 *Let $\alpha \in Const(\Delta)^{\otimes}$, $f \in Const(\Gamma)$. We have that $\alpha \approx f$ iff $(\alpha, f) \in Cl(\mathcal{B})$.*

PROOF. The 'if' part is obvious. The 'only if' part can be proved by induction on $length(\alpha)$.

- $\alpha = \varepsilon$. Then $(\varepsilon, f) \in \mathcal{B}$.
- $\alpha = X\|\beta$. As Δ is normed and $X\|\beta \approx f$, there are $w, v \in Act^*$ such that $X\|\beta \stackrel{w}{\rightarrow} \beta$, $X\|\beta \stackrel{v}{\rightarrow} X$. The process f must be able to match the sequences w, v by entering weakly bisimilar states—there are $g, h \in Const(\Delta)$ such that $\beta \approx g$, $X \approx h$, and consequently also $f \approx g\|h$ (here we need the fact that weak bisimilarity is a congruence w.r.t. the parallel operator). Clearly $(X, h) \in \mathcal{B}$ and $(\beta, g) \in Cl(\mathcal{B})$ by induction hypothesis, hence $(X\|\beta, f) \in Cl(\mathcal{B})$ by Definition 21.

The closure of any well-formed relation can in some sense be represented by a finite-state automaton, as stated in the next theorem. For this construction we first need to compute the set $\{(f\|g, h) \mid f\|g \approx h\}$. We consider the parallel composition of the finite-state system with itself, i.e., the states of this system are of the form $f\|g$. Let our new system be the union of this system with the old system. The new system has size $\mathcal{O}(m^2)$ and its states are of the form $f\|g$ or h. Then we apply the usual cubic-time partition refinement algorithm to decide bisimilarity on the new system (see Section 2). This gives us the set $\{(f\|g, h) \mid f\|g \approx h\}$ in $\mathcal{O}(m^6)$ time.

Theorem 24 *Let K be a well-formed relation. For each $g \in Const(\Gamma)$ there is a finite-state automaton $\mathcal{A}_g$ of size $\mathcal{O}(n\,m)$ constructible in $\mathcal{O}(n\,m)$ time such that the following conditions hold:*

- *whenever $\mathcal{A}_g$ accepts an element of $Lin(\alpha)$, then $(\alpha, g) \in Cl(K)$*
- *if $(\alpha, g) \in Cl(K)$, then $\mathcal{A}_g$ accepts at least one element of $Lin(\alpha)$*

PROOF. We design a regular grammar of size $\mathcal{O}(n\,m)$ such that $L(G_g)$ has the mentioned properties. Let $G_g = (N, \Sigma, \delta, S)$ where

- $N = \mathit{Const}(\Gamma) \cup \{S\}$
- $\Sigma = \mathit{Const}(\Delta)$
- δ is defined as follows:
  - for each $(X, f) \in K$ we add the rule $S \to Xf$.
  - for each $(\varepsilon, f) \in K$ we add the rule $S \to f$.
  - for all $f, r, s \in \mathit{Const}(\Gamma)$, $X \in \mathit{Const}(\Delta)$ such that $(X, r) \in K$, $f \approx r \| s$ we add the rule $s \to Xf$.
  - for all $f, r, s \in \mathit{Const}(\Gamma)$ such that $(\varepsilon, r) \in K$, $f \approx r \| s$ we add the rule $s \to f$.
  - we add the rule $g \to \varepsilon$.

The first claim follows from an observation that whenever we have $\overline{\alpha} \in \mathit{Lin}(\alpha)$ such that $\overline{\alpha} f$ is a sentence of G_g, then $(\alpha, f) \in \mathit{Cl}(K)$. This can be easily proved by induction on the length of the derivation of $\overline{\alpha} f$. For the second part, it suffices to prove that if $(\alpha, f) \in \mathit{Cl}(K)^i$, then there is $\overline{\alpha} \in \mathit{Lin}(\alpha)$ such that $\overline{\alpha} f$ is a sentence of G_g. It can be done by a straightforward induction on i.

It is important to realize that if $(\alpha, g) \in \mathit{Cl}(K)$, then $\mathcal{A}_g$ does not necessarily accept *all* elements of $\mathit{Lin}(\alpha)$. For example, if $K = \{(X, f), (Y, r), (Z, h)\}$, $\mathit{Const}(\Gamma) = \{f, g, h, r, s\}$ with $f \| r \approx s$, $s \| h \approx g$, and $f \| h \not\approx p$ for any $p \in \mathit{Const}(\Gamma)$, then $\mathcal{A}_g$ accepts the string XYZ but not the string XZY. Generally, $\mathcal{A}_g$ cannot be 'repaired' to do so (see the beginning of this section); however, there is actually no need for such 'repairs', because $\mathcal{A}_g$ has the following nice property:

Lemma 25 *Let K be a well-formed relation such that $\mathcal{B} \subseteq K$. If $\alpha \approx g$, then the automaton $\mathcal{A}_g$ of (the proof of) Theorem 24 constructed for K accepts all elements of $\mathit{Lin}(\alpha)$.*

PROOF. Let G_g be the grammar of the previous proof. First we prove that for all $s, r, f \in \mathit{Const}(\Gamma)$, $\gamma \in \mathit{Const}(\Delta)^{\otimes}$ such that $\gamma \approx r$, $s \| r \approx f$ there is a derivation $s \to^* \overline{\gamma} f$ in G_g for every $\overline{\gamma} \in \mathit{Lin}(\gamma)$. By induction on $\mathit{length}(\gamma)$.

- $\gamma = \varepsilon$. As $\varepsilon \approx r$, the pair (ε, r) belongs to $\mathcal{B}$. Hence $s \to f$ by definition of G_g.
- Let $\mathit{length}(\gamma) = i + 1$ and let $X\overline{\beta} \in \mathit{Lin}(\gamma)$. Then γ is of the form $X \| \beta$ where $\overline{\beta} \in \mathit{Lin}(\beta)$. As $X \| \beta \approx r$ and Δ is normed, there are $u, v \in \mathit{Const}(\Gamma)$ such that $X \approx u$, $\beta \approx v$, and $u \| v \approx r$. Hence we also have $s \| u \| v \approx f$, thus $s \| u \approx t$ for some $t \in \mathit{Const}(\Gamma)$. As $X \approx u$, the pair (X, u) belongs to $\mathcal{B}$.

Clearly $s \to Xt$ by definition of G_g. As $\beta \approx v$ and $v\|t \approx f$, we can use the induction hypothesis and conclude $t \to^* \overline{\beta} f$. Hence $s \to^* X\overline{\beta} f$ as required.

Now let $\alpha \approx g$. As Δ is normed, there is some $r \in \mathit{Const}(\Gamma)$ such that $\varepsilon \approx r$. Hence $(\varepsilon, r) \in \mathcal{B}$ and $S \to r$ by definition of G_g. Clearly $r\|g \approx g$ and due to the above proved property we have $r \to^* \overline{\alpha} g$ for every $\overline{\alpha} \in \mathit{Lin}(\alpha)$. As $g \to \varepsilon$ is a rule of G_g, we obtain $S \to r \to^* \overline{\alpha} g \to \overline{\alpha}$.

The set of states which are reachable from a given $X \in \mathit{Const}(\Delta)$ in one '$\stackrel{a}{\Rightarrow}$' move is no longer regular, but it can, in some sense, be represented by a CF grammar.

Theorem 26 *For all $X \in \mathit{Const}(\Delta)$, $a \in \mathit{Act}(\Delta)$ there is a context-free grammar $G_{(X,a)}$ in 3-GNF (Greibach normal form, i.e., with at most 2 variables at the right hand side of every production) of size $\mathcal{O}(n^4)$ constructible in $\mathcal{O}(n^4)$ time such that the following two conditions hold:*

- *if $G_{(X,a)}$ generates an element of $\mathit{Lin}(\alpha)$, then $X \stackrel{a}{\Rightarrow} \alpha$*
- *if $X \stackrel{a}{\Rightarrow} \alpha$, then $G_{(X,a)}$ generates at least one element of $\mathit{Lin}(\alpha)$*

PROOF. Let $G_{(X,a)} = (N, \Sigma, \delta, X^a)$ where

- $N = \{Y^a, Y^\tau \mid Y \in \mathit{Const}(\Delta)\} \cup \{S\}$
- $\Sigma = \mathit{Const}(\Delta)$
- δ is defined as follows:
  - the rule $S \to X^a$ is added to δ.
  - for each transition $Y \stackrel{a}{\to} Z_1\| \cdots \|Z_k$ of Δ we add the rule
    $$Y^a \to Z_1^\tau \cdots Z_k^\tau$$
    (if $k = 0$, we add the rule $Y^a \to \varepsilon$).
  - for each transition $Y \stackrel{\tau}{\to} Z_1\| \cdots \|Z_k$ of Δ we add the rule
    $$Y^\tau \to Z_1^\tau \cdots Z_k^\tau$$
    (if $k = 0$, we add $Y^\tau \to \varepsilon$). Moreover, if $k \geq 1$ then for each $1 \leq i \leq k$ we also add the rule
    $$Y^a \to Z_1^\tau \cdots Z_i^a \cdots Z_k^\tau$$
  - for each $Y \in \mathit{Const}(\Delta)$ we add the rule
    $$Y^\tau \to Y.$$

The fact that $G_{(X,a)}$ satisfies the above mentioned conditions follows directly from its construction. Note that the size of $G_{(X,a)}$ is $\mathcal{O}(n^2)$ at the moment. Now we transform $G_{(X,a)}$ to 3-GNF by a standard procedure of automata theory (see [19]). It can be done in $\mathcal{O}(n^4)$ time and the size of resulting grammar is $\mathcal{O}(n^4)$.

The notion of expansion is defined in a different way (when compared to the one of the previous section).

Definition 27 *Let K be a well-formed relation. We say that a pair $(X, f) \in K$* expands *in K iff the following two conditions hold:*

- *for each $X \stackrel{a}{\to} \alpha$ there is some $f \stackrel{a}{\Rightarrow} g$ such that $\overline{\alpha} \in L(\mathcal{A}_g)$, where $\overline{\alpha}$ is the canonical form of $Lin(\alpha)$.*
- *for each $f \stackrel{a}{\to} g$ the language $L(\mathcal{A}_g) \cap L(G_{(X,a)})$ is non-empty.*

A pair $(\varepsilon, f) \in K$ expands in K iff $f \stackrel{a}{\to} g$ implies $a = \tau$, and for each $f \stackrel{\tau}{\to} g$ we have that $\varepsilon \in L(\mathcal{A}_g)$. The set of all pairs of K which expand in K is denoted by $Exp(K)$.

Theorem 28 *Let K be a well-formed relation. The set $Exp(K)$ can be computed in $\mathcal{O}(n^{11}\, m^8)$ time.*

PROOF. First we compute the automata $\mathcal{A}_g$ of Theorem 24 for all $g \in Const(\Gamma)$. This takes $\mathcal{O}(n\, m^2)$ time. Then we compute the grammars $G_{(X,a)}$ of Theorem 26 for all $X \in Const(\Delta)$, $a \in Act$. This takes $\mathcal{O}(n^6)$ time. Now we show that it is decidable in $\mathcal{O}(n^{10}\, m^7)$ time whether a pair (X, f) of K expands in K.

The first condition of Definition 27 can be checked in $\mathcal{O}(n^3\, m^2)$ time, as there are $\mathcal{O}(n)$ transitions $X \stackrel{a}{\to} \alpha$, $\mathcal{O}(m)$ states g such that $f \stackrel{a}{\Rightarrow} g$, and for each such pair (α, g) we verify whether $\overline{\alpha} \in L(\mathcal{A}_g)$ where $\overline{\alpha}$ is the canonical form of $Lin(\alpha)$; this membership test can be done in $\mathcal{O}(n^2\, m)$ time, as the size of $\overline{\alpha}$ is $\mathcal{O}(n)$ and the size of $\mathcal{A}_g$ is $\mathcal{O}(n\, m)$.

The second condition of Definition 27 is more expensive. To test the emptiness of $L(\mathcal{A}_g) \cap L(G_{(X,a)})$, we first construct a pushdown automaton $\mathcal{P}$ which recognizes this language. $\mathcal{P}$ has $\mathcal{O}(m)$ control states and its total size is $\mathcal{O}(n^5\, m)$. Furthermore, each rule $pX \stackrel{a}{\to} q\alpha$ of $\mathcal{P}$ has the property that $length(\alpha) \leq 2$, because $G_{(X,a)}$ is in 3-GNF. Now we transform this automaton to an equivalent CF grammar by a well-known procedure described, e.g., in [19]. The size of the resulting grammar is $\mathcal{O}(n^5\, m^3)$, and its emptiness can be thus checked in $\mathcal{O}(n^{10}\, m^6)$ time (cf. [19]). This construction has to be performed $\mathcal{O}(m)$ times, hence we need $\mathcal{O}(n^{10}\, m^7)$ time in total.

Pairs of the form (ε, f) are handled in a similar (but less expensive) way. As K contains $\mathcal{O}(n\, m)$ pairs, the computation of $Exp(K)$ takes $\mathcal{O}(n^{11}\, m^8)$ time.

The previous theorem is actually a straightforward consequence of Definition 27. The next theorem says that *Exp* really does what we need.

Theorem 29 *Let K be a well-formed relation such that $Exp(K) = K$. Then $Cl(K)$ is a weak bisimulation.*

PROOF. Let $(\alpha, f) \in Cl(K)^i$. We prove that for each $\alpha \stackrel{a}{\rightarrow} \beta$ there is some $f \stackrel{a}{\Rightarrow} g$ such that $(\beta, g) \in Cl(K)$ and vice versa. By induction on i.

- $i = 0$. Then $(\alpha, f) \in K$, and we can distinguish the following two possibilities:

(1) $\alpha = X$

Let $X \stackrel{a}{\rightarrow} \beta$. By Definition 27 there is $f \stackrel{a}{\Rightarrow} g$ such that $\overline{\beta} \in L(\mathcal{A}_g)$ for some $\overline{\beta} \in Lin(\beta)$. Hence $(\beta, g) \in Cl(K)$ due to the first part of Theorem 24.

Let $f \stackrel{a}{\rightarrow} g$. By Definition 27 there is some string $w \in L(\mathcal{A}_g) \cap L(G_{(X,a)})$. Let $w \in Lin(\beta)$. We have $X \stackrel{a}{\Rightarrow} \beta$ due to the first part of Theorem 26, and $(\beta, g) \in Cl(K)$ due to Theorem 24.

(2) $\alpha = \varepsilon$

Let $f \stackrel{a}{\rightarrow} g$. Then $a = \tau$ and $\varepsilon \in L(\mathcal{A}_g)$ by Definition 27. Hence $(\varepsilon, g) \in Cl(K)$ due to Theorem 24.

- **Induction step.** Let $(\alpha, f) \in Cl(K)^{i+1}$. There are two possibilities.

I. $\alpha = X \| \gamma$ and there are r, s such that $(X, r) \in K$, $(\gamma, s) \in Cl(K)^i$, and $r \| s \approx f$.

Let $X \| \alpha \stackrel{a}{\rightarrow} \beta$. The action '$a$' can be emitted either by X or by α. We distinguish the two cases.

1) $X \| \gamma \stackrel{a}{\rightarrow} \delta \| \gamma$. As $(X, r) \in K$ and $X \stackrel{a}{\rightarrow} \delta$, there is some $r \stackrel{a}{\Rightarrow} r'$ such that $(\delta, r') \in Cl(K)$. As $r \| s \approx f$ and $r \stackrel{a}{\Rightarrow} r'$, there is some $f \stackrel{a}{\Rightarrow} g$ such that $r' \| s \approx g$. To sum up, we have $(\delta, r') \in Cl(K)$, $(\gamma, s) \in Cl(K)$, $r' \| s \approx g$, hence $(\delta \| \gamma, g) \in Cl(K)$ due to Lemma 22.

2) $X \| \gamma \stackrel{a}{\rightarrow} X \| \rho$. As $(\gamma, s) \in Cl(K)^i$ and $\gamma \stackrel{a}{\rightarrow} \rho$, there is $s \stackrel{a}{\Rightarrow} s'$ such that $(\rho, s') \in Cl(K)$. As $r \| s \approx f$ and $s \stackrel{a}{\Rightarrow} s'$, there is $f \stackrel{a}{\Rightarrow} g$ such that $(r \| s') \approx g$. Due to Lemma 22 we obtain $(X \| \rho, g) \in Cl(K)$.

Let $f \stackrel{a}{\rightarrow} g$. As $r \| s \approx f$, there are $r \stackrel{x}{\Rightarrow} r'$, $s \stackrel{y}{\Rightarrow} s'$ where $x = a \wedge y = \tau$ or $x = \tau \wedge y = a$ such that $r' \| s' \approx g$. As $(X, r) \in K$, $(\gamma, s) \in Cl(K)^i$, there are $X \stackrel{x}{\Rightarrow} \delta$, $\gamma \stackrel{y}{\Rightarrow} \rho$ such that (δ, r'), $(\rho, s') \in Cl(K)$. Clearly $X \| \gamma \stackrel{a}{\Rightarrow} \delta \| \rho$ and $(\delta \| \rho, g) \in Cl(K)$ due to Lemma 22.

II. $(\alpha, r) \in Cl(K)^i$ and there is some s such that $(\varepsilon, s) \in K$ and $r \| s \approx f$.

The proof can be completed along the same lines as above.

Now we can approximate (and compute) the bisimulation base in the same way as in the Section 3.

Theorem 30 *There is a $j \in \mathbb{N}$, bounded by $\mathcal{O}(n\,m)$, such that $\mathcal{B}^j = \mathcal{B}^{j+1}$. Moreover, $\mathcal{B}^j = \mathcal{B}$.*

PROOF. '$\supseteq$:' It suffices to show that $Exp(\mathcal{B}) = \mathcal{B}$. Let $(\alpha, f) \in \mathcal{B}$. Then $\alpha \approx f$, and $\alpha = X$ for some $X \in Const(\Delta)$ or $\alpha = \varepsilon$. We show that (X, f) expands in $\mathcal{B}$ (a proof for the pair (ε, f) is similar).

Let $X \stackrel{a}{\rightarrow} \beta$. As $X \approx f$, there is $f \stackrel{a}{\Rightarrow} g$ such that $\beta \approx g$. Let $\overline{\beta}$ be the canonical form of $Lin(\beta)$. Due to Lemma 25 we have $\overline{\beta} \in L(\mathcal{A}_g)$.

Let $f \stackrel{a}{\rightarrow} g$. As $X \approx f$, there is $X \stackrel{a}{\Rightarrow} \beta$ such that $\beta \approx g$. Due to Theorem 26 there is $\overline{\beta} \in Lin(\beta)$ such that $\overline{\beta} \in L(G_{(X,a)})$. Moreover, $\overline{\beta} \in L(\mathcal{A}_g)$ due to Lemma 25. Hence, $L(\mathcal{A}_g) \cap L(G_{(X,a)})$ is nonempty.

'$\subseteq$:' It follows directly from Theorem 29.

Theorem 31 *Weak bisimilarity between normed BPP and finite-state processes is decidable in $\mathcal{O}(n^{12}\, m^9)$ time.*

PROOF. By Theorem 30 the computation of the expansion of Theorem 28 (which costs $\mathcal{O}(n^{11}\, m^8)$ time) has to be done $\mathcal{O}(n\, m)$ times.

6 Conclusions

We have proved that weak bisimilarity is decidable between BPA processes and finite-state processes in $\mathcal{O}(n^5\, m^7)$ time, and between normed BPP and finite-state processes in $\mathcal{O}(n^{12}\, m^9)$ time. It may be possible to improve the algorithm by re-using previously computed information, for example about sets of reachable states, but the exponents would still be very high. This is because the whole bisimulation basis is constructed. To get a more efficient algorithm, one could try to avoid this. Note however, that once we have constructed $\mathcal{B}$ (for a BPA/nBPP system Δ and a finite-state system Γ) and the automaton $\mathcal{A}_g$ of Theorem 6/Theorem 24 (for $K = \mathcal{B}$ and some $g \in Const(\Gamma)$), we can decide weak bisimilarity between a BPA/nBPP process α over Δ and a process $f \in Const(\Gamma)$ in time $\mathcal{O}(|\alpha|)$—it suffices to test whether $\mathcal{A}_f$ accepts α (observe that there is no substantial difference between $\mathcal{A}_f$ and $\mathcal{A}_g$ except for the initial state).

The technique of bisimulation bases has also been used for strong bisimilarity in [17,18]. However, those bases are different from ours; their design and the way how they generate 'new' bisimilar pairs of processes rely on additional algebraic properties of strong bisimilarity (which is a full congruence w.r.t. sequencing, allows for unique decompositions of normed processes w.r.t. sequencing and parallelism, etc.). The main difficulty of those proofs is to show that the membership in the 'closure' of the defined bases is decidable in polyno-

mial time. The main point of our proofs is the use of 'symbolic' representation of infinite subsets of BPA and BPP state-space.

We would also like to mention that our proofs can be easily adapted to other bisimulation-like equivalences, where the notion of 'bisimulation-like' equivalence is the one of [21]. A concrete example is termination-sensitive bisimilarity of Section 4. Intuitively, almost every bisimulation-like equivalence has the algebraic properties which are needed for the construction of the bisimulation base, and the 'symbolic' technique for state-space representation can also be adapted. See [21] for details.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the Association for Computing Machinery*, 40:653–682, 1993.

[2] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *Proceedings of CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[4] 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, 1995.

[5] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science*, 5, 1997.

[6] D. Caucal. Graphes canoniques des graphes algébriques. *Informatique Théorique et Applications (RAIRO)*, 24(4):339–352, 1990.

[7] I. Černá, M. Křetínský, and A. Kučera. Comparing expressibility of normed BPA and normed BPP processes. *Acta Informatica*, 36(3):233–256, 1999.

[8] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, The University of Edinburgh, 1993.

[9] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 1993.

[10] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.

[11] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Proceedings of FCT'95*, volume 965 of *Lecture Notes in Computer Science*, pages 221–232. Springer, 1995.

[12] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[13] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 1999.

[14] J.F. Groote. A short proof of the decidability of bisimulation for normed BPA processes. *Information Processing Letters*, 42:167–171, 1992.

[15] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. *Electronic Notes in Theoretical Computer Science*, 5, 1996.

[16] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proceedings of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.

[17] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.

[18] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6:251–259, 1996.

[19] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[20] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of LICS'91*, pages 376–386. IEEE Computer Society Press, 1991.

[21] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 1998.

[22] A. Kučera. Effective decomposability of sequential behaviours. *Theoretical Computer Science*, 242(1–2):71–89, 2000.

[23] A. Kučera. Efficient verification algorithms for one-counter processes. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2000.

[24] A. Kučera. On simulation-checking with sequential systems. In *Proceedings of ASIAN 2000*, Lecture Notes in Computer Science. Springer, 2000. To Appear.

[25] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In *Proceedings of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 503–512. Springer, 1999.

[26] R. Mayr. Weak bisimulation and model checking for basic parallel processes. In *Proceedings of FST&TCS'96*, volume 1180 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 1996.

[27] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[28] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 329–341. Springer, 2000.

[29] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proceedings of IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*. Springer, 2000.

[30] R. Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 2000, To appear.

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

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

[33] F. Moller. Infinite results. In *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996.

[34] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[35] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings* 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[36] J.L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, 1981.

[37] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. In *Proceedings of EXPRESS 2000*, *Electronic Notes in Theoretical Computer Science*. 2000.

[38] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[39] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proceedings of FST&TCS 2000*, Lecture Notes in Computer Science. Springer, 2000. To appear.

On the Complexity of Bisimulation Problems for Pushdown Automata

Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France. E-mail: mayr@liafa.jussieu.fr
Phone: +33 1 44 27 28 40, Fax: +33 1 44 27 68 49

Abstract. All bisimulation problems for pushdown automata are at least *PSPACE*-hard. In particular, we show that (1) Weak bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, even for a small fixed finite automaton, (2) Strong bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, but polynomial for every fixed finite automaton, (3) Regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity is *PSPACE*-hard.

Keywords: Pushdown automata, bisimulation, verification, complexity

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [21]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [22] for a survey). While many algorithms for bisimulation problems have a very high complexity, only few lower bounds are known. Jančar [12, 13] showed that strong bisimilarity of two Petri nets [25] and weak bisimilarity of a Petri net and a finite automaton is undecidable. Stříbrná [28] showed that weak bisimilarity for Basic Parallel Processes (BPP) is $\mathcal{NP}$-hard and weak bisimilarity for context-free processes (BPA) is *PSPACE*-hard. (BPA are a proper subclass of pushdown automata.) However, it is still an open question whether these two problems are decidable. So far, the only known lower bound for a decidable bisimulation problem was an *EXPSPACE*-lower bound for strong bisimilarity of Petri nets and finite automata [15], that follows from the hardness of the Petri net reachability problem [18].

For bisimulation problems where one compares an infinite-state system with a finite-state one, much more is known about the decidability and complexity than in the general case of two infinite-state systems [14]. Also the complexity can be much lower. In particular, weak (and strong) bisimilarity of a BPA-process and a finite automaton is decidable in polynomial time [17], while weak bisimilarity of two BPA-processes is *PSPACE*-hard [28].

However, this surprising result does not carry over to general pushdown automata. We show that strong and weak bisimilarity of a pushdown automaton

and a finite automaton is *PSPACE*-hard. (These problems were already known to be in *EXPTIME* [14].) For weak bisimilarity this hardness result holds even for a small fixed finite automaton, while the same problem for strong bisimilarity is polynomial in the size of the pushdown automaton for every fixed finite automaton. These results also yield a *PSPACE* lower bound for strong bisimilarity of two pushdown automata, a problem that has recently been shown to be decidable by Sénizergues [27] (the proof in [27] uses a combination of two semidecision procedures and does not yield any complexity measure).

The problem of bisimilarity is also related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata [26]. See Section 5 for details.

Furthermore, we prove a *PSPACE* lower bound for the problem of regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity.

Thus no bisimulation problem for pushdown automata is polynomial (unless *PSPACE* is $\mathcal{P}$). This shows that there is a great difference between pushdown automata and BPA, although they describe exactly the same class of languages (Chomsky-2).

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E ::= \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '.' is a sequential composition and '$\|$' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '$\|$' are associative, '$\|$' is commutative, and 'ϵ' is a unit for '.' and '$\|$'.

A *process rewrite system* (PRS) [20] is specified by a finite set Δ of *rules* which have the form $E \stackrel{a}{\to} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\to} F) \in \Delta}{E \stackrel{a}{\to} F} \qquad \frac{E \stackrel{a}{\to} E'}{E.F \stackrel{a}{\to} E'.F} \qquad \frac{E \stackrel{a}{\to} E'}{E\|F \stackrel{a}{\to} E'\|F}$$

We extend the notation $E \stackrel{a}{\to} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \stackrel{w}{\to} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. We also use '1' to denote the set of process constants.

The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA (Basic Process Algebra, also called context-free processes), BPP (Basic Parallel Processes), and PA-processes are well-known [1], PDA correspond to pushdown automata (as proved by Caucal in [6]), PN correspond to Petri nets, PRS stands for 'Process Rewrite Systems', PAD and PAN are artificial names made by combining existing ones (PAD = PA+PDA, PAN = PA+PN).

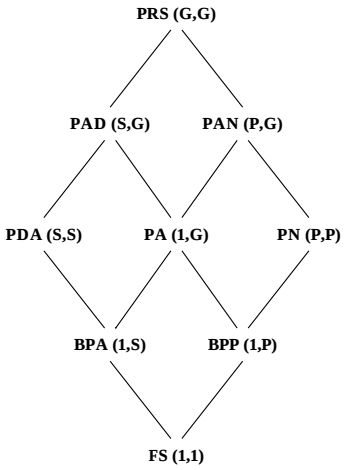


Fig. 1. A hierarchy of PRS

We consider the semantical equivalences *weak bisimilarity* and *strong bisimilarity* [21] over transition systems generated by PRS. In what follows we consider process expressions over $Const(\Delta)$ where Δ is some fixed process rewrite system.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over process expressions is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them.*

Bisimulation equivalence can also be described by *bisimulation games* between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

Note that context-free processes (BPA) correspond to the subclass of pushdown automata (PDA) where the finite control has size 1. Although BPA and PDA describe the same class of languages (Chomsky-2), BPA is strictly less expressive w.r.t. bisimulation.

3 Hardness of Weak Bisimulation Problems

In this section we show lower bounds for problems about weak bisimulation. We consider the following two problems:

Weak bisimilarity of pushdown automata and finite automata

Instance: A pushdown automaton P and a finite automaton F.
Question: $P \approx F$?

Weak Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \approx F$?

We show that both these problems are *PSPACE*-hard. The proof is done by a reduction from the *PSPACE*-complete problem if a single tape, linearly space-bounded, nondeterministic Turing-machine M accepts a given input w. There is a constant k s.t. if M accepts an input w then it has an accepting computation that uses only $k \cdot |w|$ space. For any such M and w we construct a pushdown automaton P s.t.

- If M accepts w then P is not weakly bisimilar to any finite automaton.
- If M doesn't accept w then P is weakly bisimilar to the finite automaton F of Figure 2.

The construction of P is as follows: Let $n := k \cdot |w| + 1$ and Σ be the set of tape symbols of M. Configurations of M are encoded as sequences of n symbols of the form $v_1 q v_2$ where $v_1, v_2 \in \Sigma^*$ are sequences of tape symbols of M and q is a state of the finite control of M. The sequence v_1 are the symbols to the left of the head and v_2 are the symbols under the head and to the right of it. (v_1 can be empty, but v_2 can't.) Let p_0 be the initial control-state of P and let the stack be initially empty.

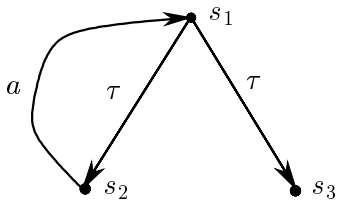


Fig. 2. The finite automaton F with initial state s_1.

Initially, P is in the phase 'guess' where it guesses an arbitrarily long sequence $c_1 \# c_2 \# \ldots \# c_m$ of configurations of M (each of these c_i has length n) and stores them on the stack. The pushdown automaton can guess a sequence of length n by n times guessing a symbol and storing it on the stack. The number of symbols guessed (from 1 to n) is counted in the finite-control of the pushdown automaton. The number m is not counted in the finite-control, since it can be arbitrarily large. The configuration c_m at the bottom of the stack must be accepting (i.e., the state q in c_m must be accepting) and the configuration c_1 at the top must be the initial configuration with the input w and the initial control-state of M. All this is done with silent τ-actions. At the end of this phase P is in the control state p. Then there are two possible transitions: (1) $p \xrightarrow{\tau} p_0 A$ where the special symbol $A \notin \Sigma$ is written on the stack and the guessing phase starts again. (2) $p \xrightarrow{\tau} p_{verify}$ where the pushdown automaton enters the new phase 'verify'.

In the phase 'verify' the pushdown automaton P pops symbols from the stack (by action τ). At any time in this phase it can (but need not) enter the special phase 'check'. For a 'check' it reads three symbols from the stack. These symbols are part of some configuration c_i. Then it pops $n-2$ symbols and then reads the three symbols at the same position in the next configuration c_{i+1} (unless the bottom of the stack is reached already). In a correct computation step from c_i to c_{i+1} the second triple of symbols depends on the first and on the definition of M. If these symbols in the second triple are as they should be in a correct computation step of M from c_i to c_{i+1} then the 'check' is successful and it goes back into the phase 'verify'. Otherwise the 'check' has failed and P is in the control-state *fail*. Here there are two possible transitions: (1) $\mathit{fail} \xrightarrow{\tau} p_2$. In the control-state p_2 the stack is ignored and the pushdown automaton from then on behaves just like the state s_2 in the finite automaton F of Figure 2. (2) $\mathit{fail} \xrightarrow{\tau} p_3$. In the control-state p_3 again the stack is ignored and from then on the pushdown automaton behaves just like the state s_3 in the finite automaton F of Figure 2. The intuition is that if the sequence of configurations represents a correct computation of M then no 'check' can fail, i.e., the control-state *fail* cannot be reached. However, if the sequence isn't a correct computation then there must be at least one error somewhere and thus the control-state *fail* can be reached by doing the 'check' at the right place.
So far, all actions have been silent τ-actions. The only case where a visible action can occur is the following: The pushdown automaton P is in phase 'verify' or 'check' (but not in state *fail*) and reads the special symbol A from the stack. Then it does the visible action 'a' and goes to the control-state p_{verify}. If P reaches the bottom of the stack while being in phase 'verify' or 'check' then it is in a deadlock.

Lemma 2. *If M accepts the input w then P is not weakly bisimilar to any finite automaton.*

Proof. We assume the contrary and derive a contradiction. Assume that there is finite automaton F' with k states s.t. $P \approx F'$. Since M accepts w, there exists an accepting computation sequence $c = c_1 \# c_2 \# \ldots \# c_m$ where all c_i are configurations of M, c_1 is the initial configuration of M with input w, c_m is accepting and for all $i \in \{1, \ldots, m-1\}$ $c_i \to c_{i+1}$ is a correct computation step of M.
P can (by a sequence of τ-steps) reach the configuration $\alpha := p_{verify}\,(cA)^{k+1}c$. Since c is an accepting computation sequence of M, none of the checks can fail. Thus α can only do the following sequence of actions: $\tau^{mn+m-1}(a\tau^{mn+m-1})^{k+1}$. We assumed that $P \approx F'$. Thus there must be some state f of F' s.t. $\alpha \approx f$. Since F' has only k states, it follows from the Pumping Lemma for regular languages that $\alpha \not\approx f$ and we have a contradiction. □

Lemma 3. *Let F be the finite automaton from Figure 2. If M doesn't accept the input w then $P \approx F$.*

Proof. Since there is no accepting computation of M on w, any reachable configuration of P belongs to one of the following three sets.

1. Let C_1 be the set of configurations of P where either P is in phase 'guess' or P is in phase 'verify' or 'check' s.t. a check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* can be reached with only τ-actions.
2. Let C_2 be the set of configurations of P where either the finite control of P is in state p_2 or P is in phase 'verify' or 'check', there is at least one symbol A on the stack and no check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* cannot be reached with only τ-actions, but possibly after another 'a' action.
3. Let C_3 be the set of configurations of P where either the finite control of P is in state p_3 or P is in phase 'verify' or 'check', there is no symbol A on the stack and no check can fail, i.e. the control-state *fail* cannot be reached.

The following relation is a weak bisimulation:

$$\{(\alpha_1, s_1) \mid \alpha_1 \in C_1\} \cup \{(\alpha_2, s_2) \mid \alpha_2 \in C_2\} \cup \{(\alpha_3, s_3) \mid \alpha_3 \in C_3\}$$

We consider all possible attacks.

1. Note that no $\alpha_1 \in C_1$ can do action 'a'.
   - If the attacker makes a move from a configuration in C_1 with control-state *fail* to p_2/p_3 then the defender responds by a move $s_1 \xrightarrow{\tau} s_1/s_2$. These are weakly bisimilar to p_2/p_3 by definition. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1, \alpha_1' \in C_1$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_2$ (this is only possible if there is at least one symbol A on the stack) then the defender responds by making a move $s_1 \xrightarrow{\tau} s_2$. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_3$ (this is only possible if there is no symbol A on the stack) then the defender responds by making a move $s_1 \xrightarrow{\tau} s_3$.
   - If the attacker makes a move $s_1 \xrightarrow{\tau} s_2/s_3$ then the defender makes a sequence of τ-moves where a 'check' fails and goes (via the control-state *fail*) to a configuration with control-state p_2/p_3. This is weakly bisimilar to s_2/s_3 by definition.
2. If α_2 is a configuration with control-state p_2 then this is bisimilar to s_2 by definition.
   - If the attacker makes a move $\alpha_2 \xrightarrow{\tau} \alpha_2'$ with $\alpha_2, \alpha_2' \in C_2$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_2 \xrightarrow{a} \alpha_2'$ (this is only possible if the symbol A is at the top of the stack) then the control-state of α_2' is q_{verify} and $\alpha_2' \in C_1$. Thus the defender can respond by $s_2 \xrightarrow{a} s_1$.
   - If the attacker makes a move $s_2 \xrightarrow{a} s_1$ then the defender responds as follows: First he makes a sequence of τ-moves $\alpha_2 \xrightarrow{\tau^*} \alpha_2'$ that pops symbols from the stack without doing any 'check' until the special symbol A is at the top. Then he makes a move $\alpha_2' \xrightarrow{a} \alpha_2''$. By definition the control-state of α_2'' is q_{verify} and $\alpha_2'' \in C_1$.

3. A configuration $\alpha_3 \in C_3$ can never reach a configuration where it can do action 'a'. The only possible action is τ. Thus $\alpha_3 \approx s_3$.

Since the initial configuration of P is in C_1 and the initial state of F is s_1, we get $P \approx F$. □

Theorem 4. *Weak bisimilarity of pushdown automata and finite automata is PSPACE-hard, even for the fixed finite automaton F of Figure 2.*

Proof. By reduction of the acceptance problem for single tape nondeterministic linear space-bounded Turing machines. Let M, w, P and F be defined as above. If M accepts w then by Lemma 2 P is not weakly bisimilar to any finite automaton and thus $P \not\approx F$. If M doesn't accept w then by Lemma 3 $P \approx F$. □

Theorem 5. *Weak finiteness of pushdown automata is PSPACE-hard.*

Proof. By reduction of the acceptance problem for single tape nondeterministic linear space-bounded Turing machines. Let M, w, P and F be defined as above. If M accepts w then by Lemma 2 P is not weakly bisimilar to any finite automaton and thus not weakly finite. If M doesn't accept w then by Lemma 3 $P \approx F$ and thus P is weakly finite. □

4 Hardness of Strong Bisimulation Problems

STRONG BISIMILARITY OF PUSHDOWN AUTOMATA AND FINITE AUTOMATA
Instance: A pushdown automaton P and a finite automaton F.
Question: $P \sim F$?

We show that this problem is *PSPACE*-hard in general, but polynomial in the size of P for every fixed finite automaton F. The *PSPACE* lower bound is shown by a reduction of the *PSPACE*-complete problem of quantified boolean formulae (QBF). Let $n \in \mathbb{N}$ and let $x_1, \ldots, x_n$ be boolean variables. W.r. we assume that n is even. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. The quantified boolean formula Q is given by

$$Q := \forall x_1 \exists x_2 \ldots \forall x_{n-1} \exists x_n (Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid. We reduce this problem to the bisimulation problem by constructing a pushdown automaton P and a finite automaton F s.t. Q is valid iff $P \sim F$.

F is defined as follows: The initial state is s_0.

$$
\begin{array}{lll}
s_{2i} \xrightarrow{x_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
s_n \xrightarrow{a} u & \\
u \xrightarrow{c} u & \\
t_n \xrightarrow{a} u & \\
t_n \xrightarrow{a} w_n & \\
w_i \xrightarrow{c} w_{i-1} & \text{for } 1 \leq i \leq n
\end{array}
$$

Note that, unlike in the previous section, the size of F is not fixed, but linear in n. Figure 3 illustrates the construction.

Now we define the pushdown automaton P. Initially the stack is empty and the initial control-state is p_0. For $1 \leq j \leq k$ and $1 \leq l \leq n$ we define $Q_j(X_l)$ iff X_l makes the clause Q_j true and $Q_j(\bar{X}_l)$ iff $\bar{X}_l$ makes Q_j true. The transitions of P are as follows:

$$
\begin{array}{lll}
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)} X_{2i+2} X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)} \bar{X}_{2i+2} X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)} X_{2i+2} \bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)} \bar{X}_{2i+2} \bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_n \xrightarrow{a} q_j & \text{for } 0 \leq j \leq k \\
q_0 \xrightarrow{c} q_0 & \\
q_j X_l \xrightarrow{c} q_j X_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(X_l). \\
q_j X_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(X_l). \\
q_j \bar{X}_l \xrightarrow{c} q_j \bar{X}_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(\bar{X}_l). \\
q_j \bar{X}_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(\bar{X}_l).
\end{array}
$$

Additionally we define for $1 \leq i \leq n/2-1$ that in the control-state r_{2i} the stack is ignored and the systems behaves just like t_{2i} in the system F of Figure 3.

Lemma 6. *If Q is not valid then $P \not\sim F$.*

Proof. If Q is not valid then $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n (\neg Q_1 \vee \ldots \vee \neg Q_k)$ and the attacker has the following winning strategy: The attacker chooses the values for the variables with the odd indices by doing actions x_i or $\bar{x}_i$ in the finite automaton F and goes from s_0 to s_n. The defender can respond in two different ways: (1) If the defender goes into a control-state r_{2i} for some i then the attacker can easily

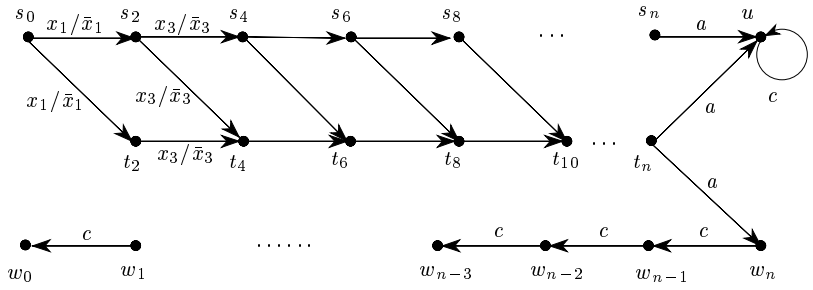


Fig. 3. Reducing QBF to strong bisimulation.

win, since r_{2i} behaves like t_{2i} and $s_{2i} \not\sim t_{2i}$ for every i. (2) If the defender stays in the 'p-domain' of control-states, he is forced to store the attacker's choices for the variables with odd indices on the stack. However, he can make his own choices for the variables with even indices and also stores them on the stack. Finally, the defender reaches the control-state p_n and the stack contains an assignment of values to all n variables. Since Q is not valid, there exists at least one Q_j with $1 \leq j \leq k$ that is not satisfied by this assignment. Now the attacker changes sides and makes the move $p_n \stackrel{a}{\rightarrow} q_j$ in the pushdown automaton P. The defender can only respond by making the move $s_n \stackrel{a}{\rightarrow} u$ in the system F. Now the pushdown automaton P can do the action 'c' only n times, while system F in state u can do it infinitely often. Thus the attacker can win. It follows that $P \not\sim F$. □

Lemma 7. *If Q is valid then $P \sim F$.*

Proof. Let C be a content of the stack and thus a (possibly incomplete) assignment of values to variables. Let $Q_i(C)$ be true iff C makes clause Q_i true. Let $Q(C) := \bigwedge_{1 \leq i \leq k} Q_i(C)$. Let $QX(C)$ be true iff C can be completed to a C' s.t. $Q(C')$. If Q is valid then the following relation is a strong bisimulation.

$$\{(p_{2i}C, s_{2i}) \,|\, 0 \leq i \leq n/2 \wedge QX(C)\} \cup \{(p_{2i}C, t_{2i}) \,|\, 1 \leq i \leq n/2 \wedge \neg QX(C)\} \cup$$
$$\{(r_{2i}C, t_{2i}) \,|\, 1 \leq i \leq n/2\} \cup \{(q_jC, u) \,|\, 1 \leq j \leq k \wedge Q_j(C)\} \cup \{(q_0C, u)\} \cup$$
$$\{(q_jC, w_i) \,|\, 1 \leq j \leq k \wedge 0 \leq i \leq n \wedge \neg Q_j(C) \wedge \mathit{length}(C) = i\}$$

Since $(p_0\epsilon, s_0)$ is in this relation, we get $P \sim F$. □

Theorem 8. *Strong bisimilarity of pushdown automata and finite automata is PSPACE-hard.*

Proof. Directly from Lemma 6 and Lemma 7. □

Corollary 9. *Strong bisimilarity of pushdown automata is PSPACE-hard.*

Note that Theorem 4 is not a corollary of Theorem 8. For weak bisimilarity the hardness result holds even for the small fixed finite automaton of Figure 2. However, strong bisimilarity of a pushdown automaton P and a finite automaton F is polynomial in the size of P for *every* fixed F.

Theorem 10. *Let F be a fixed finite automaton. For every pushdown automaton P the problem if $P \sim F$ requires only polynomial time in the size of P.*

Proof. Using the construction from [14] one can reduce the problem $P \sim F$ to a model checking problem in the temporal logic EF (a fragment of CTL). One can effectively construct Hennessy-Milner Logic formulae Φ and Ψ that depend only on F s.t.

$$P \sim F \iff (P \models \Phi) \wedge (P \models \neg EF\,\Psi)$$

where the modal operator EF denotes reachability. Let n be the size of (the description of) P and m the maximum of the nesting-depth of Φ and Ψ. (The total size of Φ and Ψ can be $\mathcal{O}(2^m)$.) Let P' be a state that is reachable from P. It depends only on the control state of P and P' and on the first m stack symbols of P and P' if they satisfy Φ and Ψ, respectively. There are only n different possibilities for the control state and n^m different possibilities for the first m stack symbols. For each of these n^{m+1} configurations we check if it satisfies Φ or Ψ. Each of those checks can be done in $\mathcal{O}(n^m)$ time. Also for each α of these n^{m+1} configurations we check if P can reach a configuration $\alpha\beta$ for some β. (β represents the stack contents below the first m stack symbols. It does not matter for Φ and Ψ.) Each of those (generalized) reachability-checks can be done in $\mathcal{O}(n^3m^2)$ time [3]. Therefore the whole property above can be checked in $\mathcal{O}(n^{2m+1}m^2)$ time. Thus the problem is polynomial in n, the size of P, but exponential in m. (To be precise, m depends only on F and can be made linear in the number of states in F [14].) □

Now we consider the strong finiteness problem.

Strong Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \sim F$?

We show that this problem is *PSPACE*-hard by a reduction of QBF. Let Q, P and F be defined just as before in the hardness proof of strong bisimilarity. As shown before, Q is valid iff $P \sim F$. We now construct a pushdown automaton P' s.t. P' is finite w.r.t. strong bisimilarity iff $P \sim F$. The initial configuration of P' is $p'Z$. The transition rules are

$$\begin{aligned}
p' &\xrightarrow{a'} p'C \\
p' &\xrightarrow{a'} q' \\
q'C &\xrightarrow{b'} q' \\
q'C &\xrightarrow{c'} p_0 \\
q'Z &\xrightarrow{b'} q'Z \\
q'Z &\xrightarrow{c'} s_0
\end{aligned}$$

Note that if P' is in control-state p_0 or s_0 then it behaves like P and F, respectively.

Lemma 11. *If $P \not\sim F$ then P' is infinite w.r.t. strong bisimilarity.*

Proof. There are infinitely many non-bisimilar reachable states $q'C^iZ$ for all $i \in \mathbb{N}$. It suffices to show that $q'C^iZ \not\sim q'C^jZ$ for $i > j$. The attacker has the following winning strategy: He does action b' exactly j times (the defender can respond in only one way) and the new state in the bisimulation game is $(q'C^{i-j}Z, q'Z)$. Then the attacker does action c' and after the defender's response the new state is $(p_0C^{i-j-1}Z, s_0)$. Since $P \not\sim F$, the attacker can win. □

Lemma 12. *If $P \sim F$ then P' is finite w.r.t. strong bisimilarity.*

Proof. Let the finite automaton F' with initial state s' be defined by

$$s' \xrightarrow{a'} s'$$
$$s' \xrightarrow{a'} t'$$
$$t' \xrightarrow{b'} t'$$
$$t' \xrightarrow{c'} s_0$$

where s_0 is the initial state of F. If $P \sim F$ then $p'C^iZ \sim s'$, $q'C^jZ \sim t'$, $p_0C^kZ \sim s_0$ and $s_0 \sim s_0$ and thus $P' \sim F'$. □

Theorem 13. *Strong finiteness of pushdown automata is PSPACE-hard.*

Proof. It follows from Lemmas 6,7, 11 and 12 that Q is satisfiable iff $P \sim F$ iff P' is finite w.r.t. strong bisimilarity. □

It might seem that Theorem 5 is a corollary of Theorem 13. However, a careful inspection reveals a slight difference. The proof of Theorem 5 shows that the question if, given a pushdown automaton P, "Is P weakly bisimilar to any finite automaton with at most 3 states ?" is *PSPACE*-hard. The same question for strong bisimilarity is polynomial, because of Theorem 10. (These results still hold if the number 3 in the question above is replaced by any other integer $k \geq 3$. For weak bisimilarity the question is *PSPACE*-hard in the size of P. For strong bisimilarity it is polynomial in the size of P and exponential in k.) So, while in general the finiteness problem for a pushdown automaton P is *PSPACE*-hard for both weak and strong bisimilarity, the modified question "Is P finite *and small* ?" is *PSPACE*-hard for weak bisimilarity, but polynomial for strong bisimilarity. To conclude, finiteness w.r.t. weak bisimilarity is hard in a slightly stronger sense.

5 Conclusion

We have shown that all bisimulation problems for pushdown automata are at least *PSPACE*-hard. Thus no bisimulation problem for pushdown automata is polynomial (unless $PSPACE = \mathcal{P}$). It is interesting to compare these results with the results for context-free processes (BPA), which describe exactly the

same class of languages (Chomsky-2). Strong and weak bisimilarity of BPA and finite automata can be decided in polynomial time [17]. This shows that there is a significant difference between pushdown automata and context-free processes (BPA) as far as 'branching-time equivalences' like strong and weak bisimulation are concerned. Intuitively, the reason for this is that, due to their finite control, pushdown automata have a limited power of self-test that context-free processes lack.

The problem of bisimulation equivalence is related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata (dPDA), which has been shown to be decidable in [26]. However, the relationship is more complex than it seems, because of the presence of ϵ-transitions in PDAs. 'Real-time' PDAs are PDAs without ϵ-transitions. We denote them by rPDA. We denote real-time deterministic PDAs as rdPDA. We can distinguish five problems.

1. For rdPDA, strong bisimilarity and trace-language equivalence coincide. (The problem of trace-language equivalence can easily be reduced to terminal-language equivalence on rdPDA.) This problem is also equivalent to strong bisimilarity of dPDA, because the ϵ-transitions don't matter for strong bisimilarity. Language equivalence on rdPDA has been shown to be decidable in [23]. Neither an upper complexity bound nor a lower complexity bound is known.
2. Strong bisimilarity for PDA and rPDA. These problems are equivalent, because the ϵ-transitions don't matter for strong bisimilarity. Decidability of strong bisimilarity for PDA has been shown in [27]. No upper complexity bound is known. Theorem 8 gives a *PSPACE* lower bound.
3. Language equivalence of dPDA. This is equivalent to weak bisimilarity of dPDA, if one renames the ϵ-transitions to τ-transitions. The problem is decidable by [26]. Neither an upper complexity bound nor a lower complexity bound is known.
4. Weak bisimilarity for PDA. It is an open question if this problem is decidable. A *PSPACE* lower bound has been shown in [28] (even for BPA). Theorem 4 shows that even the asymmetric problem of weak bisimilarity of a PDA and a (small fixed) finite automaton is *PSPACE*-hard.
5. Language equivalence for PDA and rPDA. These problems are inter-reducible and undecidable by [11].

Figure 4 shows the relationships between these five problems. The hardness results of this paper hold only for bisimilarity of nondeterministic PDA (i.e., problems number 2 and 4) and thus they don't yield a lower bound for the problem of language equivalence of dPDA (problem number 3). In particular, it is easy to see that language equivalence of a dPDA and a deterministic finite automaton is polynomial (unlike bisimilarity for nondeterministic systems; see Theorem 8). It still cannot be ruled out that a polynomial algorithm for language equivalence of dPDA might exist.

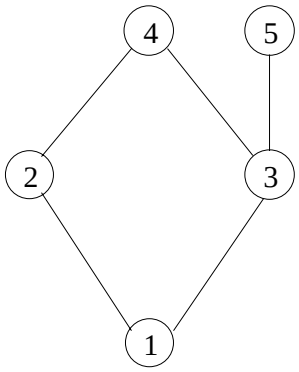


Fig. 4. Bisimulation vs. languages

Two lower bounds for bisimulation problems about Petri nets have not been mentioned explicitly in the literature so far. They concern the problems of strong bisimilarity of a Petri net and a finite automaton and finiteness of a Petri net w.r.t. strong bisimulation. It can easily be shown that these problems are *EXPSPACE*-hard by a reduction of the problem if a given place in a Petri net can ever become marked. (This problem is polynomially equivalent to the reachability problem for Petri nets [25] and thus *EXPSPACE*-hard [18].)

The following table summarizes known results about the complexity of bisimulation problems for several classes of infinite-state systems. The different columns show the results about the following problems: strong bisimilarity with finite automata, strong bisimilarity of two infinite-state systems, weak bisimilarity with finite automata and weak bisimilarity of two infinite-state systems. New results are in boldface.

	$\sim F$	$\sim$	$\approx F$	$\approx$
FS	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]
BPA	$\mathcal{P}$ [17]	$\in 2\text{-}EXPTIME$ [4]	$\mathcal{P}$ [17]	*PSPACE*-hard [28]
PDA	$\in EXPTIME$ [14] **PSPACE-hard**	decidable [27] **PSPACE-hard**	$\in EXPTIME$ [14] **PSPACE-hard**	*PSPACE*-hard [28]
BPP	$\in PSPACE$ [14]	decidable [7] co-$\mathcal{NP}$-hard [19]	$\in PSPACE$ [14]	$\mathcal{NP}$-hard [28] Π_2^p-hard [19]
PA	decidable [14]	co-$\mathcal{NP}$-hard [19]	decidable [14]	*PSPACE*-hard [28]
PAD	decidable [14] **PSPACE-hard**	**PSPACE-hard**	decidable [14] **PSPACE-hard**	*PSPACE*-hard [28]
PN	decidable [15, 14] *EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PAN	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PRS	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]

The following table summarizes results about the problems of strong and weak finiteness. New results are in boldface.

	strong finiteness	weak finiteness
BPA	$\in 2-EXPTIME$ [5, 4]	?
PDA	**PSPACE-hard**	**PSPACE-hard**
BPP	decidable [13] co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PA	co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PAD	**PSPACE-hard**	**PSPACE-hard**
PN	decidable [13] *EXPSPACE*-hard	undecidable [13]
PAN/PRS	*EXPSPACE*-hard	undecidable [13]

Some more results are known about the restricted subclasses of these systems that satisfy the 'normedness condition' (e.g. [10, 9, 8, 16]). Normedness means that from every reachable state there is a terminating computation. This condition makes many bisimulation problems much easier, e.g., strong bisimilarity of normed BPP is decidable in polynomial time [10], while it is at least co-$\mathcal{NP}$-hard in the general case [19]. Also for normed systems finiteness w.r.t. strong bisimilarity coincides with boundedness [16], while this doesn't hold in the general case.

Acknowledgment: Thanks to Colin Stirling for helpful discussions.

References

[1] J.C.M. Baeten and W.P. Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[2] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[4] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[5] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[6] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[7] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[8] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.

[9] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.

[10] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.

[11] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[12] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[13] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.

[14] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[15] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.

[16] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[17] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.

[18] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[19] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP'2000*, volume ? of *LNCS*. Springer Verlag, 2000.

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

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

[22] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[23] M. Oyamaguchi, N. Honda, and Y. Inagaki. The equivalence problem for real-time strict deterministic languages. *Information and Control*, 45:90–115, 1980.

[24] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[25] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[26] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 671–681. Springer Verlag, 1997.

[27] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS'98*. IEEE, 1998.

[28] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.

On the Complexity of Bisimulation Problems for Basic Parallel Processes

Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France. E-mail: mayr@liafa.jussieu.fr

Abstract. Strong bisimilarity of Basic Parallel Processes (BPP) is decidable, but the best known algorithm has non-elementary complexity [7]. On the other hand, no lower bound for the problem was known. We show that strong bisimilarity of BPP is co-$\mathcal{NP}$-hard.
Weak bisimilarity of BPP is not known to be decidable, but an $\mathcal{NP}$ lower bound has been shown in [31]. We improve this result by showing that weak bisimilarity of BPP is Π_2^p-hard.
Finally, we show that the problems if a BPP is regular (i.e., finite) w.r.t. strong and weak bisimilarity are co-$\mathcal{NP}$-hard and Π_2^p-hard, respectively.

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [25]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [26] for a survey). While many algorithms for bisimulation problems have a very high complexity, only few lower bounds are known.

The state of the art. Strong bisimilarity of two Petri nets and weak bisimilarity of a Petri net and a finite automaton is undecidable [13, 14]. Weak bisimilarity for Basic Parallel Processes (BPP) is $\mathcal{NP}$-hard and weak bisimilarity for context-free processes (BPA) is $PSPACE$-hard [31]. However, it is still an open question whether these problems are decidable.

Some lower bounds for decidable bisimulation problems have been shown in [23]. Strong (and weak) bisimilarity between pushdown automata (PDA) and finite automata is $PSPACE$-hard, finiteness of PDA w.r.t. weak and strong bisimilarity also $PSPACE$-hard. Finally, both strong bisimilarity of Petri nets and finite automata and finiteness of Petri nets w.r.t. strong bisimilarity are $EXPSPACE$-hard. (See the table in Section 5 for a summary of all results on the complexity of bisimulation problems.)

Basic Parallel Processes (BPP) were introduced by Christensen [6] as the fragment of CCS [25] without communication, restriction and relabeling. They are equivalent to communication-free nets [8], the subclass of Petri nets [28] where every transition has exactly one input-place with arc-weight one. While strong (and weak) bisimilarity are undecidable for Petri nets [13], strong bisimilarity is decidable for BPP (i.e., communication-free nets) [7]. However, the algorithm in [7] has non-elementary complexity and, to the best of our knowledge, no better algorithm has been found since then. In spite of this, no lower bound for the problem has been found either.

However, there is a polynomial algorithm for bisimilarity on the restricted subclass of *normed* BPP [12]. (A process is *normed* iff from every reachable state there is a terminating computation.) Thus, it was conjectured that a polynomial algorithm should also exist for general (unnormed) BPP. This belief was reinforced by the fact that many other problems for BPP are polynomial: boundedness [17], termination, liveness, (partial) deadlock reachability and (partial) livelock reachability [21, 22]. (On the other

hand there are also hard problems for BPP: reachability is $\mathcal{NP}$-complete [8], some model checking problems are $PSPACE$-complete [20, 22] or even undecidable [9].)

Our contribution. We show that strong bisimilarity for BPP is co-$\mathcal{NP}$-hard (thus proving the above mentioned conjecture wrong). We also show that weak bisimilarity for BPP is Π_2^p-hard, thus improving a previously established $\mathcal{NP}$ lower bound [31]. Finally, we show that the problem if a BPP is regular (i.e., finite) w.r.t. strong and weak bisimilarity is co-$\mathcal{NP}$-hard and Π_2^p-hard, respectively.

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E ::= \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '$.$' is a sequential composition and '$\|$' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '$.$' and '$\|$' are associative, '$\|$' is commutative, and 'ϵ' is a unit for '$.$' and '$\|$'.

A *process rewrite system* (PRS) [24] is specified by a finite set Δ of *rules* which have the form $E \stackrel{a}{\rightarrow} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\rightarrow} F) \in \Delta}{E \stackrel{a}{\rightarrow} F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E.F \stackrel{a}{\rightarrow} E'.F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E \| F \stackrel{a}{\rightarrow} E' \| F}$$

We extend the notation $E \stackrel{a}{\rightarrow} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \stackrel{w}{\rightarrow} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '$.$' operator, respectively. We also use '1' to denote the set of process constants. The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA (Basic Process Algebra, also called context-free processes), BPP (Basic Parallel Processes), and PA-processes are well-known [1],

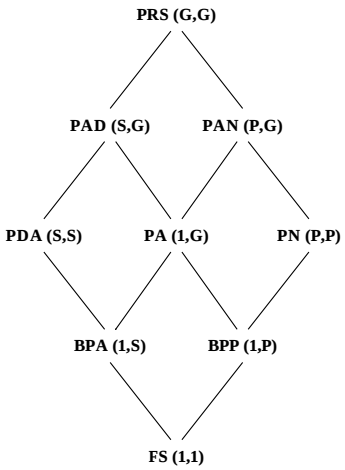


Fig. 1. A hierarchy of PRS

PDA correspond to pushdown automata (as proved by Caucal in [5]), PN correspond to Petri nets, PRS stands for 'Process Rewrite Systems', PAD and PAN are artificial names made by combining existing ones (PAD = PA+PDA, PAN = PA+PN).
Here we study Basic Parallel Processes (BPP) that correspond to process rewrite systems of type $(\mathbf{1}, P)$.
We consider the semantical equivalences *weak bisimilarity* and *strong bisimilarity* [25] over transition systems generated by PRS.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over process expressions is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them. The largest (strong or weak) bisimulation is an equivalence relation.*

Bisimulation equivalence can also be described by *bisimulation games* [30, 32] between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

3 Hardness of Strong Bisimilarity for BPP

STRONG BISIMILARITY OF BPP

Instance: Two BPP processes P_1 and P_2.
Question: $P_1 \sim P_2$?

This problem has been shown to be decidable in [7]. However, the algorithm relies on Dickson's Lemma for termination and therefore the algorithm is not primitive recursive. A polynomial algorithm for bisimilarity on the restricted subclass of *normed* BPP has been described in [12], which led to the conjecture that the general problem was also polynomial. We prove this conjecture wrong by proving a co-$\mathcal{NP}$ lower bound. Thus no polynomial algorithm for strong bisimilarity of BPP can exist, unless $\mathcal{P} = \mathcal{NP}$.
First we give an intuition why the general (unnormed) problem is so hard, using the terminology of communication-free Petri nets. The problem if a place in a communication-free net is unbounded (i.e., if there are reachable states that put arbitrarily high numbers of tokens on it) is easily decidable in polynomial time [17]. However, it is not so easy to determine if the number of tokens on a place really matters w.r.t. bisimilarity, i.e., if states with different numbers of tokens on this place are really different w.r.t. bisimilarity (i.e., non-bisimilar). First we consider the simple example Δ:

$X \xrightarrow{a} X\|Y,\ X \xrightarrow{a} \epsilon,\ Y \xrightarrow{a} \epsilon,\ Z \xrightarrow{a} Z$. The process (X, Δ) is infinite w.r.t. bisimilarity (since it has infinitely many non-bisimilar reachable states). However, $(X\|Z, \Delta)$ is finite w.r.t. bisimilarity, since $(X\|Z, \Delta) \sim (Z, \Delta)$. We say that in the process $(X\|Z, \Delta)$ the subprocess (Z, Δ) *masks* the infiniteness of (X, Δ). In particular, the subprocess Z has the effect that the number of subprocesses Y doesn't matter for bisimilarity, since $(Y^n\|Z, \Delta) \sim (Y^m\|Z, \Delta)$ for any $n, m \in \mathbb{N}$. Now consider the new system $\Delta' := \Delta \cup \{Z \xrightarrow{a} \epsilon\}$. The process $(X\|Z, \Delta')$ is infinite w.r.t. bisimilarity, because $(X\|Z, \Delta') \xrightarrow{a} (X, \Delta')$. We say that by this transition the subprocess X is *unmasked*. Of course, this is only a very trivial example of masking and unmasking. In general the question if a process can be unmasked (i.e., if a place matters w.r.t. bisimilarity) is $\mathcal{NP}$-hard. Later in this section we use a more complex example of masking and unmasking to prove this.

For the subclass of *normed* BPP, finiteness w.r.t. bisimilarity coincides with boundedness. Thus for *normed* BPP finiteness w.r.t. bisimilarity is decomposable into properties of subprocesses and decidable in polynomial time. In particular, for *normed* BPP, the parallel composition of two infinite processes yields an infinite process. For general BPP it is different. The parallel composition of infinite processes (w.r.t. bisimilarity) can yield a process that is finite w.r.t. bisimilarity. Thus, finiteness (or infiniteness) w.r.t. bisimilarity of a BPP process cannot be decomposed into properties of subprocesses in general. The following example shows this. Let Δ be

$$X_i \xrightarrow{a_{i+1}} X_i\|Y_i, \quad Y_i \xrightarrow{a_i} \epsilon \text{ for } 1 \leq i \leq n-1$$
$$X_n \xrightarrow{a_1} X_n\|Y_n,\ Y_n \xrightarrow{a_n} \epsilon$$

Then the process $X_1\|X_2\|\ldots\|X_n$ is finite w.r.t. bisimilarity, but every subprocess (e.g. $X_3\|X_4\|X_7$) is infinite w.r.t. bisimilarity.

Now we are ready to prove the co-$\mathcal{NP}$ lower bound for strong bisimilarity of BPP. We do this by a polynomial reduction of 3-SAT to the negation of the problem. Let $n \in \mathbb{N}$ and let $x_1, \ldots, x_n$ be boolean variables. A literal is either a variable or the negation of a variable. A clause is a disjunction of 3 literals. Let $Q := Q_1 \wedge \ldots \wedge Q_k$ be a boolean formula in 3-CNF over $x_1, \ldots, x_n$ with k clauses. We construct BPPs P_1 and P_2 s.t. Q is satisfiable iff $P_1 \not\sim P_2$. The set of transition rules Δ is defined as follows.

For every $i \in \{1, \ldots, n\}$ we have $X_i \xrightarrow{x_i} X_{i+1}\|\alpha_i$ where α_i is a parallel composition of constants defined as follows: For every $j \in \{1, \ldots, k\}$ let A_j be in α_i iff the first literal of Q_j is $\bar{x}_i$. For every $j \in \{1, \ldots, k\}$ let B_j be in α_i iff the second literal of Q_j is $\bar{x}_i$. For every $j \in \{1, \ldots, k\}$ let C_j be in α_i iff the third literal of Q_j is $\bar{x}_i$. For every $i \in \{1, \ldots, n\}$ we have $X_i \xrightarrow{\bar{x}_i} X_{i+1}\|\beta_i$ where β_i is a parallel composition of constants defined as follows: For every $j \in \{1, \ldots, k\}$ let A_j be in β_i iff the first literal of Q_j is x_i. For every $j \in \{1, \ldots, k\}$ let B_j be in β_i iff the second literal of Q_j is x_i. For every $j \in \{1, \ldots, k\}$ let C_j be in β_i iff the third literal of Q_j is x_i. The intuition is that by action $x_i/\bar{x}_i$ one chooses the value $true/false$ for the variable x_i. Q is satisfiable iff the assignment of values to the variables can be chosen in such a way that for every $j \in \{1, \ldots, k\}$ at least one of the constants $\{A_j, B_j, C_j\}$ does *not* appear.

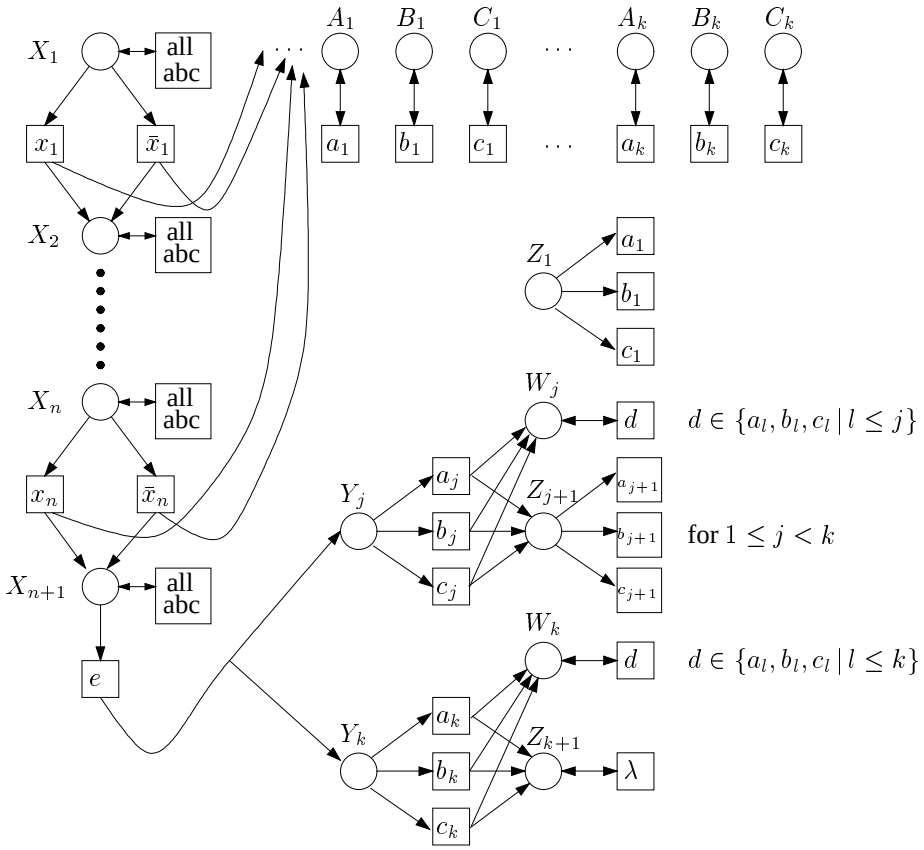


Fig. 2. Δ in Petri net notation. 'All abc' means all actions a_i, b_i, c_i for every $i \in \{1, \ldots, k\}$.

The other transition rules are as follows:

A_j	$\xrightarrow{a_j} A_j$	for $1 \le j \le k$
B_j	$\xrightarrow{b_j} B_j$	for $1 \le j \le k$
C_j	$\xrightarrow{c_j} C_j$	for $1 \le j \le k$
X_i	$\xrightarrow{d_j} X_i$	for $1 \le i \le n+1, 1 \le j \le k, d_j \in \{a_j, b_j, c_j\}$
X_{n+1}	$\xrightarrow{e} Y_1 \| \ldots \| Y_k$	
Y_j	$\xrightarrow{d_j} Z_{j+1} \| W_j$	for $1 \le j \le k, d_j \in \{a_j, b_j, c_j\}$
W_j	$\xrightarrow{d_l} W_j$	for $1 \le j \le k, 1 \le l \le j, d_l \in \{a_l, b_l, c_l\}$
Z_j	$\xrightarrow{d_j} \epsilon$	for $1 \le j \le k, d_j \in \{a_j, b_j, c_j\}$
Z_{k+1}	$\xrightarrow{\lambda} Z_{k+1}$	

Figure 2 gives a rough description of Δ in Petri net notation. Let $P_1 := (X_1 \| Z_1, \Delta)$ and $P_2 := (X_1, \Delta)$.

Lemma 2. *If Q is satisfiable then $P_1 \not\sim P_2$.*

Proof. We show that the attacker has the following winning strategy. Since Q is satisfiable, there exists an assignment of variables that makes Q true. The attacker can choose this assignment by performing the corresponding actions x_i or $\bar{x}_i$ for $1 \leq i \leq n$ in either P_1 or P_2. Then the attacker does the action e. The defender can only respond by doing exactly the same. This yields the new states P_1' and P_2' with $P_1' = P_2' \| Z_1$. For every $j \in \{1, \ldots, k\}$ there is at least one constant $D_j \in \{A_j, B_j, C_j\}$ that does not appear in P_1' or P_2'. Let d_j be the action corresponding to D_j, e.g. if $D_1 = B_1$ then $d_1 = b_1$.

The attacker performs the action d_1 by the rule $Z_1 \xrightarrow{d_1} \epsilon$ in P_1'. Since neither D_1 nor Z_1 occurs in P_2' the defender can only respond by $Y_1 \xrightarrow{d_1} Z_2 \| W_1$. Let the resulting states be P_1'' and P_2''. Now the attacker performs $Z_2 \xrightarrow{d_2} \epsilon$ in P_2'' to which the defender can only respond by $Y_2 \xrightarrow{d_2} Z_3 \| W_2$ in P_1'' and so on with Z_j, d_j for $1 \leq j \leq k$. In the end the defender is forced to perform the transition $Y_k \xrightarrow{d_k} Z_{k+1} \| W_k$. Now the action λ is enabled in one process (by the constant Z_{k+1}), but not in the other. Thus the attacker can win and $P_1 \not\sim P_2$. □

Lemma 3. *If Q is not satisfiable then $P_1 \sim P_2$.*

Proof. Let AS (for 'assignments') be the set of subterms containing only constants A_j, B_j, C_j for $1 \leq j \leq k$. We call a term $t \in AS$ a *faulty assignment* iff there is at least one $m \in \{1, \ldots, k\}$ s.t. all three constants A_m, B_m, C_m occur in t. We call the minimal such m the *index* of t, denoted $ind(t)$. Let FAS be the set of faulty assignments. Since Q is not satisfiable, every assignment t that is created by performing one of each pair of actions $x_1/\bar{x}_1 \ldots x_n/\bar{x}_n$ is a faulty assignment. Any incomplete assignment t' that is created by an incomplete prefix of choices from $x_1/\bar{x}_1 \ldots x_j/\bar{x}_j$ (with $j < n$) must in the end become a faulty assignment once all choices from $x_1/\bar{x}_1 \ldots x_n/\bar{x}_n$ have been made. Let $IFAS_j$ be the set of these incomplete faulty assignments created by choosing one of each pair of actions $x_1/\bar{x}_1 \ldots x_j/\bar{x}_j$. Let O_j be the set of terms containing only constants Y_l, W_l, Z_l, Z_{l+1} with $l \leq j$. To keep the notation simple we define $W_0 := \epsilon$. The symmetric closure of the following relation is a bisimulation.

$$
\begin{array}{l}
\{(X_i \| t \| Z_1^u,\ X_i \| t \| Z_1^v) \mid 1 \leq i \leq n \wedge t \in IFAS_{i-1} \wedge u, v \in \mathbb{N}_0\} \cup \\
\{(X_{n+1} \| t \| Z_1^u,\ X_{n+1} \| t \| Z_1^v) \mid t \in FAS \wedge u, v \in \mathbb{N}_0\} \cup \\
\{(Y_1 \| \ldots \| Y_k \| t \| Z_1^u,\ Y_1 \| \ldots Y_k \| t \| Z_1^v) \mid t \in FAS \wedge u, v \in \mathbb{N}_0\} \cup \\
\{(Y_j \| \ldots \| Y_k \| W_{j-1} \| t \| \gamma,\ Y_{j+1} \| \ldots \| Y_k \| t \| Z_{j+1} \| W_j \| \gamma') \mid \\
t \in FAS \wedge j+1 \leq ind(t) \wedge \gamma, \gamma' \in O_{j-1}\} \cup \\
\{(Y_j \| \ldots \| Y_k \| t \| W_{j-1} \| \gamma,\ Y_{j+1} \| \ldots \| Y_k \| t \| W_j \| \gamma') \mid \\
t \in FAS \wedge j+1 \leq ind(t) \wedge \gamma, \gamma' \in O_{j-1}\} \cup \\
\{(Y_{j+1} \| \ldots \| Y_k \| t \| Z_{j+1} \| W_j \| \gamma,\ Y_{j+1} \| \ldots \| Y_k \| t \| W_j \| \gamma') \mid \\
t \in FAS \wedge j+1 \leq ind(t) \wedge \gamma, \gamma' \in O_{j-1}\} \cup \\
\{(W_j \| Z_{j+1}^u \| Y_{j+1} \| \ldots \| Y_k \| \gamma \| t,\ W_j \| Z_{j+1}^u \| Y_{j+1} \| \ldots \| Y_k \| \gamma' \| t) \mid \\
u \in \{0,1\} \wedge t \in FAS \wedge 1 \leq j \leq k \wedge \gamma, \gamma' \in O_{j-1}\}
\end{array}
$$

Since $(X_1 \| Z_1, X_1)$ is in this relation, we get $P_1 \sim P_2$. □

Theorem 4. *Strong bisimilarity of BPP is co-$\mathcal{NP}$-hard.*

Proof. Directly from Lemma 2 and Lemma 3 and the $\mathcal{NP}$-completeness of 3-SAT.

Note that both P_1 and P_2 are bounded, i.e., they have only finitely many reachable states. It is easy to see that in general the number of reachable states of P_1/P_2 is exponential in the size of the description of Δ. Moreover, the number of reachable states of P_1/P_2 is even exponential up to strong bisimilarity, i.e., they generally have an exponential number of non-bisimilar reachable states. Let $t \in FAS$. Analogously to the definition of the index of t we define $ind'(t)$ as the *maximal* m s.t. all three A_m, B_m, C_m appear in t. Consider the reachable states $X_{n+1}\|t_1\|t_2$, where $t_1\|t_2 \in FAS$ encodes a faulty assignment and the constants A_j, B_j, C_j in t_1 have $j \leq ind'(t_1\|t_2)$ and the constants A_j, B_j, C_j in t_2 have $j > ind'(t_1\|t_2)$. In particular $ind'(t_1\|t_2) = ind'(t_1)$. While the particular structure of t_1 does not matter for bisimilarity (as long as $t_1 \in FAS$), the structure of t_2 does. We have $X_{n+1}\|t_1\|t_2 \not\sim X_{n+1}\|t_1\|t_2'$ for every $t_2' \neq t_2$. Since there are in general exponentially many different such t_2' it follows that P_1/P_2 is at least exponential w.r.t. strong bisimilarity. Thus, our construction does not yield a lower bound for the problem of strong bisimilarity of a BPP and a finite-state process (with polynomially many states). It seems to be impossible to prove a lower bound for this asymmetric problem, since whenever one encodes a sufficiently complex problem (e.g. SAT) into a BPP, this BPP is never bisimilar (neither strongly nor weakly) to any finite-state system of polynomial size (although it can be bisimilar to a finite-state system of exponential size). Thus, we conjecture that strong and weak bisimilarity of a BPP and a finite-state system is decidable in polynomial time. (Is is known that strong and weak bisimilarity of a *normed* BPP and a finite-state system is polynomial [18]).

Now we consider the strong finiteness problem.

STRONG FINITENESS OF BPP

Instance: A BPP process P.
Question: Does there exist a finite-state system F s.t. $P \sim F$?

Finiteness w.r.t. strong bisimilarity is decidable even for general Petri nets [14], and this result carries over immediately to communication-free nets (i.e., BPP). However, the algorithm in [14] consists of two semidecision procedures and gives no upper bound on the complexity. For general Petri nets one gets an $EXPSPACE$ lower bound by reducing the problem if a given place can ever become marked to the finiteness problem. For general Petri nets the problem if a given place can ever become marked is $EXPSPACE$-hard [19, 28]. For communication-free nets (i.e., BPP) this is different. While the reachability problem is $\mathcal{NP}$-complete for communication-free nets [8], it is easy to see that the problem if a given place can ever become marked in a communication-free net is polynomial. Thus, one does not obtain a lower bound for the strong finiteness problem of BPP that way.

It is clear that a *constructive* solution to the problem, i.e., constructing the finite-state system F if it exists, must require at least exponential time. This is because there are BPPs s.t. the smallest finite-state system F that is bisimilar to them has an exponential number of states (in the size of the description of the BPP). However, it is not immediately clear if a simple yes/no answer to the strong finiteness problem must be as hard. The following theorem shows this.

Theorem 5. *Strong finiteness of BPP is co-$\mathcal{NP}$-hard.*

Proof. By a polynomial reduction of 3-SAT to strong infiniteness. Let the formula Q and the set of rules Δ be defined as before and let $\Delta' := \Delta \cup \{X_{n+1} \xrightarrow{p} X_{n+1} \| Z_1\}$. We show that the process X_1 w.r.t. the set of rules Δ', denoted (X_1, Δ'), is infinite w.r.t. strong bisimilarity iff Q is satisfiable.

- $\Leftarrow$ If Q is satisfiable then there are infinitely many reachable states $Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^m$ for every $m \in \mathbb{N}_0$, where γ is a term that encodes a satisfying assignment of Q. This means that γ is a parallel composition of constants A_j, B_j, C_j where for every $j \in \{1, \ldots, k\}$ at least one of the constants A_j, B_j, C_j does not occur in γ. However, for every $m_1 \neq m_2$ we have $Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^{m_1} \not\sim Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^{m_2}$, because the attacker has a winning strategy similar to the one in Lemma 2. Thus (X_1, Δ') is infinite w.r.t. strong bisimilarity.
- $\Rightarrow$ Let $\Delta'' := \Delta \cup \{X_{n+1} \xrightarrow{p} X_{n+1}\}$. The process (X_1, Δ'') has finitely many reachable states. (However, (X_1, Δ'') has an exponential (in the size of Δ'') number of non-bisimilar reachable states.) If Q is not satisfiable then $(X_1, \Delta') \sim (X_1, \Delta'')$ and is thus finite w.r.t. bisimilarity. The bisimulation relation is the same as in Lemma 3. □

The previous construction shows that the problem if a place can be unmasked (i.e., made to count w.r.t. bisimulation) is $\mathcal{NP}$-hard. Here this particular place was Z_1.

4 Hardness of Weak Bisimilarity for BPP

WEAK BISIMILARITY OF BPP

Instance: Two BPP processes P_1 and P_2.
Question: $P_1 \approx P_2$?

It is still an open question if this problem is decidable. It has been shown to be semidecidable in [8], using the facts that weak bisimulation equivalence on BPPs is semilinear (since it is a congruence on a finitely generated commutative semigroup) and that it is decidable if a given semilinear relation on a BPP is a weak bisimulation. An $\mathcal{NP}$ lower bound for this problem has been shown in [31] (by reduction of a variant of the bin-packing problem), and the co-$\mathcal{NP}$ lower bound of Theorem 4 carries over immediately to weak bisimilarity. Here we prove a Π_2^p-lower bound (in the polynomial hierarchy) that subsumes these results.

Let $Q := Q_1 \wedge \ldots \wedge Q_k$ be a boolean formula in 3-CNF over the boolean variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ with k clauses. We construct BPP processes P_1, P_2 s.t. $P_1 \approx P_2$ iff $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$. Since this problem is Π_2^p-complete, we get a Π_2^p-lower bound for the problem of weak bisimilarity.

Let α_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in α_i iff x_i makes clause Q_j true (i.e., x_i appears positively in Q_j). Let β_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in β_i iff $\bar{x}_i$ makes clause Q_j true (i.e., x_i appears negatively in Q_j). Let γ_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in γ_i iff y_i makes clause Q_j true. Let δ_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears

in δ_i iff $\bar{y}_i$ makes clause Q_j true. The set of transition rules Δ is defined by

$$
\begin{array}{lll}
X_i & \xrightarrow{x_i} X_{i+1} \| \alpha_i & \text{for } 1 \leq i \leq n \\
X_i & \xrightarrow{\bar{x}_i} X_{i+1} \| \beta_i & \text{for } 1 \leq i \leq n \\
X'_i & \xrightarrow{x_i} X'_{i+1} \| \alpha_i & \text{for } 1 \leq i \leq n \\
X'_i & \xrightarrow{\bar{x}_i} X'_{i+1} \| \beta_i & \text{for } 1 \leq i \leq n \\
X_{n+1} & \xrightarrow{a} Y_1 \| \ldots \| Y_n & \\
X'_{n+1} & \xrightarrow{a} Y_1 \| \ldots \| Y_n & \\
X'_{n+1} & \xrightarrow{a} Z & \\
Y_i & \xrightarrow{\tau} \gamma_i & \text{for } 1 \leq i \leq n \\
Y_i & \xrightarrow{\tau} \delta_i & \text{for } 1 \leq i \leq n \\
Q_j & \xrightarrow{q_j} Q_j & \text{for } 1 \leq j \leq k \\
Z & \xrightarrow{q_j} Z & \text{for } 1 \leq j \leq k
\end{array}
$$

Let $P_1 := (X_1, \Delta)$ and $P_2 := (X'_1, \Delta)$.

Lemma 6. *If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ is false then $P_1 \not\approx P_2$.*
Proof. If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ is false then $\exists(x_1, \ldots, x_n)\forall(y_1, \ldots, y_n)\neg Q$. The attacker chooses these values for $x_1, \ldots, x_n$ by choosing $x_i/\bar{x}_i$. The defender can only copy these moves. Then the attacker chooses the transition $X'_{n+1} \xrightarrow{a} Z$. The defender can only respond by $X_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ and then a sequence of silent τ-actions ending in a state t. By definition of Δ and since $\exists(x_1, \ldots, x_n)\forall(y_1, \ldots, y_n)\neg Q$ there will be at least one action q_j (with $1 \leq j \leq k$) that is not enabled by t (and cannot made to be enabled by τ-moves). However, all q_j are enabled by Z. Thus, the attacker has a winning strategy and $P_1 \not\approx P_2$. □

Lemma 7. *If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ then $P_1 \approx P_2$.*
Proof. The attacker can choose the assignment for $x_1, \ldots, x_n$. The defender can only imitate these choices. If the attacker chooses the transition $X_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ or $X'_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ then the defender can respond in such a way that the two processes become equal and the defender wins. If the attacker chooses $X'_{n+1} \xrightarrow{a} Z$ then the defender can (by a long internal move of τ-actions) choose the values for $y_1, \ldots, y_n$ on his side. Since $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ there are choices for $y_1, \ldots, y_n$ s.t. in the resulting state all actions $q_1, \ldots, q_k$ are permanently enabled. Since $q_1, \ldots, q_k$ are also permanently enabled by Z in the other process and all other actions are not, the defender wins. Thus, the defender has a winning strategy and $P_1 \approx P_2$. □

Theorem 8. *Weak bisimilarity of BPP is Π_2^p-hard.*
Proof. Directly from Lemma 6 and Lemma 7.

WEAK FINITENESS OF BPP

Instance: A BPP process P.
Question: Does there exist a finite-state system F s.t. $P \approx F$?

We show that the weak finiteness problem for BPP is also Π_2^p-hard by using the previously defined processes P_1 and P_2 and constructing a new process P that is weakly

finite iff $P_1 \approx P_2$. Let Δ' be $\Delta \cup \Gamma$, where Γ is the following set of transition rules:
$I \xrightarrow{\tau} I\|C \quad I \xrightarrow{\tau} \epsilon \quad C \xrightarrow{c} \epsilon \quad D \xrightarrow{c} E \quad D \xrightarrow{c} E' \quad D \xrightarrow{e} X_1\|S$
$D \xrightarrow{e} X_1'\|S \quad E \xrightarrow{c} E \quad E' \xrightarrow{c} E' \quad E \xrightarrow{e} X_1\|S \quad E' \xrightarrow{e} X_1'\|S \quad S \xrightarrow{c} S$
Let $P := (I\|D, \Delta')$.

Lemma 9. *If $P_1 \not\approx P_2$ then P is not weakly finite.*

Proof. P has infinitely many non-weakly-bisimilar states $D\|C^i$ for all $i \in \mathbb{N}$. It suffices to show that $D\|C^j \not\approx D\|C^i$ for $j > i$. The attacker has the following winning strategy. He does action c exactly $i+1$ times in $D\|C^j$ and reaches the state $D\|C^{j-i-1}$. The defender can respond in different ways in $D\|C^i$, but the reached state will always be either $E\|C^k$ or $E'\|C^k$ for some $k \leq i$. In the first case the attacker does the transition $D \xrightarrow{e} X_1'\|S$. The defender can only respond by $E \xrightarrow{e} X_1\|S$ and the new state in the bisimulation game is $(X_1'\|S\|C^{j-i-1}, X_1\|S\|C^k)$. This is not weakly bisimilar, because $P_1 \not\approx P_2$. The second case is symmetric with X_1 and X_1' exchanged. □

Lemma 10. *If $P_1 \approx P_2$ then P is weakly finite.*

Proof. Let Γ' be Γ where X_1' is replaced by X_1 and $\Delta'' := \Delta \cup \Gamma'$. Since $P_1 \approx P_2$ and weak bisimilarity is a congruence on BPP, we get $P = (I\|D, \Delta') \approx (I\|D, \Delta'')$. It is easy to see that $(I\|D, \Delta'') \approx (E, \Delta'')$, because $S \xrightarrow{c} S$. Thus $P \approx (E, \Delta'')$. However, (E, Δ'') has only finitely many reachable states. □

Theorem 11. *Weak finiteness of BPP is Π_2^p-hard.*

Proof. By Lemmas 6, 7, 9 and 10. □

5 Conclusion

The following table summarizes known results about the complexity of bisimulation problems for several classes of infinite-state systems. New results are in boldface.

	$\sim F$	$\sim$	$\approx F$	$\approx$
FS	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]
BPA	$\mathcal{P}$ [18]	$\in 2\text{-}EXPTIME$ [3]	$\mathcal{P}$ [18]	$PSPACE$-hard [31]
PDA	$\in EXPTIME$ [15] $PSPACE$-hard [23]	decidable [29] $PSPACE$-hard [23]	$\in EXPTIME$ [15] $PSPACE$-hard [23]	$PSPACE$-hard [31]
BPP	$\in PSPACE$ [15]	decidable [7] **co-NP-hard**	$\in PSPACE$ [15]	**Π_2^p-hard**
PA	decidable [15]	**co-NP-hard**	decidable [15]	$PSPACE$-hard [31]
PAD	decidable [15] $PSPACE$-hard [23]	$PSPACE$-hard [23]	decidable [15] $PSPACE$-hard [23]	$PSPACE$-hard [31]
PN	decidable [16, 15] $EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]
PAN	$EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]
PRS	$EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]

The different columns in the table above show the results about the following problems: strong bisimilarity with finite automata, strong bisimilarity of two infinite-state systems, weak bisimilarity with finite automata and weak bisimilarity of two infinite-state systems.
The following table summarizes results about the problems of strong and weak finiteness. New results are in boldface.

	strong finiteness	weak finiteness
BPA	$\in 2-EXPTIME$ [4, 3]	?
PDA	$PSPACE$-hard [23]	$PSPACE$-hard [23]
BPP	decidable [14] **co-NP-hard**	$\mathbf{\Pi_2^P}$**-hard**
PA	**co-NP-hard**	$\mathbf{\Pi_2^P}$**-hard**
PAD	$PSPACE$-hard [23]	$PSPACE$-hard [23]
PN	decidable [14] $EXPSPACE$-hard	undecidable [14]
PAN/PRS	$EXPSPACE$-hard	undecidable [14]

Some more results are known about the restricted subclasses of these systems that satisfy the 'normedness condition' (e.g. [12, 11, 10, 17, 18]).

References

[1] J.C.M. Baeten and W.P. Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[2] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[3] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[4] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[5] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[6] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[7] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[8] J. Esparza. Petri nets, commutative context-free grammars and Basic Parallel Processes. In Horst Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *LNCS*. Springer Verlag, 1995.

[9] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[10] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.
[11] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.
[12] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.
[13] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.
[14] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.
[15] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.
[16] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.
[17] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.
[18] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.
[19] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
[20] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.
[21] R. Mayr. Tableau methods for PA-processes. In D. Galmiche, editor, *Analytic Tableaux and Related Methods (TABLEAUX'97)*, volume 1227 of *LNAI*. Springer Verlag, 1997.
[22] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.
[23] R. Mayr. On the complexity of bisimulation problems for pushdown automata. 2000.
[24] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
[25] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[26] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.
[27] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
[28] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.
[29] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS'98*. IEEE, 1998.
[30] C. Stirling. The joys of bisimulation. In *Proc. of MFCS'98*, volume 1450 of *LNCS*, pages 142–151. Springer Verlag, 1998.
[31] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.
[32] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *Proc. of TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer Verlag, 1993.

Simulation Preorder over Simple Process Algebras

Antonín Kučera[1]
Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic

and

Richard Mayr[2]
Institut für Informatik, TU München, Arcisstr. 21, 80290 München, Germany

E-mail: tony@fi.muni.cz; mayrri@in.tum.de

We consider the problem of simulation preorder/equivalence between infinite-state processes and finite-state ones. First, we describe a general method how to utilize the decidability of bisimulation problems to solve (certain instances of) the corresponding simulation problems. For certain process classes, the method allows to design effective reductions of simulation problems to their bisimulation counterparts and some new decidability results for simulation have already been obtained in this way.

Then we establish the *decidability border* for the problem of simulation preorder/equivalence between infinite-state processes and finite-state ones w.r.t. the hierarchy of process rewrite systems. In particular, we show that simulation preorder (in both directions) and simulation equivalence are decidable in $EXPTIME$ between pushdown processes and finite-state ones. On the other hand, simulation preorder is undecidable between PA and finite-state processes in both directions. These results also hold for those PA and finite-state processes which are deterministic and normed, and thus immediately extend to trace preorder. Regularity (finiteness) w.r.t. simulation and trace equivalence is also shown to be undecidable for PA.

Finally, we prove that simulation preorder (in both directions) and simulation equivalence are *intractable* between all classes of infinite-state systems (in the hierarchy of process rewrite systems) and finite-state ones. This result is obtained by showing that the problem whether a BPA (or BPP) process simulates a finite-state one is $PSPACE$-hard, and the other direction is co-$\mathcal{NP}$-hard; consequently, simulation equivalence between BPA (or BPP) and finite-state processes is also co-$\mathcal{NP}$-hard.

Key Words: concurrency, simulation equivalence, infinite-state systems

1. INTRODUCTION

We study the decidability and computational complexity of checking simulation preorder and equivalence between certain infinite-state systems and finite-state ones. The motivation is that the intended behavior of a process can often be easily specified by a finite-state system, while the actual implementation may contain components which are infinite-state (e.g., counters, buffers, recursive procedures). The task of formal verification is to prove that the specification and the implementation are equivalent.

The same problem has been studied recently for strong and weak bisimilarity [14, 23, 16, 13], and it has been shown that these equivalences are not only *decidable*, but also *tractable* between certain infinite-state processes and finite-state ones. Those issues (namely the complexity ones) are dramatically different from the 'symmetric' case when we compare two infinite-state processes. Here we consider (and answer) analogous questions for simulation, establishing both the decidability and tractability border w.r.t. the hierarchy of process rewrite systems [25] (see Fig. 2).

The state of the art: Simulation preorder/equivalence is known to be undecidable for BPA [9] and BPP [11] processes. An interesting positive result is [1] which shows that simulation preorder (and hence also equivalence) is decidable for one-counter nets, which are 'weak' one-counter automata where the counter cannot be tested for zero explicitly (one-counter nets are computationally equivalent to the subclass of Petri nets with at most one unbounded place). A simpler proof has been given later in [17] where it is also shown that simulation preorder/equivalence for 'general' one-counter automata is already undecidable. Simulation with finite-state systems has been first studied in [16]; in contrast to the 'symmetric' case, simulation preorder between Petri nets and finite-state processes is *decidable* in both directions. Moreover, a related problem of *regularity* (finiteness) of Petri nets w.r.t. simulation equivalence is proved to be undecidable. Recently, it has been shown in [21] that simulation preorder between one-counter nets and finite-state processes is decidable in *polynomial* time in both directions (while, for example, weak bisimilarity between one-counter nets and finite-state processes is still intractable—a DP-hardness results for this problem has been demonstrated in [20]). Moreover, in [21] it is also shown that simulation equivalence between one-counter automata and finite-state processes is already co-$\mathcal{NP}$-hard.

Our contribution: In Section 3 we study the relationship between bisimilarity and simulation equivalence. Our effort is motivated by a general trend that problems for bisimilarity (equivalence, regularity) are often decidable, but the corresponding problems for simulation equivalence are not. We propose a method how to use existing algorithms for 'bisimulation' problems to solve certain instances of the corresponding (and possibly undecidable) 'simulation' ones. Such techniques are interesting from a practical point of view, as only small instances of undecidable problems can be solved in an ad-hoc fashion, and some kind of computer support is necessary for problems of 'real' size. Recently, the

[1] On leave at the Institute for Informatics, Technical University Munich, Germany. Supported by a Research Fellowship granted by the Alexander von Humboldt Foundation and by the Grant Agency of the Czech Republic, grant No. 201/00/0400.

[2] This work was partly supported by DAAD Post-Doc grant D/98/28804.

method has also been used in [15] to reduce certain simulation problems for one-counter nets to the corresponding bisimulation problems for one-counter automata (which had been known to be decidable); some new decidability results have been obtained in this way.

In Section 4 we establish the decidability border of Fig. 2. First we prove that simulation preorder between pushdown processes (PDA) and finite-state ones is *decidable* in $EXPTIME$ in both directions. Consequently, simulation equivalence is also in $EXPTIME$. Then we show that simulation preorder between PA and finite-state processes is *undecidable* in both directions. It is rather interesting that the undecidability results hold even for those PA and finite-state processes which are *deterministic* and *normed*. Simulation *equivalence* between such processes is decidable (it coincides with bisimilarity [14]); however, as soon as we allow just one nondeterministic state in the PA processes, simulation equivalence becomes undecidable. We also show that all the obtained undecidability results can be formulated in a 'stronger' form—it is possible to *fix* a PA or a finite-state process in each of the mentioned undecidable problems. Then we demonstrate that regularity of (normed) PA processes w.r.t. simulation equivalence is also undecidable. Again, it contrasts with regularity w.r.t. bisimilarity for normed PA processes, which is decidable in polynomial time [19]. All of the obtained undecidability results also hold for trace preorder and trace equivalence, and therefore they might be also interesting from a point of view of 'classical' automata theory (see the last section for further comments).

In Section 5 we concentrate on the complexity issues for simulation preorder and equivalence with finite-state processes. We prove that the problem whether a BPA (or BPP) process simulates a finite-state one is $PSPACE$-hard, and the other direction is co-$\mathcal{NP}$-hard. Consequently, simulation equivalence between BPA (or BPP) and finite-state processes is also co-$\mathcal{NP}$-hard. Hence, the main message of this section is that simulation with finite-state systems is *intractable* for all classes of infinite-state systems of the hierarchy shown in Fig. 2. It contrasts sharply with the complexity issues for strong and weak bisimilarity; for example, weak bisimilarity between BPA and finite-state processes, and between normed BPP and finite-state processes is in $\mathcal{P}$ [23].

In the last section we give a summary of existing results in the area of comparing infinite-state systems with finite-state ones and discuss language-theoretic aspects of the obtained results.

2. DEFINITIONS

In concurrency theory, a *process* is typically defined to be a state in a *transition system* (which is a general and widely accepted model of discrete systems).

DEFINITION 2.1. A *transition system* is a triple $T = (S, \mathcal{A}, \rightarrow)$ where S is a set of *states*, $\mathcal{A}$ is a set of *actions*, and $\rightarrow \subseteq S \times \mathcal{A} \times S$ is a *transition relation*.

As usual, we write $s \stackrel{a}{\rightarrow} t$ instead of $(s, a, t) \in \rightarrow$ and we extend this notation in the natural way to elements of $\mathcal{A}^*$. We say that a state t is *reachable* from a state s iff $s \stackrel{w}{\rightarrow} t$ for some $w \in \mathcal{A}^*$. Furthermore, T is said to be *image-finite* iff for all $s \in S$ and $a \in \mathcal{A}$ the set $\{t \mid s \stackrel{a}{\rightarrow} t\}$ is finite; T is *deterministic* if each such set is of size at most 1.

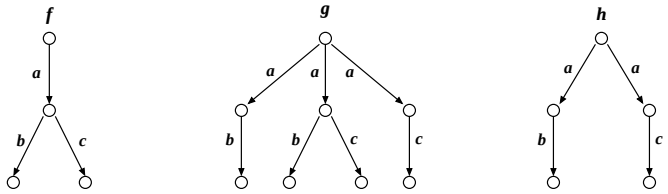


FIG. 1. Processes f, g, and h

2.1. Trace, Simulation, and Bisimulation Equivalence

In this paper we compare infinite-state processes with finite-state ones w.r.t. certain 'levels' of their semantical sameness. Those 'levels' are formally defined as certain preorders and equivalences over the class of all processes (i.e., states in transition systems).

We start with *trace preorder* and *trace equivalence*, which are very similar to the 'classical' notions of language inclusion and language equivalence of automata theory.

DEFINITION 2.2. Let $T = (S, \mathcal{A}, \rightarrow)$ be a transition system. We say that $w \in Act^*$ is a *trace* of a process $s \in S$ iff $s \stackrel{w}{\rightarrow} s'$ for some $s' \in S$. Let $Tr(s)$ be the set of all traces of s. We write $s \sqsubseteq_t t$ iff $Tr(s) \subseteq Tr(t)$. Moreover, we say that s and t are *trace equivalent*, written $s =_t t$, iff $Tr(s) = Tr(t)$.

In concurrency theory, trace equivalence is usually considered as being too coarse. A plethora of finer 'behavioral' equivalences have been proposed (see, e.g., [30] for an overview). *Simulation* and *bisimulation* equivalence are of special importance and their accompanying theory has been developed very intensively.

DEFINITION 2.3. Let $T = (S, \mathcal{A}, \rightarrow)$ be a transition system. A binary relation $R \subseteq S \times S$ is a *simulation* if whenever $(s, t) \in R$ then for each $a \in Act$

$$\text{if } s \stackrel{a}{\rightarrow} s', \text{ then } t \stackrel{a}{\rightarrow} t' \text{ for some } t' \text{ such that } (s', t') \in R$$

A symmetric simulation is called a *bisimulation*. A process s is *simulated* by a process t, written $s \sqsubseteq_s t$, if there is a simulation R such that $(s, t) \in R$. We say that s and t are *simulation equivalent*, written $s =_s t$, iff $s \sqsubseteq_s t$ and $t \sqsubseteq_s s$. Similarly, we say that s and t are *bisimilar* (or *bisimulation equivalent*), written $s \sim t$, iff there is a bisimulation relating them.

It follows immediately from Definition 2.2 and 2.3 that trace equivalence is coarser than simulation equivalence which is coarser than bisimilarity. Moreover, these containments are proper. To see this, consider the processes f, g, h of Fig. 1. Obviously $f =_t g =_t h$. Furthermore, $f =_s g$ but $f \neq_s h \neq_s g$, and $f \not\sim g \not\sim h \not\sim f$.

REMARK 2.1. *All of the introduced equivalences can also be used to relate states of* different *transition systems. Formally, we can consider two transition systems to be a single one by taking their disjoint union.*

Another natural (and studied) problem is the decidability of *regularity* (i.e., 'semantical finiteness') of processes w.r.t. a given behavioral equivalence.

DEFINITION 2.4. A process s is *regular* w.r.t. bisimulation (or simulation, trace) equivalence iff there is a finite-state process f such that $s \sim f$ (or $s =_s f$, $s =_t f$, respectively).

2.2. Process Rewrite Systems

In this paper, we use the syntax of *process rewrite systems* [25] to describe processes. This model is especially suitable for our purposes as it allows to define most of the known (i.e., studied) classes of infinite-state systems in a uniform and succinct way. Similar formalisms for describing processes are used in [3]. However, process rewrite systems have the advantage that they can also describe classes of systems, like PA, that contain both the operators for sequential and parallel composition. A formal definition is as follows: Let $Act = \{a, b, c, \ldots\}$ and $Const = \{X, Y, Z, \ldots\}$ be countably infinite sets of *actions* and *process constants*, respectively. The set of *general process expressions*, denoted G, is defined by the following abstract syntax equation:

$$E \quad ::= \quad \varepsilon \quad | \quad X \quad | \quad E \| E \quad | \quad E.E$$

Here X ranges over $Const$ and ε denotes the empty expression. Intuitively, the '.' operator corresponds to a sequential composition, while the '$\|$' operator models a simple form of parallelism. In the rest of this paper we do not distinguish between expressions related by *structural congruence* which is the smallest congruence relation over process expressions such that the following laws hold:

- associativity for '.' and '$\|$'
- commutativity for '$\|$'
- 'ε' as a unit for '.' and '$\|$'.

DEFINITION 2.5. A *process rewrite system* is a finite set Δ of *rules* which are of the form $E \stackrel{a}{\rightarrow} F$, where $a \in Act$ and $E, F \in G$, $E \neq \varepsilon$ are process expressions. The (finite) sets of process constants and actions which are used in the rules of Δ are denoted by $Const(\Delta)$ and $Act(\Delta)$, respectively.

Each system Δ determines a unique transition system where states are process expressions over $Const(\Delta)$, the set of labels is $Act(\Delta)$, and transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\rightarrow} F) \in \Delta}{E \stackrel{a}{\rightarrow} F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E.F \stackrel{a}{\rightarrow} E'.F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E \| F \stackrel{a}{\rightarrow} E' \| F}$$

All notions and properties of transition systems can be also used for processes of process rewrite systems in the following sense: We say that a process E of Δ has a property p iff the part of the transition system generated by Δ which is reachable from E has the property p. (Observe that, e.g., E can be deterministic even if the transition system generated by Δ is not deterministic.)

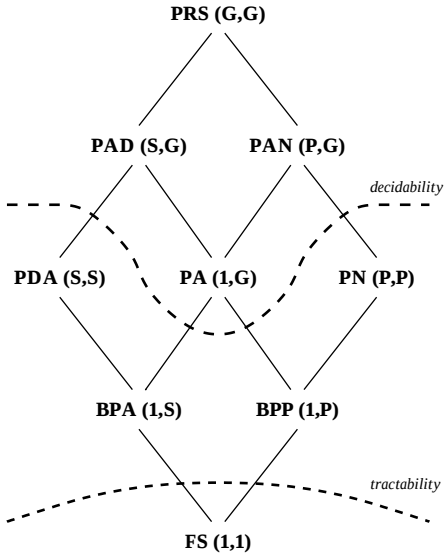


FIG. 2. A hierarchy of process rewrite systems with the decidability/tractability border for simulation with finite-state processes

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of the rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. For short, we also use 1 to denote the set $Const \cup \{\varepsilon\}$. A hierarchy of process rewrite systems is presented in Fig. 2; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively [3]. The set of states of a system Δ which belongs to the subclass determined by (A, B) is then formed by all expressions of B which contain only the constants of $Const(\Delta)$. (In Fig. 2 we also indicated the decidability/tractability border for simulation preorder and equivalence with finite-state systems which is established in the following sections.) This hierarchy contains a variety of widely studied classes of infinite state systems; BPA, BPP, and PA processes are well-known [2], PDA correspond to pushdown processes (as proved by Caucal in [6]), PN correspond to Petri nets (see, e.g., [29]), etc.

It can be shown that the hierarchy of Fig. 2 is *strict* w.r.t. bisimulation semantics [25]; for example, there is a PN process for which there is no bisimilar PAD process, there is a PDA process for which there is no bisimilar BPA or BPP process, etc.

Sometimes we also work with the subclass of *normed* process rewrite systems; a process E of Δ is *normed* if $E \stackrel{w}{\rightarrow} \varepsilon$ for some $w \in Act^*$ (intuitively, this condition means that E can successfully terminate). A system Δ is normed if each of its processes is normed. Observe that for every PA (and hence also BPA, BPP, or FS) system Δ we have that Δ is normed iff each $X \in Const(\Delta)$ is normed. The extra condition of normedness can substantially simplify certain bisimilarity-problems; for example, regularity w.r.t. bisimilarity is easily

[3] It has been shown in [25] that it does not make much sense to consider those restricted classes where A is more general than B or incomparable to B. Therefore, we only study the subclasses for which $A \subseteq B$.

decidable for normed PA processes in polynomial time [19], while the general problem is open and seems to be complicated. However, normedness is not a particular advantage when one tries to solve problems related to *simulation* equivalence, as we shall see in the next sections.

2.3. Minsky Machines

Almost all undecidability results in this paper are obtained by reduction from the halting problem for Minsky counter machines.

DEFINITION 2.6. *A counter machine* $\mathcal{M}$ with nonnegative counters $c_1, c_2, \cdots, c_m$ is a sequence of instructions

$$\begin{array}{ll} 1: & \mathrm{INS}_1 \\ 2: & \mathrm{INS}_2 \\ \vdots & \\ k-1: & \mathrm{INS}_{k-1} \\ k: & \texttt{halt} \end{array}$$

where each INS_i $(i = 1, 2, ..., k-1)$ is in one of the following two forms (assuming $1 \leq n, n', n'' \leq k, 1 \leq j \leq m$)

- $c_j := c_j + 1;$ `goto` n
- `if` $c_j = 0$ `then goto` n' `else` $(c_j := c_j - 1;$ `goto` $n'')$

The halting problem, i.e., the question whether or not $\mathcal{M}$ will reach its `halt` instruction, is undecidable even for Minsky machines with two counters initialized to zero [27].

3. THE RELATIONSHIP BETWEEN SIMULATION AND BISIMULATION EQUIVALENCE

In this section we concentrate on the relationship between simulation and bisimulation equivalence. It is a general trend that decidability results for bisimulation equivalence are positive, while the 'same' problems for simulation equivalence are undecidable. Major examples of that phenomenon come from the area of equivalence-checking (bisimilarity is decidable in various classes of infinite-state processes, while simulation equivalence is not), and from the area of regularity-testing (finiteness up to bisimilarity is often decidable, while finiteness up to simulation equivalence is not). BPP and BPA are examples for this [7, 5, 13], and some new examples will be also given in Section 4.

Now we propose a method which allows to 'reduce' certain simulation problems to their bisimulation counterparts. Although this 'reduction' is not effective in general (it cannot be expected), it works effectively for some (interesting) classes of infinite-state processes.

DEFINITION 3.1. For every image-finite transition system $T = (S, \mathcal{A}, \rightarrow)$ we define the transition system $\mathcal{B}(T) = (S, \mathcal{A}, \mapsto)$ where $\mapsto$ is given by

$$s \stackrel{a}{\mapsto} t \text{ iff } s \stackrel{a}{\rightarrow} t \text{ and } \forall u \in S : (s \stackrel{a}{\rightarrow} u \wedge t \sqsubseteq_s u) \Longrightarrow u \sqsubseteq_s t$$

Observe that $\mathcal{B}(T)$ is obtained from T by deleting certain transitions (only those are preserved which are maximal w.r.t. simulation preorder). As T is image-finite, for each

transition $s \stackrel{a}{\to} t$ there is a 'maximal' transition $s \stackrel{a}{\mapsto} t'$ such that $t \sqsubseteq_s t'$. As we often need to distinguish between processes 's of T' and 's of $\mathcal{B}(T)$', we denote the latter one by $s_{\mathcal{B}}$.

LEMMA 3.1. *Let $T = (S, \mathcal{A}, \to)$ be an image-finite transition system. For each $s \in S$ we have that $s =_s s_{\mathcal{B}}$.*

Proof. Obviously $s_{\mathcal{B}} \sqsubseteq_s s$. For the other direction, let us define the relation $R \subseteq S \times S$ as follows:

$$R = \{(t, u_{\mathcal{B}}) \mid t \sqsubseteq_s u\}$$

We prove that s is simulated by $s_{\mathcal{B}}$ in R. Clearly $(s, s_{\mathcal{B}}) \in R$; it remains to show that whenever $(t, u_{\mathcal{B}}) \in R$ and $t \stackrel{a}{\to} t'$, then there is a transition $u_{\mathcal{B}} \stackrel{a}{\mapsto} u'_{\mathcal{B}}$ with $(t', u'_{\mathcal{B}}) \in R$. As $t \sqsubseteq_s u$, there is at least one a-successor of u which simulates t'. Let u' be the maximal one of those a-successors w.r.t. simulation preorder (see above); then $u_{\mathcal{B}} \stackrel{a}{\mapsto} u'_B$ and $(t', u'_B) \in R$ as required. ■

THEOREM 3.1. *Let $T_1 = (S_1, \mathcal{A}, \to)$, $T_2 = (S_2, \mathcal{A}, \to)$ be image-finite transition systems, $s \in S_1$, $t \in S_2$. We have that $s =_s t$ iff $s_{\mathcal{B}} \sim t_{\mathcal{B}}$.*

Proof. The '$\Longleftarrow$' is obvious, as bisimilarity is finer than simulation equivalence and $s =_s s_{\mathcal{B}}$, $t =_s t_{\mathcal{B}}$ by Lemma 3.1. For the other direction, we show that the following relation $R \subseteq S_1 \times S_2$ is a bisimulation:

$$R = \{(u_{\mathcal{B}}, v_{\mathcal{B}}) \mid u_{\mathcal{B}} =_s v_{\mathcal{B}}\}$$

It clearly suffices because $(s_{\mathcal{B}}, t_{\mathcal{B}}) \in R$. By the definition of bisimulation, we must show that for each $u_{\mathcal{B}} \stackrel{a}{\mapsto} u'_{\mathcal{B}}$ there is a $v_{\mathcal{B}} \stackrel{a}{\mapsto} v'_{\mathcal{B}}$ with $(u'_{\mathcal{B}}, v'_{\mathcal{B}}) \in R$ and vice versa (we only show the first part; the other one is symmetric). Let $u_{\mathcal{B}} \stackrel{a}{\mapsto} u'_{\mathcal{B}}$. As $u_{\mathcal{B}} =_s v_{\mathcal{B}}$, we also have $u_{\mathcal{B}} \sqsubseteq_s v_{\mathcal{B}}$ and hence $v_{\mathcal{B}}$ must be able to 'match' the move $u_{\mathcal{B}} \stackrel{a}{\mapsto} u'_{\mathcal{B}}$ by performing some $v_{\mathcal{B}} \stackrel{a}{\mapsto} v'_{\mathcal{B}}$ with $u'_{\mathcal{B}} \sqsubseteq_s v'_{\mathcal{B}}$. Now it suffices to show that $v'_{\mathcal{B}} \sqsubseteq_s u'_{\mathcal{B}}$. As $u_{\mathcal{B}} =_s v_{\mathcal{B}}$, we also have $v_{\mathcal{B}} \sqsubseteq_s u_{\mathcal{B}}$ and hence the move $v_{\mathcal{B}} \stackrel{a}{\mapsto} v'_{\mathcal{B}}$ must be matched by some $u_{\mathcal{B}} \stackrel{a}{\mapsto} u''_{\mathcal{B}}$ with $v'_{\mathcal{B}} \sqsubseteq_s u''_{\mathcal{B}}$. To sum up, we have $u'_{\mathcal{B}} \sqsubseteq_s v'_{\mathcal{B}} \sqsubseteq_s u''_{\mathcal{B}}$ and hence $u'_{\mathcal{B}} \sqsubseteq_s u''_{\mathcal{B}}$ — but it also means that $u''_{\mathcal{B}} \sqsubseteq_s u'_{\mathcal{B}}$ by Definition 3.1 and Lemma 3.1. We obtain $u'_{\mathcal{B}} \sqsubseteq_s v'_{\mathcal{B}} \sqsubseteq_s u''_{\mathcal{B}} \sqsubseteq_s u'_{\mathcal{B}}$, hence $v'_{\mathcal{B}} \sqsubseteq_s u'_{\mathcal{B}}$ as required. ■

EXAMPLE 3.1. Let us consider the processes f, g, h of Fig. 1. We see that $f =_s g$, but $f \not\sim g$. According to Theorem 3.1, it should hold that $f_B \sim g_{\mathcal{B}}$ — and it is indeed the case since $g_{\mathcal{B}}$ has only one a-successor (the 'middle' one; the other two a-transitions lead to 'strictly weaker' states and therefore they are deleted).

The previous theorem also says that if we are to decide simulation equivalence between processes s and t of T_1 and T_2, we can instead check bisimilarity between processes $s_{\mathcal{B}}$ and $t_{\mathcal{B}}$ of $\mathcal{B}(T_1)$ and $\mathcal{B}(T_2)$, respectively. Similarly, if we are interested whether s is regular w.r.t. simulation equivalence, we can try to construct $\mathcal{B}(T)$ and check the regularity of $s_{\mathcal{B}}$ w.r.t. bisimilarity. This concept has recently been used in [15] where it is shown that the system $\mathcal{B}(T)$ is effectively constructible for transition systems generated by labeled Petri nets with

at most one unbounded place. More precisely, for each such net $\mathcal{N}$ which determines a transition system T one can effectively construct a one-counter automaton $\mathcal{A}$ such that the transition system which is generated by $\mathcal{A}$ is exactly $\mathcal{B}(T)$ (up to isomorphism). As a number of 'bisimulation' problems for one-counter automata are known to be decidable [12], some new (positive) decidability results for simulation on the restricted class of Petri nets have been obtained in this way.

It is also possible to attack undecidable simulation problems with the help of Theorem 3.1. For example, simulation equivalence is known to be undecidable for BPP processes [11], while bisimilarity is decidable [7]. Therefore, the system $\mathcal{B}(T)$, where T is generated by a BPP system, cannot be effectively definable in the BPP syntax in general. However, one can design a rich subclass of BPP systems where it *is* possible (by putting certain effectively checkable restrictions on BPP systems); see [22] for details.

In this paper, we use Theorem 3.1 to obtain a decidability result for PA processes (see Section 4).

4. THE DECIDABILITY BORDER

In this section we establish the decidability border of Fig. 2. We show that simulation preorder (in both directions) and simulation equivalence with finite-state processes are decidable for PDA processes in $EXPTIME$. It is possible to reduce each of the mentioned problems to the model-checking problem for an (almost) fixed formula φ of the alternation-free modal μ-calculus [18] and therefore we can apply the result of [31, 4] which says that model-checking the alternation-free modal μ-calculus for PDA processes is in $EXPTIME$.

Then we turn our attention to PA processes. We prove that, in contrast to the BPA and BPP subclasses, simulation preorder is *undecidable* between PA processes and finite-state ones in both directions. Moreover, simulation preorder is undecidable even if we consider those PA and finite-state processes which are *deterministic* and *normed*. Thus, our undecidability results immediately extend to trace preorder (which coincides with simulation preorder on deterministic processes). It is worth noting that simulation *equivalence* between deterministic PA and deterministic finite-state processes is decidable, as it coincides with bisimilarity which is known to be decidable [14]. However, as soon as we allow just one nondeterministic state in the PA process, simulation equivalence with finite-state processes becomes undecidable (there is even a fixed normed deterministic finite-state process F such that simulation equivalence with F is undecidable for PA processes). The same applies to trace equivalence.

Finally, we also prove that regularity (finiteness) of PA processes w.r.t. simulation and trace equivalence is undecidable, even for the normed subclass of PA. Again, the role of nondeterminism is very special as regularity of normed deterministic PA processes w.r.t. simulation and trace equivalence coincides with regularity w.r.t. bisimilarity, which is easily decidable in polynomial time [19]. However, just one nondeterministic state in the PA process suffices to make the undecidability proof possible.

THEOREM 4.1. *Simulation preorder is decidable between PDA processes and finite-state ones in $EXPTIME$ (in both directions).*

Proof. Let P be a PDA process with the underlying system Δ and F a finite-state process with the underlying system Γ. We construct another PDA system Δ', two processes A, B

of Δ', and a formula φ of the alternation-free modal μ-calculus such that $P \sqsubseteq_s F$ iff $A \models \varphi$, and $F \sqsubseteq_s P$ iff $B \models \varphi$.

We can safely assume that the set $Const(\Delta)$ can be partitioned into two disjoint subsets $Control(\Delta)$ and $Stack(\Delta)$, and that the rules of Δ are of the form $pX \stackrel{a}{\rightarrow} q\alpha$, where $p, q \in Control(\Delta)$, $X \in Stack(\Delta)$, and $\alpha \in Stack(\Delta)^*$. (It has been shown in [6] that PDA systems generate the same class of transition systems (up to isomorphism) as pushdown automata, and that each PDA system can be effectively transformed into an 'equivalent' pushdown automaton in such a way that the increase in size is only polynomial.) The system Δ' is constructed as follows:

- $Control(\Delta') := Control(\Delta) \times Const(\Gamma) \times \{0, 1\}$
- $Stack(\Delta') := Stack(\Delta) \cup \{Z_0\}$ where $Z_0 \notin Stack(\Delta)$
- for every rule $pX \stackrel{a}{\rightarrow} q\alpha$ of Δ and every $G \in Const(\Gamma)$ we add the rule $(p, G, 0)X \stackrel{a}{\rightarrow} (q, G, 1)\alpha$ to Δ'
- for every rule $G \stackrel{a}{\rightarrow} H$ of Γ, every $p \in Control(\Delta)$, and every $X \in Stack(\Delta')$ we add the rule $(p, G, 1)X \stackrel{a}{\rightarrow} (p, H, 0)X$ to Δ'

Intuitively, the system Δ' alternates the moves of Δ and Γ; the '0' and '1' stored in the finite control indicate whose turn it is. The new bottom symbol Z_0 is added so that F cannot 'get stuck' just due to the emptiness of the stack.

Let us consider a property φ of processes which can be informally described as follows: a process f satisfies φ iff for all a and $f \stackrel{a}{\rightarrow} f'$ there is a move $f' \stackrel{a}{\rightarrow} f''$ such that the state f'' also satisfies φ. This (recursively defined) property can be expressed in the modal μ-calculus [18] by putting

$$\varphi \equiv \nu X. \left(\bigwedge_{a \in \mathcal{A}} [a]\langle a \rangle X \right)$$

where $\mathcal{A} = Act(\Delta) \cup Act(\Gamma)$ (note that $\mathcal{A}$ is finite). Intuitively, the recursion is 'translated' into an explicit fixed-point definition. The problem whether a PDA process satisfies φ is decidable in $EXPTIME$ [31, 4].

Let P be of the form $p\alpha$. Keeping the intuitive interpretation of φ in mind, it is easy to see that $p\alpha \sqsubseteq_s F$ iff $(p, F, 0)\alpha Z_0 \models \varphi$, and similarly $F \sqsubseteq_s p\alpha$ iff $(p, F, 1)\alpha Z_0 \models \varphi$. ■

COROLLARY 4.1. *Simulation equivalence between PDA and finite-state processes is decidable in $EXPTIME$.*

REMARK 4.2. *Recently, it has been shown in [21] that the problem whether a PDA process can simulate a finite-state one, and the problem whether a PDA and a FS process are simulation equivalent, are both $EXPTIME$-hard. Hence, the reduction to the model-checking problem with φ used in the proof of Theorem 4.1 is an essentially optimal decision algorithm. The issue seems to be different with bisimilarity, which is known to be 'only' $PSPACE$-hard between PDA and FS processes [24]; in fact, we conjecture that even* weak *bisimilarity [26] between PDA and FS processes is a $PSPACE$-complete problem.*

Now we show that simulation preorder between PA and FS processes is already undecidable in both directions, even if those processes are deterministic and normed.

THEOREM 4.2. *Let P be a deterministic PA process and F a deterministic finite-state process. It is undecidable whether $P \sqsubseteq_s F$.*

Proof. Let $\mathcal{M}$ be an arbitrary Minsky machine with two counters initialized to m_1, m_2. We construct a deterministic PA process P and a deterministic finite-state process F such that $P \sqsubseteq_s F$ iff the machine $\mathcal{M}$ does not halt.

Let $\mathcal{A} := \{zero_1, inc_1, dec_1, zero_2, inc_2, dec_2\}$. The underlying system of P is defined by the following rules:

$$Z_1 \stackrel{zero_1}{\longrightarrow} Z_1,\ \ Z_1 \stackrel{inc_1}{\longrightarrow} C_1.Z_1,\ \ C_1 \stackrel{inc_1}{\longrightarrow} C_1.C_1,\ \ C_1 \stackrel{dec_1}{\longrightarrow} \varepsilon,$$
$$Z_2 \stackrel{zero_2}{\longrightarrow} Z_2,\ \ Z_2 \stackrel{inc_2}{\longrightarrow} C_2.Z_2,\ \ C_2 \stackrel{inc_2}{\longrightarrow} C_2.C_2,\ \ C_2 \stackrel{dec_2}{\longrightarrow} \varepsilon$$

We define $P \equiv (C_1^{m_1}.Z_1) \,\|\, (C_2^{m_2}.Z_2)$, where $C_i^{m_i}$, $i \in \{1,2\}$, denotes a sequential composition of m_i copies of the constant C_i.

The underlying system of F corresponds to the finite control of $\mathcal{M}$. For every instruction of the form

$$n:\ c_i := c_i + 1;\ \ \texttt{goto}\ n'$$

we have a rule $F_n \stackrel{inc_i}{\longrightarrow} F_{n'}$. For every instruction of the form

$$n:\ \texttt{if}\ c_i = 0\ \texttt{then goto}\ n'\ \texttt{else}\ (c_i := c_i - 1;\ \texttt{goto}\ n'')$$

we have rules $F_n \stackrel{zero_i}{\longrightarrow} F_{n'}$ and $F_n \stackrel{dec_i}{\longrightarrow} F_{n''}$. Then we add a new constant U and rules $U \stackrel{a}{\rightarrow} U$ for every $a \in \mathcal{A}$. Finally, we complete the system of F in the following way: For every constant F_i, except for the one which corresponds to the (label of the) halting instruction of $\mathcal{M}$, and every $a \in \mathcal{A}$, if there is no rule $F_i \stackrel{a}{\rightarrow} F_j$ for any F_j, then add a rule $F_i \stackrel{a}{\rightarrow} U$. The process F corresponds to the initial state of $\mathcal{M}$, i.e., $F \equiv F_1$.

The state of P corresponds to the contents of the counters of $\mathcal{M}$ and the state of F corresponds to the state of the finite control of $\mathcal{M}$. A simulation step corresponds to a computational step of $\mathcal{M}$.

The only problem is that P may do steps that do not correspond to steps of the counter machine, e.g., P does a step dec_1 when the current state in F expects inc_1. In all these cases the construction of the system of F ensures that F can (and must) respond by a step that ends in the state U. After such a step F can simulate anything. It is easy to see that $P \not\sqsubseteq_s F$ iff P can force F to enter the state corresponding to `halt` via a sequence of moves which correspond to the correct simulation of $\mathcal{M}$. Hence, $P \sqsubseteq_s F$ iff the machine $\mathcal{M}$ does not halt. ■

REMARK 4.3. *Theorem 4.2 still holds under the additional condition that the underlying systems of both the PA process and the finite-state one are normed. We can make the PA system normed by adding the following rules:*

$$Z_1 \stackrel{x_1}{\longrightarrow} \varepsilon,\ \ C_1 \stackrel{x_1}{\longrightarrow} \varepsilon,$$
$$Z_2 \stackrel{x_2}{\longrightarrow} \varepsilon,\ \ C_2 \stackrel{x_2}{\longrightarrow} \varepsilon$$

To make sure that F can simulate the actions x_1, x_2, we add the rules $N \stackrel{x_1}{\longrightarrow} U$ and $N \stackrel{x_2}{\longrightarrow} U$ for every constant N of the system of F (including U). Then, the system of

F is made normed by adding the rule $U \xrightarrow{x} \varepsilon$. It is easy to see that P and F are still deterministic, and still satisfy the property that $P \sqsubseteq_s F$ iff the machine $\mathcal{M}$ does not halt.

The halting problem is undecidable even for Minsky machines with two counters initialized to zero. The construction of P is then independent of $\mathcal{M}$. Furthermore, there exists a universal Minsky machine $\mathcal{M}'$; the halting problem for $\mathcal{M}'$ (with given input values) is undecidable, and the construction of F is independent of those input values. Hence we can conclude:

THEOREM 4.3. *There is a normed deterministic PA process $\overline{P}$ and a normed deterministic finite-state process $\overline{F}$ such that*

•the problem whether $\overline{P} \sqsubseteq_s F$ for a given (normed and deterministic) finite-state process F is undecidable,

•the problem whether $P \sqsubseteq_s \overline{F}$ for a given (normed and deterministic) PA process P is undecidable.

The other direction of simulation preorder is also undecidable, as we prove in the next theorem.

THEOREM 4.4. *Let P be a deterministic PA process and F a deterministic finite-state process. It is undecidable whether $F \sqsubseteq_s P$.*

Proof. Let $\mathcal{M}$ be an arbitrary Minsky machine with two counters initialized to m_1, m_2. We construct a deterministic PA process P and a deterministic finite-state system F such that $F \sqsubseteq_s P$ iff the machine $\mathcal{M}$ does not halt.

Let $\mathcal{A} := \{zero_1, inc_1, dec_1, zero_2, inc_2, dec_2, c\}$. For the construction of P we start with the same PA system as in Theorem 4.2 and extend it by the following rules, which handle all the behaviors that are 'illegal' in a given state of P w.r.t. the counter values it represents.

$$\begin{array}{ll} Z_1 \xrightarrow{dec_1} A_1, & C_1 \xrightarrow{zero_1} A_1, \\ Z_2 \xrightarrow{dec_2} A_2, & C_2 \xrightarrow{zero_2} A_2, \\ A_1 \xrightarrow{a} A_1 & \text{for every } a \in \{zero_1, inc_1, dec_1, c\}, \\ A_2 \xrightarrow{a} A_2 & \text{for every } a \in \{zero_2, inc_2, dec_2, c\} \end{array}$$

The intuition is that an illegal step that concerns the counter i (with $i \in \{1,2\}$) always introduces the symbol A_i, and from then on everything can be simulated. We define $P \equiv (C_1^{m_1}.Z_1) \parallel (C_2^{m_2}.Z_2)$ (where $C_i^{m_i}$, $i \in \{1,2\}$, denotes a sequential composition of m_i copies of the constant C_i). Note that P is deterministic; a term that contains both A_1 and A_2 can do the action c in two different ways, but the result is always the same.

The system of F corresponds to the finite control of $\mathcal{M}$. For every instruction of the form

$$n : \ c_i := c_i + 1; \ \texttt{goto} \ n'$$

we have a rule $F_n \xrightarrow{inc_i} F_{n'}$. For every instruction of the form

$$n : \texttt{if} \ c_i = 0 \ \texttt{then goto} \ n' \ \texttt{else} \ (c_i := c_i - 1; \ \texttt{goto} \ n'')$$

we have rules $F_n \stackrel{zero_i}{\longrightarrow} F_{n'}$ and $F_n \stackrel{dec_i}{\longrightarrow} F_{n''}$. For the unique instruction

$$k : \texttt{halt}$$

we add the rule $F_k \stackrel{c}{\rightarrow} F_k$. Note that a reachable state of P cannot do c, unless it contains A_1 or A_2. We let $F \equiv F_1$. A simulation step now corresponds to a computational step of $\mathcal{M}$. It follows that $F \not\sqsubseteq_s P$ iff F can reach the 'halting' state F_k via a sequence of legal steps that correspond to steps of the Minsky machine (and do not introduce the symbol A_1 or A_2 in P). Thus $F \sqsubseteq_s P$ iff the machine $\mathcal{M}$ does not halt. ■

REMARK 4.4. *Theorem 4.4 still holds under the additional condition that the underlying systems of both the PA process and the finite-state one are normed. The system of F is made normed by introducing the rules $N \stackrel{x}{\rightarrow} \varepsilon$ for every constant N of the system of F. To assure that P can always simulate the action x, we add the rules*

$$Z_1 \stackrel{x}{\rightarrow} \varepsilon,\ C_1 \stackrel{x}{\rightarrow} \varepsilon,\ A_1 \stackrel{x}{\rightarrow} \varepsilon$$

To make the system of P normed, it now suffices to add the following:

$$Z_2 \stackrel{y}{\rightarrow} \varepsilon,\ C_2 \stackrel{y}{\rightarrow} \varepsilon,\ A_2 \stackrel{y}{\rightarrow} \varepsilon$$

It is easy to see that P and F are still deterministic and satisfy the property that $F \sqsubseteq_s P$ iff the machine $\mathcal{M}$ does not halt.

The following theorem can be proved in the same way as Theorem 4.3.

THEOREM 4.5. *There is a normed deterministic PA process $\overline{P}$ and a normed deterministic finite-state process $\overline{F}$ such that*

- *the problem whether $F \sqsubseteq_s \overline{P}$ for a given (normed and deterministic) finite-state process F is undecidable,*
- *the problem whether $\overline{F} \sqsubseteq_s P$ for a given (normed and deterministic) PA process P is undecidable.*

We have seen that simulation preorder is undecidable between deterministic PA processes and deterministic finite-state ones in both directions. However, simulation *equivalence* (as well as any other equivalence of the linear time/branching time spectrum of [30]) is *decidable* for such a pair of processes, because it coincides with bisimilarity which is known to be decidable [14]. With the help of Theorem 3.1, we can extend the decidability result to all (not only deterministic) finite-state processes.

THEOREM 4.6. *Simulation equivalence is decidable between deterministic PA processes and (arbitrary) finite-state ones.*

Proof. As simulation preorder between finite-state processes is decidable, the system $\mathcal{B}(T)$ (see Definition 3.1) can be effectively constructed for any finite-state system T. Moreover, if T is deterministic then $\mathcal{B}(T) = T$. As bisimilarity between PA and FS processes is decidable [14], we can apply Theorem 3.1. ■

The decidability result of Theorem 4.6 is rather tight—in the next theorem we prove that simulation equivalence becomes *undecidable* as soon as we consider PA processes with just one nondeterministic state.

THEOREM 4.7. *There is a fixed normed deterministic finite-state process F such that the problem whether $P =_s F$ for a given normed PA process P is undecidable.*

Proof. We reduce the second undecidable problem of Theorem 4.3 to the problem if $P =_s F$. Let P' be a normed deterministic PA process, $\overline{F}$ be the fixed deterministic normed finite-state system derived from the finite control of the universal Minsky machine as in Theorem 4.3. We construct a normed PA process P and a fixed deterministic normed finite-state process F such that $P' \sqsubseteq_s \overline{F}$ iff $P =_s F$. It suffices to define F by $F \stackrel{a}{\to} \overline{F}$, and P by $P \stackrel{a}{\to} P'$, $P \stackrel{a}{\to} \overline{F}$. It follows immediately that $P =_s F$ iff $P' \sqsubseteq_s F'$. Note that P is not deterministic; however, it contains only one state (the P itself) where an action can be done in two different ways. ■

REMARK 4.5. *All undecidability results for simulation preorder which have been proved in this section immediately extend to trace preorder, because trace preorder coincides with simulation preorder in the class of deterministic processes. The argument of Theorem 4.7 carries over to trace equivalence as well.*

Now we prove that regularity w.r.t. simulation and trace equivalence is undecidable for normed PA processes with at least one nondeterministic state. It is interesting that regularity of normed deterministic PA processes w.r.t. any equivalence of the linear time/branching time spectrum of [30] is easily decidable in polynomial time, as it coincides with regularity w.r.t. bisimilarity which is known to have this property [19]. To see that a deterministic process P is regular w.r.t. bisimilarity iff it is regular w.r.t. any equivalence $\simeq$ which is not finer than bisimilarity and not coarser than trace equivalence (all equivalences of [30] fulfill this requirement), it suffices to realize that

- if P is regular w.r.t. bisimilarity, then $P \sim F$ for some finite-state process F, which means that $P \simeq F$ as $\simeq$ is not finer than bisimilarity;
- if P is regular w.r.t. $\simeq$, then $P \simeq F$ for some finite-state process F. It means that $P =_t F$, because $\simeq$ is not coarser than trace equivalence. Now we can use the standard subset construction [10] to obtain a deterministic finite-state system F' such that $F =_t F'$. As both P and F' are deterministic and trace equivalent, they are also bisimilar and hence $P \simeq F'$.

THEOREM 4.8. *Regularity w.r.t. simulation and trace equivalence is undecidable for normed PA processes.*

Proof. Let $\mathcal{M}$ be an arbitrary Minsky machine with two counters initialized to m_1, m_2. We construct a normed PA process Q such that Q is regular w.r.t. simulation (and trace) equivalence iff $\mathcal{M}$ does not halt.

Let P and F be the processes constructed in the proof of Theorem 4.2, modified in the same way as in Remark 4.3. The underlying system of Q is obtained by taking the disjoint union of the system of P and F, and extending it with the rules $Q \stackrel{a}{\to} P$, $Q \stackrel{a}{\to} F$ (note that

the resulting system is normed). If $\mathcal{M}$ does not halt (i.e., if $P \sqsubseteq_s F$), then Q is regular w.r.t. simulation and trace equivalence, because $Q =_s F'$ where the system of F' is the one of F extended with $F' \stackrel{a}{\to} F$. To complete the proof, we need to show that if $\mathcal{M}$ halts, then Q is not trace equivalent to any finite-state process. Let w be the sequence of actions which corresponds to the correct simulation of $\mathcal{M}$ by the process P. The process F can perform the sequence w, but it has to enter the 'halting' state F_k from which it can only emit the actions x_1, x_2 (see the proof of Theorem 4.2 and Remark 4.3). In particular, it means that F does not have any trace of the form $w\,v$ where $v \in \{inc_1, dec_1\}^+$. On the other hand, P can perform any trace of the form $w\,inc_1^n\,dec_1^n$ where $n \in \mathbb{N}$. Suppose there is a finite-state process G with k states such that $Q =_t G$. Then G must have a trace $a\,w\,inc_1^k\,dec_1^k$, and hence it can also perform the sequence $a\,w\,inc_1^k\,dec_1^m$ for any $m \in \mathbb{N}$ (here we use a well-known 'pumping' argument from the theory of finite automata [10]). However, Q does not have this property—each trace of Q which is of the form $a\,w\,v$ where $v \in \{inc_1, dec_1\}^+$ must satisfy the condition that $w\,v$ is a trace of P. If we choose $m = length(w) + k + 1$, then obviously P cannot do the sequence $w\,inc_1^k\,dec_1^m$. Hence $a\,w\,inc_1^k\,dec_1^m$ is a trace of G but not a trace of Q, and we have a contradiction. ■

5. THE TRACTABILITY BORDER

In this section we show that the problem whether a BPA process simulates a finite-state one is $PSPACE$-hard. The reverse preorder is shown to be co-$\mathcal{NP}$-hard. Consequently, we also obtain co-$\mathcal{NP}$-hardness of simulation equivalence between BPA and finite-state processes. All hardness proofs can be easily adapted so that they also work for BPP processes. As simulation preorder and equivalence are easily decidable for finite-state processes in polynomial time, the tractability border for simulation preorder/equivalence with finite-state systems of Fig. 2 is established.

THEOREM 5.1. *Let P be a BPA process, F a finite-state process. The problem whether $F \sqsubseteq_s P$ is $PSPACE$-hard.*

Proof. We show $PSPACE$-hardness by a reduction of the $PSPACE$-complete problem QBF. Let $n \in \mathbb{N}$ and $x_0, \ldots, x_{n-1}$ be boolean variables. We assume (without restrictions) that n is even. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. The quantified boolean formula Q is given by

$$Q := \forall x_0 \exists x_1 \ldots \forall x_{n-2} \exists x_{n-1} (Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid.

We reduce this problem to the simulation problem. Let us define a finite-state system Γ with constants $F_0, F_2, F_4, \ldots, F_n, Q_1, Q_2, \ldots, Q_k$ consisting of the following rules:

- $F_{2i} \stackrel{x_{2i}}{\longrightarrow} F_{2(i+1)}$ for each $0 \leq i \leq n/2 - 1$
- $F_{2i} \stackrel{\bar{x}_{2i}}{\longrightarrow} F_{2(i+1)}$ for each $0 \leq i \leq n/2 - 1$
- $F_n \stackrel{check}{\longrightarrow} Q_j$ for each $1 \leq j \leq k$
- $Q_j \stackrel{q_j}{\longrightarrow} Q_j$ for each $1 \leq j \leq k$

We also define a BPA system Δ with constants $P, X_1, X_2, \ldots, X_{n-1}, \overline{X}_1, \overline{X}_2, \ldots, \overline{X}_{n-1}$ which has the rules

- $P \xrightarrow{x_{2i}} P.X_{2i+1}.X_{2i}$ for each $0 \leq i \leq n/2-1$
- $P \xrightarrow{x_{2i}} P.\overline{X}_{2i+1}.X_{2i}$ for each $0 \leq i \leq n/2-1$
- $P \xrightarrow{\bar{x}_{2i}} P.X_{2i+1}.\overline{X}_{2i}$ for each $0 \leq i \leq n/2-1$
- $P \xrightarrow{\bar{x}_{2i}} P.\overline{X}_{2i+1}.\overline{X}_{2i}$ for each $0 \leq i \leq n/2-1$
- $P \xrightarrow{check} \varepsilon$
- $X_i \xrightarrow{q_j} X_i$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$ such that the literal x_i occurs in the clause Q_j
- $X_i \xrightarrow{q_j} \varepsilon$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$
- $\overline{X}_i \xrightarrow{q_j} \overline{X}_i$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$ such that the literal $\bar{x}_i$ occurs in the clause Q_j
- $\overline{X}_i \xrightarrow{q_j} \varepsilon$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$

Intuitively, the process F_0 guesses the assignment for variables with even index. P stores this assignment and adds its own assignment for the variables with odd index. After the action $check$ it is checked if the assignment satisfies the formula. It follows immediately from the construction of Δ that the assignment satisfies the formula iff the state which encodes the assignment can do each action q_j infinitely many times. If Q holds, then $F_0 \sqsubseteq_s P$ because P can choose the 'correct' assignment for variables with the odd index and then perform each q_j infinitely many times. If Q does not hold, then $F_0 \not\sqsubseteq_s P$ because F_0 can 'force' P to reach an assignment for which some Q_j is false; then it starts to perform q_j repeatedly and P inevitably reaches ε from which there are no moves. Hence, Q is valid iff $F_0 \sqsubseteq_s P$. ■

THEOREM 5.2. *Let P be a BPP process, F a finite-state process. The problem whether $F \sqsubseteq_s P$ is $PSPACE$-hard.*

Proof. The $PSPACE$-hardness proof of Theorem 5.1 carries over directly. We use the same rules for Δ with parallel composition instead of sequential composition. ■

THEOREM 5.3. *Let P be a BPA process, F a finite-state process. The problem whether $P \sqsubseteq_s F$ is co-$\mathcal{NP}$-hard.*

Proof. We reduce the $\mathcal{NP}$-complete problem SAT to the problem if $P \not\sqsubseteq_s F$. Let $n \in \mathbb{N}$ and $x_0, \ldots, x_{n-1}$ be boolean variables. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. The formula Q is given by

$$Q := Q_1 \wedge \ldots \wedge Q_k$$

where the Q_i are clauses. The problem is if Q is satisfiable.

We define a BPA system Δ with constants $P_0, P_1, \ldots, P_n, X_1, X_2, \ldots, X_{n-1}, \overline{X}_1, \overline{X}_2, \ldots, \overline{X}_{n-1}$ as follows:

- $P_i \xrightarrow{a} P_{i+1}.X_i$ for each $0 \leq i \leq n-1$
- $P_i \xrightarrow{a} P_{i+1}.\overline{X}_i$ for each $0 \leq i \leq n-1$
- $P_n \xrightarrow{check} \varepsilon$
- $X_i \xrightarrow{q_j} \varepsilon$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$ such that the literal x_i occurs in the clause Q_j

- $X_i \stackrel{b}{\longrightarrow} \varepsilon$ for each $0 \leq i \leq n-1$
- $\overline{X}_i \stackrel{q_j}{\longrightarrow} \varepsilon$ for all $0 \leq i \leq n-1$, $1 \leq j \leq k$ such that the literal $\bar{x}_i$ occurs in the clause Q_j
- $\overline{\overline{X}}_i \stackrel{b}{\longrightarrow} \varepsilon$ for each $0 \leq i \leq n-1$

Now we define a finite-state system Γ with constants $F, F_1, F_2, \ldots, F_k$ by

- $F \stackrel{a}{\longrightarrow} F$
- $F \stackrel{check}{\longrightarrow} F_i$ for each $1 \leq i \leq k$
- $F_i \stackrel{q_j}{\longrightarrow} F_i$ for all $1 \leq i \leq k$, $1 \leq j \leq k$ such that $i \neq j$
- $F_i \stackrel{b}{\longrightarrow} F_i$ for all $1 \leq i \leq k$

If Q is satisfiable then there is an assignment that satisfies all clauses Q_j. Then F cannot simulate P_0, because P_0 can choose this assignment and then it can perform a sequence of actions where each q_j is present (the sequence can also contain some 'auxiliary' occurrences of b); F cannot match this sequence because no F_i can do every action q_j. If Q is not satisfiable then in every assignment some Q_j is not true. Then F can simulate P_0 by going to the state F_j. Hence, Q is valid iff $P_0 \not\sqsubseteq_s F$. ■

THEOREM 5.4. *Let P be a BPP process and F a finite-state process. The problem whether $P \sqsubseteq_s F$ is co-$\mathcal{NP}$-hard.*

Proof. The proof is similar to the one of Theorem 5.3. The rules for Δ are like in Theorem 5.3 with parallel composition instead of sequential composition. Γ is defined in the same way, but we also add the rules $F \stackrel{b}{\longrightarrow} U$ and $F \stackrel{q_i}{\longrightarrow} U$ for every $1 \leq i \leq k$, and $U \stackrel{x}{\longrightarrow} U$ for every $x \in \{q_1, \ldots, q_k, a, b, check\}$. Intuitively, if some b or q_i is emitted before P_0 completes the guess (i.e., before $check$ is emitted), F goes to U where it can simulate everything. Again we have that Q is valid iff $P_0 \not\sqsubseteq_s F$. ■

COROLLARY 5.1. *The problems of simulation equivalence between BPA and finite-state processes, and between BPP and finite-state processes are co-$\mathcal{NP}$-hard.*

Proof. Let P be a BPA (or BPP) process and F a finite-state process. Let P' be defined by the rules $P' \stackrel{a}{\rightarrow} P$ and $P' \stackrel{a}{\rightarrow} F$ and F' be defined by the rule $F' \stackrel{a}{\rightarrow} F$. Then $P' =_s F'$ iff $P \sqsubseteq_s F$. The results follow from Theorem 5.3 and 5.4. ■

REMARK 5.6. *All of the obtained hardness results are also valid under the normedness assumption. Observe that the BPA systems constructed in the proof of Theorem 5.1 and Theorem 5.3* are *normed; the finite-state systems used in those proofs can be made normed by adding the transitions $Q_j \stackrel{q_j}{\rightarrow} \varepsilon$ for all $1 \leq j \leq k$ (in the case of Theorem 5.1), and $F_i \stackrel{b}{\rightarrow} \varepsilon$ for all $1 \leq i \leq k$ (in the case of Theorem 5.3). This extension does not influence the validity of any argument used in our proofs.*

6. SUMMARY AND CONCLUSIONS

Table 1 summarizes the known decidability results in the area of equivalence/preorder checking between infinite-state processes and finite-state ones. The results which have been

obtained in this paper are in boldface. In the case of trace preorder/equivalence/regularity we distinguish between deterministic infinite-state processes (left column) and general ones (right column); finite-state systems can be considered as deterministic here, because the subset construction [10] preserves trace equivalence.

TABLE 1

A summary of known decidability results

	BPA		BPP		PA		PDA		PN	
$\sim$ FS	yes [8]		yes [7]		yes [14]		yes [28]		yes [16]	
reg. $\sim$	yes [5]		yes [13]		?		?		yes [13]	
$\underline{\sqsubseteq}_s$ FS	**YES**		yes [16]		**NO**		**YES**		yes [16]	
FS $\underline{\sqsubseteq}_s$	**YES**		yes [16]		**NO**		**YES**		yes [16]	
$=_s$ FS	**YES**		yes [16]		**NO**		**YES**		yes [16]	
reg. $=_s$	?		?		**NO**		?		no [16]	
$\underline{\sqsubseteq}_t$ FS	yes	yes	yes [16]	yes [16]	**NO**	**NO**	yes	yes	yes [16]	yes [16]
FS $\underline{\sqsubseteq}_t$	yes	no	yes [16]	yes [16]	**NO**	no	yes	no	yes [16]	yes [16]
$=_t$ FS	yes	no	yes [16]	yes [16]	yes [14]	no	yes	no	yes [16]	yes [16]
reg. $=_t$	yes	no	yes [13]	?	?	no	yes	no	yes [13]	no [16]

The results for trace preorder/equivalence might be also interesting from the point of view of automata theory (trace preorder and equivalence are closely related to language inclusion and equivalence, respectively). All 'trace' results for BPA and PDA are consequences of the 'classical' ones for language equivalence (see [10]). It is interesting to compare those decidability issues with the ones for PA, especially in the deterministic subcase. Trace preorder with finite-state systems tends to be decidable for deterministic processes; PA is the only exception. At the same time, trace *equivalence* with finite-state systems is *decidable* for deterministic PA. The PA processes we used in our undecidability proofs are parallel compositions of two deterministic and normed BPA processes (which can be seen as deterministic CF grammars). The parallel composition corresponds to the *shuffle* operator on languages [10]. Thus, our results also bring some insight into the power of shuffle on (deterministic) CF languages.

Interesting open questions are left in the area of regularity-testing. We can conclude that all of the '?' problems are at least semidecidable, as it is possible to enumerate all finite-state systems and decide equivalence with them.

ACKNOWLEDGMENT

We would like to thank Javier Esparza who observed the idea of the proof of Theorem 4.1.

REFERENCES

1. P.A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In *Proceedings of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 1998.
2. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
3. Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.
4. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *Proceedings of CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
5. O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 1996.

6. D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.
7. S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 1993.
8. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.
9. J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):353–371, 1994.
10. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
11. H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 1994.
12. P. Jančar. Decidability of bisimilarity for one-counter processes. *Information and Computation*, 158(1):1–17, 2000.
13. P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimilarity. In *Proceedings of ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 1996.
14. P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2):409–433, 2001.
15. P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of STACS 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2000.
16. P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 1995.
17. P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In *Proceedings of SOFSEM'99*, volume 1725 of *Lecture Notes in Computer Science*, pages 404–413. Springer, 1999.
18. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
19. A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proceedings of FST&TCS'96*, volume 1180 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 1996.
20. A. Kučera. Efficient verification algorithms for one-counter processes. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2000.
21. A. Kučera. On simulation-checking with sequential systems. In *Proceedings of ASIAN 2000*, volume 1961 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2000.
22. A. Kučera and R. Mayr. Simulation preorder on simple process algebras. Technical report TUM-I9902, Institute for Informatics, TU-Munich, 1999.
23. A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proceedings of CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 1999.
24. R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proceedings of IFIP TCS'2000*, volume 1872 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2000.
25. R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
26. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
27. M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
28. D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second order logic. *Theoretical Computer Science*, 37(1):51–75, 1985.
29. W. Reisig. *Petri Nets—An Introduction*. Springer, 1985.
30. R.J. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.
31. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

Automatic Verification of Recursive Procedures with one Integer Parameter

Ahmed Bouajjani [a] Peter Habermehl [a] Richard Mayr [b,*]

[a] *LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu, F-75251 Paris Cedex 05. France.*

[b] *Department of Computer Science, Albert-Ludwigs-University Freiburg, Georges-Koehler-Allee Geb. 051, D-79110 Freiburg, Germany.*

Abstract

Context-free processes (BPA) have been used for dataflow analysis in recursive procedures with applications in optimizing compilers [6]. We introduce a more refined model called BPA($\mathbb{Z}$) that can model not only recursive dependencies, but also the passing of an integer parameter to a subroutine. Moreover, this parameter can be tested against conditions expressible in Presburger arithmetic. This new and more expressive model can still be analyzed automatically. We define $\mathbb{Z}$-input 1-CM, a new class of 1-counter machines that take integer numbers as input, to describe sets of configurations of BPA($\mathbb{Z}$). We show that the $Post^*$ (the set of successors) of a set of BPA($\mathbb{Z}$)-configurations described by a $\mathbb{Z}$-input 1-CM can be effectively constructed. The Pre^* (set of predecessors) of a regular set can be effectively constructed as well. However, the Pre^* of a set described by a $\mathbb{Z}$-input 1-CM cannot be represented by a $\mathbb{Z}$-input 1-CM in general and has an undecidable membership problem. Then we develop a new temporal logic based on reversal-bounded counter machines (i.e. machines which use counters such that the change between increasing and decreasing mode of each counter is bounded [9]) that can be used to describe properties of BPA($\mathbb{Z}$) and show that the model-checking problem is decidable.

Key words: Verification, Model Checking, Context-free Processes

⋆ An extended abstract of this paper appeared in the proceedings of MFCS 2001. This work was partially supported by the European Commission (Project Advance IST 1999-29082)

* Corresponding author.
Email addresses: abou@liafa.jussieu.fr (Ahmed Bouajjani), haberm@liafa.jussieu.fr (Peter Habermehl), mayrri@informatik.uni-freiburg.de (Richard Mayr).

1 Introduction

Besides their classical use in formal language theory, pushdown automata have recently gained importance as an abstract process model for recursive procedures. Algorithms for model checking pushdown automata have been presented in [3,1,14,4]. Reachability analysis for pushdown automata is particularly useful in formal verification. For example, the satisfaction of a safety property corresponds to the fact, that a certain set of "bad" configurations is not reachable. Polynomial algorithms for reachability analysis have been presented in [1] and further optimized in [5]. For most purposes in formal verification it is sufficient to consider BPA ('Basic Process Algebra'; also called context-free processes), the subclass of pushdown automata without a finite control. BPA have been used for dataflow analysis in recursive procedures with applications in optimizing compilers [6].

The weakness of BPA is that it is not a very expressive model for recursive procedures. It can model recursive dependencies between procedures, but not the passing of data between procedures or different instances of a procedure with different parameters.

Example 1 *Consider the following abstract model of recursive procedures P, Q, R, S and F, which take an integer number as argument: ($x|y$ means "x divides y").*

$$
\begin{array}{lll}
P(x): & \textit{If } x \geq 16 & \\
 & \qquad \textit{If } 8|x \textit{ then } Q(x+1) & \\
 & \qquad \textit{else } P(x-2) & \\
 & \textit{else } F(x) &
\end{array}
\qquad\qquad
\begin{array}{lll}
Q(x): & \textit{If } 2|x \textit{ then } R(x) & \\
 & \textit{else } S(x+1) &
\end{array}
$$

If one starts by calling procedure P (with any parameter) then procedure R will never be called, because P never calls Q with an even number as parameter. However, a BPA model for these procedures cannot detect this.

Thus, we define a new more expressive model called BPA($\mathbb{Z}$) that extends BPA with an integer parameter. Procedures are now called with an integer parameter that can be tested, modified and passed to subroutines. We limit ourselves to one integer parameter, because two would give the model full Turing power and make all problems undecidable. BPA($\mathbb{Z}$) is a compromise between expressiveness and automatic analysability. On the one hand it is much more expressive than BPA and can model more aspects of full programs. On the other hand it is still simple enough such that most verification problems about BPA($\mathbb{Z}$) stay decidable. For the verification of safety properties, it is particularly useful to have a symbolic representation of sets of configurations

and to be able to effectively construct representations of the Pre^* (the set of predecessors) and the $Post^*$ (the set of successors) of a given set of configurations. While finite automata suffice for describing sets of configurations of BPA, a more expressive formalism is needed for BPA($\mathbb{Z}$). We define $\mathbb{Z}$-input 1-CM, a new class of 1-counter machines that take integer numbers as input, to describe sets of configurations of BPA($\mathbb{Z}$). We show that the $Post^*$ (the set of successors) of a set described by a $\mathbb{Z}$-input 1-CM can be effectively constructed. The Pre^* (the set of predecessors) of a regular set can be effectively constructed as well. However, the Pre^* of a set described by a $\mathbb{Z}$-input 1-CM cannot be represented by a $\mathbb{Z}$-input 1-CM in general and has an undecidable membership problem.

We develop a new temporal logic based on reversal-bounded counter machines that can be used to describe properties of BPA($\mathbb{Z}$). By combining our result on the constructibility of the $Post^*$ with some results by Ibarra et al. on reversal bounded counter machines [9,10] we show that the model-checking problem is decidable.

2 BPA($\mathbb{Z}$)

We define BPA($\mathbb{Z}$), an extension of BPA, as an abstract model for recursive procedures with an integer parameter.

Presburger arithmetic is the first-order theory of integers with addition and linear ordering (see, e.g. [7,8,2]).

Definition 2 *A n-ary Presburger predicate $P(k_1, \ldots, k_n)$ is an expression in Presburger arithmetic of type boolean (i.e., the outermost operator is a logical operator or quantifier) that contains exactly n free variables $k_1, \ldots, k_n$ of type integer. A set S of n dimensional integer vectors is Presburger definable if there exists a n-ary Presburger predicate $P(k_1, \ldots, k_n)$ such that $(k_1, \ldots, k_n) \in S$ iff $P(k_1, \ldots, k_n)$ is true.*

Presburger definable sets are also known as *semilinear sets*.

Definition 3 *We define* integer symbol sequences *(ISS) to describe configurations of processes. ISS are finite sequences of the form $X_1(k_1)X_2(k_2)\ldots X_n(k_n)$ with $n \geq 0$, where the X_i are symbols from a given finite set and the $k_i \in \mathbb{Z}$ are integers. (The brackets are mere 'syntactic sugar' and can be omitted.) Greek letters $\alpha, \beta, \ldots$ are used to denote ISS. The constant ϵ denotes the empty sequence.*

Definition 4 *Let $Act = \{\tau, a, b, c, \ldots\}$ and $Const = \{X, Y, Z, \ldots\}$ be dis-*

joint sets of actions *and* process constants, *respectively. A BPA($\mathbb{Z}$)* (α, Δ) *is given by an initial configuration* α *(where* α *is an ISS) and a finite set* Δ *of conditional rewrite rules of the form*

$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\dots X_n(e_n), \quad P(k)$$

where

- $X \in Const$, $a \in Act$, k *is a free variable of type integer.*
- $\forall i \in \{1,\dots,n\}.\, X_i \in Const$.
- *For every* $i \in \{1,\dots,n\}$ e_i *is an expression of one of the following two forms:*
  - $e_i = k_i$ *for some constant* $k_i \in \mathbb{Z}$, *or*
  - $e_i = k + k_i$ *for some constant* $k_i \in \mathbb{Z}$.
- $P(k)$ *is a unary Presburger predicate.*

Note that n *can be* 0. *In this case the rule has the form* $X(k) \xrightarrow{a} \epsilon, \quad P(k)$. *We denote the finite set of constants used in* Δ *by* $Const(\Delta)$ *and the finite set of actions used in* Δ *by* $Act(\Delta)$. *These rewrite rules induce a transition relation on ISS by prefix-rewriting as follows: For any* α *we have* $X(q)\alpha \xrightarrow{a}_\Delta X_1(q_1)X_2(q_2)\dots X_n(q_n)\alpha$ *if there is a rewrite rule*

$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\dots X_n(e_n), \quad P(k)$$

such that the following conditions are satisfied.

- $P(q)$
- *If* $e_i = k_i$ *then* $q_i = k_i$.
- *If* $e_i = k + k_i$ *then* $q_i = q + k_i$.

In the following we use also the notation $\alpha \to_\Delta \beta$ *if* $\alpha \xrightarrow{a}_\Delta \beta$ *for some* a.

Remark 5 *The Presburger predicates can be used to describe side conditions for the application of rules, e.g., the rule*

$$X(k) \xrightarrow{a} Y(k-7)Z(k+1), \quad 3|k \wedge k \geq 8$$

can only be applied to ISS starting with $X(q)$ *where* q *is at least 8 and divisible by 3. Furthermore, we can use Presburger predicates to express rules with constants on the left-hand side, e.g., the rule* $X(5) \xrightarrow{a} Y(2)Z(17)$ *can be expressed by* $X(k) \xrightarrow{a} Y(2)Z(17), \quad k = 5$. *In the following we sometimes use rules with constants on the left-hand side as a shorthand notation.*

Remark 6 *If one extends the model BPA($\mathbb{Z}$) by allowing two integer parameters instead of one (i.e., BPA($\mathbb{Z},\mathbb{Z}$)), it becomes Turing-powerful, because it*

can simulate a Minsky 2-counter machine (in the sense that one can reduce the halting problem of a Minsky-2 counter machine to the reachability problem of a BPA($\mathbb{Z},\mathbb{Z}$)).
If one extends the model by allowing multiplication and division on the one integer parameter, it becomes Turing-powerful as well. This is because in this case one can encode two counters into one by Gödel-coding. Two counters that hold numbers n_1 and n_2 are represented by one counter holding $2^{n_1}3^{n_2}$. Thus, all verification problems for these extensions of BPA($\mathbb{Z}$) become undecidable.

Definition 7 *We say that a BPA($\mathbb{Z}$) is in* normal form *if it only contains the following three types of rules:*

$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2), \quad P(k)$$
$$X(k) \xrightarrow{a} Y(e), \quad P(k)$$
$$X(k) \xrightarrow{a} \epsilon, \quad P(k)$$

where e, e_1, e_2 are expressions and $P(k)$ is a unary Presburger predicate as in Def. 4.

We call the rules of the third type decreasing *and the first two types* nondecreasing.

Remark 8 *It is easy to see that general BPA($\mathbb{Z}$) can be simulated by BPA($\mathbb{Z}$) in normal form (it can execute the same sequences of actions different from τ) with the introduction of some auxiliary constants. Long rules are split into several short rules. For example the long rule $X(k) \xrightarrow{a} Y(k+1).Z(k-2).W(k+7)$ is replaced by $X(k) \xrightarrow{a} X'(k).W(k+7)$ and $X'(k) \xrightarrow{\tau} Y(k+1).Z(k-2)$. If one is only interested in the set of reachable configurations of the original BPA($\mathbb{Z}$) then one has to filter out the intermediate configurations that contain auxiliary constants. It will turn out in Section 4 that this is possible. We will show that the set of reachable configurations of a BPA($\mathbb{Z}$) in normal form can be represented by a $\mathbb{Z}$-input 1-CM (a special type of 1-counter machine). These $\mathbb{Z}$-input 1-CMs are closed under synchronization with finite automata. Thus, to filter out the intermediate configurations it suffices to synchronize with the finite automaton that accepts exactly all sequences not containing auxiliary constants.*

It is clear that a BPA($\mathbb{Z}$) can simulate a 1-counter machine. However, the set of reachable configurations of a BPA($\mathbb{Z}$) cannot be described by a normal 1-counter machine.

Example 9 *Consider the BPA($\mathbb{Z}$) with just one rule $X(k) \xrightarrow{a} X(k+1)X(k)$ and initial state $X(0)$. The set of reachable configurations are all decreasing sequences of the form $X(n)X(n-1)X(n-2)\ldots X(0)$ for any $n \in \mathbb{N}$. The language consisting of these sequences cannot be accepted by a normal 1-counter*

machine, no matter how the integer numbers are coded (e.g., in unary coding or in binary as sequences of 0 and 1). The reason is that one cannot test the equality of the counter against the input without losing the content of the counter during the test.

The central problem in this paper is to compute a representation of the set of reachable states of a BPA($\mathbb{Z}$).

Definition 10 *Let Δ be the set of rules of a BPA($\mathbb{Z}$) and L a language of ISS (describing configurations of the BPA($\mathbb{Z}$)). We define $Post^0_\Delta(L) = L$. By $Post_\Delta(L)$ we denote the set of all successors (reachable configurations) of elements of L w.r.t. Δ in one step. $Post_\Delta(L) = \{\beta \mid \exists \alpha \in L. \alpha \rightarrow_\Delta \beta\}$. Then, $Post^n_\Delta(L)$ is inductively defined as $Post^n_\Delta(L) = Post_\Delta(Post^{n-1}_\Delta(L))$ for $n > 0$. By $Post^*_\Delta(L)$ we denote the set of all successors (reachable configurations) of elements of L w.r.t. Δ, i.e. $Post^*_\Delta(L) = \bigcup_{n \geq 0} Post^n_\Delta(L)$. In the same way we define $Pre_\Delta(L) = \{\alpha \mid \exists \beta \in L. \alpha \rightarrow_\Delta \beta\}$, $Pre^n_\Delta(L)$ and $Pre^*_\Delta(L)$ for the predecessors of elements of L.*

3 Automata

We define several classes of automata that are used in our constructions. For alternating pushdown automata we use the definitions of [1].

Definition 11 *An alternating pushdown automaton (APDA for short) is a triple $\mathcal{P} = (P, \Gamma, \Delta)$ where P is a finite set of control locations, Γ is a finite stack alphabet and Δ is the set of transition rules with $\Delta \subseteq (P \times \Gamma) \times 2^{P \times \Gamma^*}$.*

A *configuration* is a tuple $\langle q, w \rangle$ with $q \in P$, $w \in \Gamma^*$.
If $((p, \gamma), \{(p_1, w_1), \ldots, (p_n, w_n)\}) \in \Delta$ then for every $w \in \Gamma^*$ the configuration $\langle p, \gamma w \rangle$ is an *immediate predecessor* of the set $\{\langle p_1, w_1 w \rangle, \ldots, \langle p_n, w_n w \rangle\}$, and this set is an *immediate successor* of $\langle p, \gamma w \rangle$. Intuitively, at the configuration $\langle p, \gamma w \rangle$ the APDA selects nondeterministically a transition rule of the form $((p, \gamma), \{(p_1, w_1), \ldots, (p_n, w_n)\})$ and forks into n copies in the configurations $\langle p_1, w_1 w \rangle, \ldots, \langle p_n, w_n w \rangle$.
A *run* of $\mathcal{P}$ for an initial configuration c is a tree of configurations with root c such that the children of each node c' are the configurations that belong to one of its immediate successors (nodes of the form $\langle p, \epsilon \rangle$ have no successors).
We define the *reachability relation* $\Rightarrow \subseteq (P \times \Gamma^*) \times 2^{P \times \Gamma^*}$ between configurations and sets of configurations. Informally, $c \Rightarrow C$ iff C is a finite frontier (finite maximal set of incomparable nodes) of a run of $\mathcal{P}$ starting from c. Formally, $\Rightarrow$ is the smallest subset of $(P \times \Gamma^*) \times 2^{P \times \Gamma^*}$ such that:

- $c \Rightarrow \{c\}$ for every $c \in P \times \Gamma^*$,

- if c is an immediate predecessor of C, then $c \Rightarrow C$,
- if $c \Rightarrow \{c_1, \ldots, c_n\}$ and $c_i \Rightarrow C_i$ for each $1 \leq i \leq n$, then $c \Rightarrow (C_1 \cup \ldots \cup C_n)$.

The set of predecessors of a set of configurations C is defined as $pre^*_{\mathcal{P}}(C) = \{c \in P \times \Gamma^* \mid \exists C' \subseteq C \,.\, c \Rightarrow C'\}$.

We can add a new accepting state q_a to P and designate an initial state $q_0 \in P$. Then the language $L(\mathcal{P}) \subseteq \Gamma^*$ *accepted* by $\mathcal{P}$ is defined as the set of initial stack contents w for which $\mathcal{P}$ starting in q_0 accepts, i.e. $L(\mathcal{P}) = \{w \mid \langle q_0, w\rangle \in pre^*_{\mathcal{P}}(\{\langle q_a, w'\rangle \mid w' \in \Gamma^*\})\}$.

An *alternating 1-counter machine* is an automaton with one integer counter which can be incremented, decremented, set to a value and tested for 0. Additionally, Presburger tests on the counter can be performed.

Definition 12 *An alternating 1-counter machine (ACM) is a tuple* $\mathcal{M} = (Q_M, \Delta_M)$, *where* Q_M *is a finite set of states and* $\Delta_M \subseteq Q_M \times 2^{Q_M \times Op}$ *is a set of transition rules, where* $Op = \{c := c + k \mid k \in \mathbb{Z}\} \cup \{c := k \mid k \in \mathbb{Z}\} \cup \{c = 0\} \cup \{P(c) \mid P(c)$ *is a unary Presburger predicate*$\}$.

A *configuration* of an ACM is a tuple $\langle q, d\rangle$ with $q \in Q_M$ and $d \in \mathbb{Z}$. If

$$(q, \{\ (q_1, c := c + k_1), \ldots, (q_n, c := c + k_n),$$
$$(q'_1, c := k'_1), \ldots, (q'_{n'}, c := k'_{n'}), (q''_1, P_1(c)), \ldots, (q''_{n''}, P_{n''}(c))\}) \in \Delta_M$$

then the configuration $\langle q, d\rangle$ is an *immediate predecessor* of the set $\{\langle q_1, d + k_1\rangle, \ldots, \langle q_n, d + k_n\rangle, \langle q'_1, k'_1\rangle \ldots \langle q'_{n'}, k'_{n'}\rangle, \langle q''_1, d\rangle \ldots \langle q''_{n''}, d\rangle\}$ provided that $P_i(d)$ is true for all $1 \leq i \leq n''$ and this set is an *immediate successor* of $\langle q, d\rangle$. If

$$(q, \{\ (q_1, c := c + k_1), \ldots, (q_n, c := c + k_n), (q'_1, c := k'_1), \ldots, (q'_{n'}, c := k'_{n'}),$$
$$(q''_1, c = 0), \ldots, (q''_{n''}, c = 0), (q'''_1, P_1(c)), \ldots, (q'''_{n'''}, P_{n'''}(c))\}) \in \Delta_M$$

then the configuration $\langle q, 0\rangle$ is an *immediate predecessor* of the set $\{\langle q_1, k_1\rangle, \ldots, \langle q_n, k_n\rangle, \langle q'_1, k'_1\rangle, \ldots, \langle q'_n, k'_n\rangle, \langle q''_1, 0\rangle, \ldots, \langle q''_{n''}, 0\rangle, \langle q'''_1, 0\rangle, \ldots, \langle q'''_{n'''}, 0\rangle\}$ provided that $P_i(0)$ is true for all $1 \leq i \leq n'''$ and this set is an *immediate successor* of $\langle q, 0\rangle$. In the same way as for APDA, we define a *run*, the *reachability relation*, and $pre^*_{\mathcal{M}}(C)$.
We can add an accepting state q_a to Q_M and designate an initial state $q_0 \in Q_M$. Then the language $L(\mathcal{M}) \subseteq \mathbb{Z}$ *accepted* by $\mathcal{M}$ is defined as the set of initial counter values $d \in \mathbb{Z}$ for which $\mathcal{M}$ starting in q_0 accepts, i.e. $L(\mathcal{M}) = \{d \mid \langle q_0, d\rangle \in pre^*_{\mathcal{P}}(\{\langle q_a, d'\rangle \mid d' \in \mathbb{Z}\})\}$.

We show in Lemma 17 that Presburger tests can be eliminated.

If we restrict the set of transition rules to a subset of $Q_M \times Q_M \times Op$ we obtain *1-counter machines* with Presburger tests. Their *reachability relation* $\Rightarrow \subseteq (Q_M \times \mathbb{Z}) \times (Q_M \times \mathbb{Z})$ is defined in the obvious way. We define $reach_{\mathcal{M}}(q, d, q') = \{d' \in \mathbb{Z} \mid \langle q, d\rangle \Rightarrow \langle q', d'\rangle\}$, i.e. the set of all counter values at state q' reachable

from a configuration $\langle q, d \rangle$.

Pushdown counter automata (PCA) have been introduced by Ibarra in [9].

Definition 13 *A pushdown counter automaton (PCA) [9] is a pushdown automaton that is augmented with a finite number of reversal-bounded counters (containing integers) which can be incremented, decremented and tested for 0. A counter is reversal bounded iff there is a fixed constant k s.t. in any accepting computation the counter can change at most k times between increasing and decreasing.*

Now we define a new class of 1-counter machines with infinite input. These $\mathbb{Z}$-input 1-counter machines consider whole integer numbers as one piece of input and can compare them to constants, or to the internal counter without changing the counter's value. Additionally, they have several other useful features like Presburger tests on the counter. $\mathbb{Z}$-input 1-counter machines will be used in Section 4 to represent sets of reachable configurations of BPA($\mathbb{Z}$).

Definition 14 *A $\mathbb{Z}$-input 1-counter machine M is described by a finite set of states Q, an initial state $q_0 \in Q$, a final state $q_f \in Q$, a non-accepting state $fail \in Q$, and a counter c that contains initially 0. The initial configuration is given by the tuple $(q_0, 0)$. It reads pieces of input of the form $S(i)$ where S is a symbol out of a given finite set and $i \in \mathbb{Z}$ is an integer number. The instructions have the following form (q is different from q_f and $fail$):*

(1) $(q: \ c := c + 1; \mathsf{goto}\ q')$
(2) $(q: \ c := c - 1; \mathsf{goto}\ q')$
(3) $(q: \ \mathsf{If}\ c \geq 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(4) $(q: \ \mathsf{If}\ c = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(5) $(q: \ \textit{Read input } S(i).\ \mathsf{If}\ S = X\ \mathsf{and}\ i = K\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(6) $(q: \ \textit{Read input } S(i).\ \mathsf{If}\ S = X\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(7) $(q: \ \mathsf{If}\ P(c)\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$, *where P is a unary Presburger predicate.*

where $X \in Const$ is a symbol constant and $K \in \mathbb{Z}$ is an integer constant.

$\mathbb{Z}$-input 1-counter machines can be nondeterministic, i.e., there can be several instructions at the same control state. Each transition arc to a new control state can be labeled with an atomic action. The *language* $L(M)$ accepted by a machine M is the set of ISS which are read by M in a run from the initial configuration to the state q_f.

In the following we use several shorthand notations for operations which can be encoded by the standard operations above. We use $c := c + j$ (incrementing the counter by a constant j), $c := j$ (setting the counter to a given constant j) and the operation $guess(c)$ (setting the counter to a nondeterministically

chosen integer).

It is now easy to see that the set of reachable states of Example 9 can be described by the following $\mathbb{Z}$-input 1-counter machine:

q_0 : $guess(c)$; goto q_1

q_1 : Read input $S(i)$. If $S = X$ and $i = c$ then goto q_2 else goto *fail*

q_2 : $c := c - 1$; goto q_1

q_2 : If $c = 0$ then goto q_f else goto *fail*

While instructions of type 6 (integer input) do increase the expressive power of 1-counter machines, this is not the case for instructions of type 7 (Presburger tests). The following lemma shows that instructions of type 7 can be eliminated from $\mathbb{Z}$-input 1-counter machines if necessary. We use them only as a convenient shorthand notation.

Lemma 15 *For every $\mathbb{Z}$-input 1-counter machine M with Presburger tests (i.e., instructions of type 7), an equivalent $\mathbb{Z}$-input 1-counter machine M' without Presburger tests can be effectively constructed (Equivalent means $L(M) = L(M')$).*

PROOF. Any Presburger formula can be written in a normal form that is a boolean combination of linear inequalities and tests of divisibility. As we consider only Presburger formulae with one free variable, it suffices to consider tests of the forms $c \geq k$, $c \leq k$ and $k|c$ for constants $k \in \mathbb{Z}$. Let K be the set of constants k used in these tests. K is finite and depends only on the Presburger predicates used in M. Let $K' = \{k_1, \ldots, k_m\} \subseteq K$ be the finite set of constants used in divisibility tests. For every control state q of M we define a set of control states of M' of the form $(q, j_1, \ldots, j_m)$ where $j_i \in \{0, \ldots, k_i - 1\}$ for every $i \in \{1, \ldots, m\}$. Now M' simulates the computation of M in such a way that M' is in a state $(s, j_1, \ldots, j_m)$ iff M is in state s and $j_i = c \bmod k_i$. For example if $K' = \{2, 5\}$ then the step $(s, n) \xrightarrow{c:=c+1} (s', n+1)$ of M yields e.g. the step $((s, 1, 2), n) \xrightarrow{c:=c+1} ((s', 0, 3), n+1)$ of M'. The divisibility tests thus become trivial in M', because this information is now encoded in the control states of M'. The linear inequality tests are even easier to eliminate. For example the test $c \geq 5$ can be done by decrementing the counter by 5, testing for ≥ 0 and re-incrementing by 5. Thus, the Presburger tests can be eliminated from M'. $\square$

It is only a matter of convention if a $\mathbb{Z}$-input 1-CM reads the input from left to right (the normal direction) or from right to left (accepting the mirror image

as in the example above). It is often more convenient to read the input from right to left (e.g., in Section 5), but the direction can always be reversed, as shown by the following lemma.

Lemma 16 *Let M be a $\mathbb{Z}$-input 1-CM that reads the input from right to left. A $\mathbb{Z}$-input 1-CM M' can be constructed that reads the input from left to right and accepts the same language as M.*

PROOF. (sketch) M' has the same control states as M plus a new initial state q_0' and a new final state q_f'. M' starts in configuration $(q_0', 0)$. It guesses a value for its counter and goes to q_f. Then it does the computation of M in reverse (reading the input from left to right) until it reaches q_0. It tests if the counter has value 0. If yes, it goes to q_f' and accepts. If no, then it doesn't accept. □

Now, we give some results concerning APDA and ACMs. In the same way as in Lemma 15 we can show the following lemma for ACMs:

Lemma 17 *For every alternating 1-counter machine $\mathcal{M}$ with Presburger tests, an equivalent alternating 1-counter machine $\mathcal{M}'$ without Presburger tests can be effectively constructed. (Equivalent means $L(\mathcal{M}) = L(\mathcal{M}')$).*

Theorem 18 *[1] Given an APDA $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$, $Pre^*_{\mathcal{P}}(\mathcal{C})$ (in particular $L(\mathcal{P})$) is regular and effectively constructible.*

With an APDA we can easily simulate an alternating 1-counter machine with Presburger tests: First, we eliminate the Presburger tests with Lemma 17. Then, with the stack we can easily simulate the counter. Because the Parikh-image of regular sets is Presburger definable (semilinear) [12], we obtain the following:

Corollary 19 *Let $\mathcal{M}$ be an alternating 1-counter machine with Presburger tests. Then, $L(\mathcal{M})$ is effectively Presburger definable.*

The next corollary follows from the fact that for a 1-counter machine without alternation successors correspond to predecessors of the reversed machine.

Corollary 20 *Let $\mathcal{M}$ be a 1-counter machine with Presburger tests, $q, q' \in Q_M$ and $d \in \mathbb{Z}$. Then, $reach_{\mathcal{M}}(q, d, q')$ is effectively Presburger definable.*

4 Constructing Post*

In this section we prove the following theorem:

Theorem 21 *Let Δ be a set of BPA($\mathbb{Z}$) rules in normal form and M a $\mathbb{Z}$-input 1-counter machine. Then a $\mathbb{Z}$-input 1-counter machine M' with $L(M') = Post^*_\Delta(L(M))$ can be effectively constructed.*

To prove this theorem we generalize the proof of a theorem in [1] which shows that the $Post^*$ of a regular set of configurations of a pushdown automaton is regular. This proof uses a saturation method, i.e. adding a finite number of transitions and states to the automaton representing configurations.

We cannot directly adapt this proof to BPA($\mathbb{Z}$), because process constants in a configuration can disappear for certain values of the parameter by applying decreasing rules. We show how to calculate a Presburger formula to characterize these values. This allows us to eliminate decreasing rules from Δ. This means that symbols produced by rules in some derivation can not disappear later. Then, we can apply the saturation method.

First, we show how to characterize for a given X the set $\{d \mid X(d) \rightarrow^*_\Delta \epsilon\}$ by a Presburger formula. We transform the set of rules Δ into an alternating 1-counter machine and use Corollary 19.

Lemma 22 *Let Δ be a set of BPA($\mathbb{Z}$) rules and X a process constant. Then a Presburger formula $P_X(d)$ with $\{d \mid P_X(d)\} = \{d \mid X(d) \rightarrow^*_\Delta \epsilon\}$ can be effectively constructed.*

PROOF. We construct an alternating 1-counter machine $\mathcal{M}$ with Presburger tests such that $L(\mathcal{M}) = \{d \mid X(d) \rightarrow^*_\Delta \epsilon\}$, i.e. $\mathcal{M}$ with initial counter value d has an accepting run iff $X(d) \rightarrow^*_\Delta \epsilon$. Then, we apply Corollary 19.

We construct $\mathcal{M} = (Q_M, \Delta_M)$ as follows: To each process constant Y of the BPA($\mathbb{Z}$) we associate a state q_Y in Q_M. The initial state of $\mathcal{M}$ is q_X and its accepting state q_a. Δ_M is the smallest set such that:
If Δ contains a non-decreasing rewrite rule $Z(k) \xrightarrow{a} X_1(e_1)X_2(e_2), \quad P(k)$, then $(q_Z, \{(q_{X_1}, op_1), (q_{X_2}, op_2), (q_a, P(c))\}) \in \Delta_M$ (where op_i $(i = 1, 2)$ is $c := c + k_i$ if $e_i = k + k_i$ and op_i is $c := k_i$ if $e_i = k_i$). If Δ contains a non-decreasing rewrite rule $Z(k) \xrightarrow{a} X_1(e_1), \quad P(k)$, then $(q_Z, \{(q_{X_1}, op_1), (q_a, P(c))\}) \in \Delta_M$ (where op_1 is $c := c + k_1$ if $e_1 = k + k_1$ and op_1 is $c := k_1$ if $e_1 = k_1$). If Δ contains a decreasing rule $Z(k) \xrightarrow{a} \epsilon, \quad P(k)$ then $(q_Z, \{(q_a, P(c))\}) \in \Delta_M$.
It is a clear, that a run of $\mathcal{M}$ with initial counter value d is accepting iff $X(d)$ can disappear with rules of Δ. $\square$

Lemma 23 *Let Δ be a set of BPA($\mathbb{Z}$) rules and M a $\mathbb{Z}$-input 1-counter machine representing a set of configurations. Then, we can effectively construct a set of rules Δ' without decreasing rules and a $\mathbb{Z}$-input 1-counter machine M', such that $Post^*_\Delta(L(M)) = Post^*_{\Delta'}(L(M'))$.*

PROOF. The proof is done in two steps. First we construct a machine M' such that $L(M') = Post^*_{\Delta_d}(L(M))$ (where $\Delta_d \subseteq \Delta$ is the set of decreasing rules in Δ), i.e. M' is the closure of M under decreasing rules. Then, we construct Δ' without decreasing rules such that $Post^*_\Delta(L(M)) = Post^*_{\Delta'}(Post^*_{\Delta_d}(L(M)))$.

Let us first construct M': The machine M represents a set of configurations. M' represents the closure of this set under application of decreasing rules. For each state q we add a new transition from the initial state q_0 to q. These transition is composed of $guess(c)$ (which sets the counter non-deterministically to some value) and a Presburger test $P_q(c)$ which characterizes all the counter values which can be obtained at state q by following a path from q_0 to q such that all process constants read on this path can disappear by applying rules of Δ.

We obtain $P_q(c)$ by first constructing a 1-counter machine with Presburger tests M'' mimicking the counter operations of M and then using Corollary 20. M'' is constructed from M as follows: M'' has the same states as M. All transitions which read a process constant X are replaced by the corresponding Presburger test $P_X(k)$ (with Lemma 22). Then $P_q(c)$ is the Presburger formula we get by applying Corollary 20 to characterize the set $reach_{M''}(q_0, 0, q)$.

Clearly, $L(M') = Post^*_{\Delta_d}(L(M))$.

Now, we construct Δ' which provides rules which non-deterministically guess what process constants will disappear later in a derivation. Obviously, only the first process constant from the left can disappear. Δ' contains all non-decreasing rules of Δ. Furthermore, for each conditional rewrite rule in Δ of the form

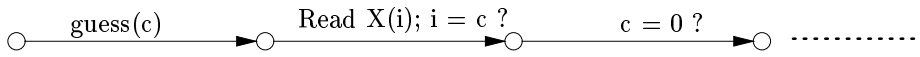
$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2), \quad P(k)$$

we add the rule

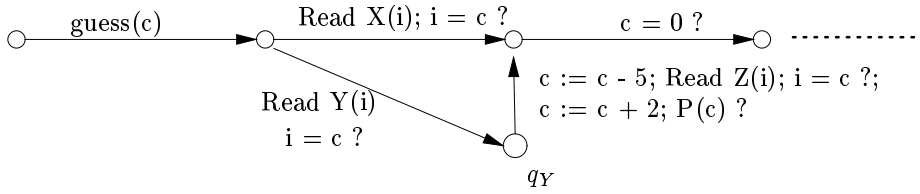
$$X(k) \xrightarrow{a} X_2(e_2), \quad P(k) \wedge P_{X_1}(e_1)$$

to Δ', where P_{X_1} is the Presburger formula of Lemma 22. Clearly, we have $Post^*_\Delta(L(M)) = Post^*_{\Delta'}(Post^*_{\Delta_d}(L(M)))$. $\square$

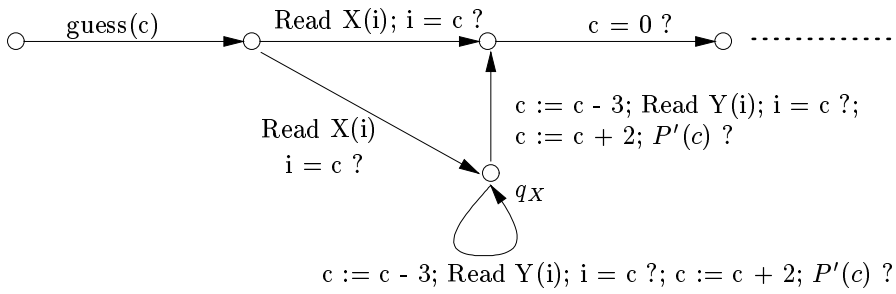
We use this lemma to prove Theorem 21. To construct a counter machine M' representing $Post^*_\Delta(L(M))$, given a counter machine M and a set of BPA($\mathbb{Z}$) rules Δ, it suffices to consider Δ which doesn't contain decreasing rules. Before giving the detailed construction and its correctness proof we explain the main idea with an example: Suppose we have a rule of the form $X(k) \xrightarrow{a} Y(k+3)Z(k-2), \quad P(k)$ in Δ and the automaton M is of the following form:



Notice that the counter is not tested before the input instruction. This is not a restriction (see Lemma 25). We add a new state q_Y for Y and transitions to M and obtain:



The transition going out of q_Y changes the counter value in such a way that if Y is read with parameter k then Z is read with parameter $k-5$, where -5 is the difference between -2 and 3. Then, the transition restores the counter value to the value before application of the rule by adding 2 and tests $P(c)$. Now consider instead a rule $X(k) \xrightarrow{a} X(k+1)Y(k-2) \quad P'(k)$ in Δ. Following the same principle as before we add a state for X and transitions. This will create a loop:



It is clear that in this way we only add a finite number of states (one for each process constant) and transitions. In the following we give the detailed proof

of our main theorem.

Proof of Theorem 21

First we show how to transform a $\mathbb{Z}$-input 1-CM into a special form:

Definition 24 *A $\mathbb{Z}$-input 1-CM M is said to be in special form iff*

- *¿From the initial state q_0 there is only an instruction guess(c) going to q_1.*
- *No instructions go back to q_0 or q_1.*
- *All instructions from q_1 are of the form*
  (q_1 : Read input $S(i)$. If $S = X$ and $i = c$ then goto q' else goto fail)
  or
  (q_1 : Read input $S(i)$. If $S = X$ and $i \neq c$ then goto q' else goto fail)
  We call this instructions first input instructions.
- *All instructions with a test of the counter ($c \geq 0$ or $c = 0$) are of the form*
  (q : If $test(c)$ then goto q' else goto fail)

Machines in special form guess a counter value, read an input and only then can test the counter and continue. We can prove the following lemma:

Lemma 25 *Any $\mathbb{Z}$-input 1-CM M can be replaced by a $\mathbb{Z}$-input 1-CM M' in special form which accepts the same language.*

PROOF. Putting the test instructions into the required form is trivial. To construct a machine where no tests on the counter are done before an input, we have to characterize all the counter values which can be obtained by taking a path (without inputs) from the initial configuration $\langle q_0, 0\rangle$ to another state q'. We can construct a Presburger formula $P(c)$ for this (using Corollary 20). Then we can construct a machine with states q_0 and q_1 as required by the special form (by putting the corresponding Presburger test behind each input instruction). At last, input instructions starting at q_1 of the form

(q_1 : Read input $S(i)$. If $S = X$ and $i = K$ then goto q' else goto *fail*)

can be replaced by

(q_1 : Read input $S(i)$. If $S = X$ and $i = c$ then goto q'' else goto *fail*)
(q'' : If $c = K$ then goto q'''else goto *fail*)
(q''' : *guess*(c); goto q')

The same is done for instructions with inequations. □

To prove Theorem 21 we have to show, that given a set Δ of BPA($\mathbb{Z}$) rules and a $\mathbb{Z}$-input 1-counter machine M we can construct a $\mathbb{Z}$-input 1-counter machine M' with $L(M') = Post^*_\Delta(L(M))$. Because of Lemma 23 we can suppose that Δ does not contain decreasing rules. We suppose that rules of Δ are in normal form and that M is in special form (Lemma 25). We first give the construction of M' and then we prove that $L(M') = Post^*_\Delta(L(M))$.

Construction of M'

In the following we will omit the else-part of all the instructions (they all go to *fail*). The construction of M' is done by adding states and instructions to the machine M. The basic idea is the following: Each rule of Δ can replace a process constant read starting from the initial state of the automaton M by other process constants. Therefore, instructions have to be added to M. These instructions have to change also the counter in order to simulate correctly the change in the parameters. Therefore, the counter has to be changed in such a way, that after reading the symbols on the right-hand side of a rule its value is the same as before. The special form of M insures that the counter is not tested before the input instructions. Furthermore, because there are no decreasing rules, only symbols read in the first input instructions can be replaced.

Let q_0 be the initial state of M and q_1 the state after the initial state. To simplify the presentation we only show how to treat input instructions with a test of the form $i = c$. The same can be done for $i \neq c$. For each process constant X we add a new state q_X and an instruction

$$(q_1 : \text{ Read input } S(i).\ \mathsf{If}\ S = X\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q_X\ ).$$

Now, for each input rule of the form

$$(q_1 : \text{ Read input } S(i).\ \mathsf{If}\ S = X\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q')$$

in M (including instructions added before) and for each rule in Δ of the following forms, we add instructions to M to obtain M'.

- $X(k) \xrightarrow{a} X_1(k + k_1), \quad P(k)$:
  add one instruction

  $$(q_{X_1} : \ c := c - k_1;\ \mathsf{If}\ P(c)\ \mathsf{then\ goto}\ q')$$

- $X(k) \xrightarrow{a} X_1(k_1), \quad P(k)$:
  add two instructions (q_n is a new state)

$(q_{X_1}$: If $c = k_1$ then goto $q_n)$
$(q_n$: guess(c); If $P(c)$ then goto $q')$

- $X(k) \xrightarrow{a} X_1(k + k_1)X_2(k + k_2), \quad P(k)$:
  add two instructions (q_n is a new state)

  $(q_{X_1} : c := c - k_1 + k_2;$ Read input $S(i)$.
  If $S = X_2$ and $i = c$ then goto $q_n)$
  $(q_n : c := c - k_2;$ If $P(c)$ then goto $q')$

- $X(k) \xrightarrow{a} X_1(k_1)X_2(k + k_2), \quad P(k)$:
  add three instructions (q_{n_1}, q_{n_2} are new states)

  $(q_{X_1}$: If $c = k_1$ then goto $q_{n_1})$
  $(q_{n_1}$: guess(c); Read input $S(i)$. If $S = X_2$ and $i = c$ then goto $q_{n_2})$
  $(q_{n_2}$: $c := c - k_2;$ If $P(c)$ then goto $q')$

- $X(k) \xrightarrow{a} X_1(k + k_1)X_2(k_2), \quad P(k)$:
  add two instructions (q_n is a new state)

  $(q_{X_1} : c := c - k_1;$ Read input $S(i)$.
  If $S = X_2$ and $i = k_2$ then goto $q_n)$
  $(q_n :$ If $P(c)$ then goto $q')$

- $X(k) \xrightarrow{a} X_1(k_1)X_2(k_2), \quad P(k)$:
  add three instructions (q_{n_1}, q_{n_2} are new states)

  $(q_{X_1}$: If $c = k_1$ then goto $q_{n_1})$
  $(q_{n_1}$: Read input $S(i)$. If $S = X_2$ and $i = k_2$ then goto $q_{n_2})$
  $(q_{n_2}$: guess(c); If $P(c)$ then goto $q')$

Since there are only a finite number of instructions starting at q_1 and a finite number of rules in Δ, it is obvious that only a finite number of instructions are added. In M' loops containing the states q_X can be created by the construction.

Correctness of M'

We have to show that $Post^*(L(M)) = L(M')$.

First, $Post^*(L(M)) \subseteq L(M')$:
We show by induction that $Post^n_\Delta(L(M)) \subseteq L(M')$ for all $n \in \mathbb{N}$. Base case: Obviously, we have $L(M) \subseteq L(M')$ because M' is obtained by adding states to M. Induction step: Consider an $\alpha \in Post^{n+1}_\Delta(L(M))$. Then, there exists an $\alpha' \in Post^n_\Delta(L(M))$ with $\alpha' \to_\Delta \alpha$. By induction hypothesis $\alpha' \in L(M')$. Now,

the construction of M' insures that $\alpha \in L(M')$, because for each rule of Δ there are corresponding transitions which are added to obtain M'.

Second, $L(M') \subseteq Post^*(L(M))$:
We prove this by induction on the number of new instructions (added to M by the construction) taken in accepting runs of M'. Let $\alpha \in L(M')$. If no new instruction is taken in an accepting run of α, then $\alpha \in L(M)$ and therefore $\alpha \in Post^*(L(M))$. Now, suppose that an accepting run of α in M' takes some new instructions. These are all taken at the beginning of the run. α must be of the form $X_1(m)\alpha'$. The machine M' takes first the $guess(c)$ instruction and then an input instruction of the form

$$(q_1 : \text{ Read input } S(i).\ \mathsf{If}\ S = X_1\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q_{X_1}\ \mathsf{else\ goto}\ fail)$$

Then, there are several cases to consider depending on the next instruction taken. We give the proof in detail for one case. All the others are done analogously. Suppose that the next instruction taken by the machine is

$$(q_{X_1} : \ c := c - k_1;\ \mathsf{If}\ P(c)\ \mathsf{then\ goto}\ q')$$

Therefore, $P(m - k_1)$ is true. By construction there is an instruction in M'

$$(q_1 : \text{ Read input } S(i).\ \mathsf{If}\ S = X\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q')$$

for some process constant X. Furthermore there is a rule $X(k) \xrightarrow{a} X_1(k + k_1), \quad P(k)$ in Δ. Because of $\alpha \in L(M')$ we have $X(m - k_1)\alpha' \in L(M')$ with an accepting run which takes less new instructions than the one for α. By induction hypothesis, $X(m - k_1)\alpha' \in Post^*(L(M))$ and furthermore $X(m - k_1)\alpha' \rightarrow_\Delta X(m)\alpha' = \alpha$. It follows, that $\alpha \in Post^*(L(M))$.

Thus Theorem 21 is proven. □

Remark 26 *While the language of reachable states of any BPA($\mathbb{Z}$) can be described by a $\mathbb{Z}$-input 1-CM, the converse is not true. Some $\mathbb{Z}$-input 1-CMs describe languages that cannot be generated by any BPA($\mathbb{Z}$). Consider the language*

$$\{X(k)Y(j_1)Y(j_2)\ldots Y(j_n)X(k) \mid k \in \mathbb{Z}, n \in \mathbb{N}, j_1, \ldots, j_n \in \mathbb{Z}\}$$

It is easy to construct a $\mathbb{Z}$-input 1-CM for this language (it just ignores the values of the j_i). However, no BPA($\mathbb{Z}$) generates this language, since it cannot guess the values of the arbitrarily many j_i without losing the value for k, which it needs again at the end.

The complexity of constructing a representation of Post* must be at least as high as the complexity of the reachability problem for BPA($\mathbb{Z}$). A special case of the reachability problem is the problem if the empty state ϵ is reachable from the initial state.

ϵ-REACHABILITY FOR BPA($\mathbb{Z}$)
Instance: A BPA($\mathbb{Z}$) Δ with initial state $X(0)$.
Question: $X(0) \rightarrow^* \epsilon$?

It is clear that for BPA($\mathbb{Z}$) with Presburger constraints the complexity of ϵ-reachability is at least as high as that of Presburger arithmetic. Consider a closed (i.e., without free variables) Presburger formula P and a BPA($\mathbb{Z}$) with one rule $X(k) \rightarrow \epsilon, \quad P(k)$. Then we have $X(0) \rightarrow^* \epsilon$ iff P is true. Presburger arithmetic is complete for the class $\bigcup_{k>1} TA[2^{2^{n^k}}, n]$ (see [13]), and thus requires at least doubly exponential time. Now we consider a restricted case of BPA($\mathbb{Z}$) without full Presburger constraints. In Remark 5 it was shown how Presburger constraints can be used to encode rules with constants on the left-hand side. Without rules with constants on the left-hand side BPA($\mathbb{Z}$) would not be very meaningful. In the following theorem we do not use full Presburger constraints, but we do use rules with constants on the left-hand side.

Theorem 27 *The ϵ-reachability problem for BPA($\mathbb{Z}$) without full Presburger constraints, but using integer constants in the left-hand sides of rules, is $\mathcal{NP}$-hard.*

PROOF. We reduce 3-SAT to ϵ-reachability. Let $Q := Q_1 \wedge \ldots \wedge Q_j$ be a boolean formula in 3-CNF with j clauses over the variables $x_1, \ldots, x_n$. We construct a BPA($\mathbb{Z}$) Δ with initial state $X(0)$ s.t. $X(0) \rightarrow^*_\Delta \epsilon$ iff Q is satisfiable. Let p_l be the l-th prime number. We encode an assignment of boolean values to $x_1, \ldots, x_n$ in a natural number x by Gödel coding, i.e., x_i is true iff x is divisible by p_i. The set of rules Δ is defined as follows:

$$
\begin{array}{lll}
X(k) & \rightarrow X(k+1) & \\
X(k) & \rightarrow Q_1(k+1).Q_2(k+1).\ldots.Q_j(k+1) & \\
Q_i(k) & \rightarrow X_l(k) & \text{if } x_l \text{ occurs in clause } Q_i. \\
Q_i(k) & \rightarrow \bar{X}_l(k) & \text{if } \bar{x}_l \text{ occurs in clause } Q_i. \\
X_l(k) & \rightarrow X_l(k-p_l) & \\
X_l(0) & \rightarrow \epsilon & \\
\bar{X}_l(k) & \rightarrow \bar{X}_l(k-p_l) & \\
\bar{X}_l(r) & \rightarrow \epsilon & \text{for every } r \in \{1, \ldots, p_l - 1\}
\end{array}
$$

The $X(k)$ is used to guess a number k that encodes an assignment to $x_1, \ldots, x_n$. If follows from the construction that $Q_i(k) \to^* \epsilon$ iff k encodes an assignment that makes clause Q_i true, $X_l(k) \to^* \epsilon$ iff k encodes an assignment where x_l is true and $\bar{X}_l(k) \to^* \epsilon$ iff k encodes an assignment where x_l is false. Thus we get $X(0) \to^* \epsilon$ iff Q is satisfiable. As the l-th prime number is $\mathcal{O}(l \cdot \log l)$, the size of Δ is $\mathcal{O}(jn + n^2 \log n)$. $\square$

5 The Constructibility of Pre*

In this section we show that the Pre^* of a regular set of configurations (w.r.t. a BPA($\mathbb{Z}$)) is effectively constructible. However, the Pre^* of a set of configurations described by a $\mathbb{Z}$-input 1-CM is not constructible. It is not even representable by a $\mathbb{Z}$-input 1-CM in general. Regular sets are given by finite automata. We define that finite automata ignore all integer input and are only affected by symbols. So, in the context of BPA($\mathbb{Z}$) we interpret the language $(ab)^*$ as $\{a(k_1)b(k_1') \ldots a(k_n)b(k_n') \mid n \in \mathbb{N}_0, \forall i.\, k_i, k_i' \in \mathbb{Z}\}$.

Theorem 28 *Let Δ be a BPA($\mathbb{Z}$) and R a finite automaton. Then a $\mathbb{Z}$-input 1-CM M can be effectively constructed s.t. $M = Pre^*_\Delta(L(R))$.*

PROOF. Every element in $Pre^*_\Delta(L(R))$ can be written in the form $\alpha X(k)\gamma$ where $\alpha \to^* \epsilon$, $X(k) \to^* \beta$ and $\beta\gamma \in L(R)$. Thus there must exist a state r in R s.t. there is a path from the initial state r_0 of R to r labeled β and a path from r to a final state of R labeled γ. We consider all (finitely many) pairs (X, r) where $X \in Const(\Delta)$ and $r \in states(R)$. Let R_r be the finite automaton that is obtained from R by making r the only final state. We compute the set of integers k for which there exists a β s.t. $X(k) \to^* \beta$ and $\beta \in L(R_r)$. First we compute the $\mathbb{Z}$-input 1-CM M_X in special form that describes $Post^*(X(k))$ as in Theorem 21. Then we compute the product of M_X with R_r, which is again a $\mathbb{Z}$-input 1-CM in special form. The set of counter values at state q_1 of M_X for which $M_X \times R_r$ is nonempty is Presburger definable and effectively computable (like in Corollaries 19 and 20). Let $P_{X,r}$ be the corresponding unary Presburger predicate. Let R_r' be the finite automaton that is obtained from R by making r the initial state. We define $M_{X,r}$ to be the $\mathbb{Z}$-input 1-CM that behaves as follows: First it accepts $X(k)$ iff $P_{X,r}(k)$ and then it behaves like R_r'. Let M_ϵ be the $\mathbb{Z}$-input 1-CM that accepts all sequences α s.t. $\alpha \to^* \epsilon$. M_ϵ is effectively constructible, since for every symbol Y the set of k for with $Y(k) \to^* \epsilon$ is Presburger and effectively constructible by Lemma 22. Then finally we get

$$M = M_\epsilon \cdot \bigcup_{X,r} M_{X,r}$$

and $M = Pre^*_\Delta(L(R))$. $\square$

Now we consider the problem of the Pre^* of a set of configurations described by a $\mathbb{Z}$-input 1-CM.

MEMBERSHIP IN Pre^* OF $\mathbb{Z}$-INPUT 1-CM

Instance: A BPA($\mathbb{Z}$) Δ, a $\mathbb{Z}$-input 1-CM M and a state $X_0(0)$
Question: $X_0(0) \in Pre^*_\Delta(M)$?

Theorem 29 *Membership in Pre^* of $\mathbb{Z}$-input 1-CM is undecidable.*

PROOF. We reduce the undecidable halting problem for Minsky 2-counter machines (with both counters initially 0) to the membership in Pre^* of a $\mathbb{Z}$-input 1-CM. The first observation is that $X_0(0) \in Pre^*_\Delta(M)$ iff $Post^*_\Delta(X_0(0)) \cap L(M) \neq \emptyset$. Let M' be a Minsky 2-counter machine. We will define the BPA($\mathbb{Z}$) Δ and the $\mathbb{Z}$-input 1-CM M in such a way that each of them simulates a 1-counter machine and together they simulate the 2-counter machine M'.

We define the BPA($\mathbb{Z}$) Δ in such a way that it correctly simulates the part of the computation of M' that only affects the first counter c_1. The integer parameter is used to store the first counter c_1.

- For every instruction of M' of the form $(X: \ c_1 := c_1 + 1; \mathsf{goto}\ X')$ we have a rule $X(k) \to X'(k+1)X(k)$.
- For every instruction of M' of the form $(X: \ \mathsf{If}\ c_1 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_1 := c_1 - 1; \mathsf{goto}\ X'')$ we have two rules $X(0) \to X'(0)X(0)$ and $X(k) \to X''(k-1)X(k), \quad k > 0$.
- For every instruction of M' of the form $(X: \ c_2 := c_2 + 1; \mathsf{goto}\ X')$ we have a rule $X(k) \to X'(k)X(k)$.
- For every instruction of M' of the form $(X: \ \mathsf{If}\ c_2 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_2 := c_2 - 1; \mathsf{goto}\ X'')$ we have two rules $X(k) \to X'(k)X(k)$ and $X(k) \to X''(k)X(k)$.

In the last of these four cases the BPA($\mathbb{Z}$) guesses the successor state, because it knows nothing about the counter c_2. Thus, $Post^*_\Delta(X_0(0))$ contains all correct computation sequences of M' starting at the initial control state X_0 and initial counter value 0, but also some wrong ones (if it has guessed wrongly in the fourth case). These sequences are read from right to left.

Then we use the $\mathbb{Z}$-input 1-CM M to simulate the other part of the computation of M' which affects the second counter c_2. The counter of M is used to store the second counter c_2 of M' (which is initially 0). M ignores all integer

input and only checks the symbols.

- For every instruction of M' of the form $(X : \ c_1 := c_1 + 1; \mathsf{goto}\ X')$ the machine M reads the input, but ignores it, goes to control state X' and leaves the internal counter unchanged.
- For every instruction of M' of the form $(X : \ \mathsf{If}\ c_1 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_1 := c_1 - 1; \mathsf{goto}\ X'')$ the machine M reads the input symbol, which is either X or X', and changes the control state accordingly to X or X'. M ignores the integer input and leaves the internal counter unchanged.
- For every instruction of M' of the form $(X : \ c_2 := c_2 + 1; \mathsf{goto}\ X')$ the machine M increases the internal counter by 1 and goes to the control state X'.
- For every instruction of M' of the form $(X : \ \mathsf{If}\ c_2 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_2 := c_2 - 1; \mathsf{goto}\ X'')$ the machine M checks if the internal counter is 0.
  - If the internal counter is 0, then it reads the input symbol and checks if it is X'. If yes, then it goes to the control state X'. If no, then it stops and rejects. The internal counter is left unchanged. The integer input is ignored.
  - If the internal counter is > 0 then it decrements the internal counter by 1, reads the input symbol and checks if it is X''. If yes, then it goes to the control state X''. If no, then it stops and rejects. The integer input is ignored.

The machine M only accepts in the final control state X_f, which is also the final state of M'. As for the BPA($\mathbb{Z}$) above, these computation sequences of M are read from right to left. This is not a restriction by Lemma 16.

Together Δ and M simulate the computation of M'. Δ ensures that the computation step is correct when the first counter is concerned. M does the same for the second counter and ensures that only those sequences are accepted that end in the final state X_f of M'.

So we get that $X_0(0) \in \mathit{Pre}^*_\Delta(M) \iff \mathit{Post}^*_\Delta(X_0(0)) \cap L(M) \neq \emptyset \iff$ M halts, and thus the membership problem in Pre^* is undecidable. $\square$

Theorem 29 does not automatically imply that the Pre^* of a $\mathbb{Z}$-input 1-CM (w.r.t. Δ) cannot be represented by a $\mathbb{Z}$-input 1-CM. It leaves the possibility that this $\mathbb{Z}$-input 1-CM is just not effectively constructible. (Cases like this occur, e.g., the set of reachable states of a classic lossy counter machine is semilinear, but not effectively constructible [11].) However, the following theorem shows that the Pre^* of a $\mathbb{Z}$-input 1-CM is not a $\mathbb{Z}$-input 1-CM in general.

Theorem 30 *Let Δ be a BPA($\mathbb{Z}$) and M a $\mathbb{Z}$-input 1-CM. Then, the set $Pre^*_\Delta(L(M))$ cannot be represented by a $\mathbb{Z}$-input 1-CM in general.*

PROOF. Let M' be the 2-counter machine that accepts if and only if the initial counter value in the first counter c_1 is a power of 2, i.e., 2^m for some positive integer m. Let Δ and M be defined as in the proof of Theorem 29.

We assume that $Pre^*_\Delta(M)$ could be represented by a $\mathbb{Z}$-input 1-CM and derive a contradiction. If there were a $\mathbb{Z}$-input 1-CM that represents $Pre^*_\Delta(M)$ then there would also exist a $\mathbb{Z}$-input 1-CM that represents $Pre^*_\Delta(M) \cap \{X_0(n) \mid n \in \mathbb{N}\} = \{X_0(n) \mid \exists m \in \mathbb{N}.\, n = 2^m\}$. This is a contradiction, because the set $\{n \mid \exists m \in \mathbb{N}.\, n = 2^m\}$ is not Presburger definable. $\Box$

6 The Logic and its Applications

We define a logic called ISL (Integer Sequence Logic) that can be used to verify properties of BPA($\mathbb{Z}$). It is interpreted over ISS (see Def. 3). We define a notion of satisfaction of an ISL formula by a BPA($\mathbb{Z}$) and show that the verification problem is decidable.

Let *const* denote the projection of ISS on sequences of constants obtained by omitting the integers; formally $const(X_1(k_1)X_2(k_2)\ldots X_m(k_m)) = X_1X_2\ldots X_m$. Then, the logic ISL is defined as follows:

Definition 31 *ISL formulae have the following syntax:*

$$F := (A_1, \ldots, A_n, P)$$

where $A_1, \ldots, A_n$ are finite automata over an alphabet of process constants, and P is an $(n-1)$-ary Presburger predicate. Formulae are interpreted over sequences w of the form $X_1(k_1)X_2(k_2)\ldots X_m(k_m)$, where the satisfaction relation is defined as follows:

$w \models F$ iff there exist words $w_1, \ldots, w_n$, constants $Y_1, \ldots, Y_{n-1}$ and integers $k_1, \ldots, k_{n-1}$ s.t. $w = w_1Y_1(k_1)w_2Y_2(k_2)\ldots w_{n-1}Y_{n-1}(k_{n-1})w_n$ and

- *$\forall i \in \{1, \ldots, n-1\}.\, A_i$ accepts $const(w_i)Y_i$.*
- *A_n accepts $const(w_n)$ and $P(k_1, \ldots, k_{n-1})$ is true.*

The set of sequences which satisfy a formula F is given by $[\![F]\!] = \{w \mid w \models F\}$.

Intuitively, ISL formulae specify regular patterns (using automata) involving a finite number of integer values which are constrained by a Presburger formula.

We use ISL formulae to specify properties on the configurations of the systems and not on their computation sequences, the typical use of specification logics in verification. For instance, when BPA($\mathbb{Z}$)'s are used to model recursive programs with an integer parameter, a natural question that can be asked is whether some procedure X can be called with some value k satisfying a Presburger constraint P. This can be specified by asking whether there is a reachable configuration corresponding to the pattern $Const^* X(k) Const^*$, where $P(k)$ holds. Using ISL formulae, we can specify more complex questions such as whether it is possible that the execution stack of the recursive program can contain two consecutive copies of a procedure with the same calling parameter. This corresponds to the pattern $Const^* X(k_1)(Const - \{X\})^* X(k_2) Const^*$, where $k_1 = k_2$.

The first result we show, is that we can characterize $[\![F]\!]$ by means of reversal bounded counter automata. However, elements of $[\![F]\!]$ are sequences over an infinite alphabet, since they may contain any integer. To characterize over a finite alphabet an element $w \in [\![F]\!]$ we can encode the integers in w *in unary*: a positive (resp. negative) integer k_i is replaced by k_i (resp. $-k_i$) occurrences of a symbol p_i (resp. n_i). Hence, given a set L of ISS, let $\widehat{L}$ denote the set of all sequences in L encoded in this way. We can characterize $\widehat{[\![F]\!]}$ with a reversal bounded counter automaton.

Lemma 32 *We can construct a reversal bounded counter automaton M over a finite alphabet Σ such that $\widehat{[\![F]\!]} = L(M)$.*

PROOF. The reversal bounded counter automaton M simulates sequentially the automata $A_1, \ldots, A_n$ in order to check if the input is of the correct regular pattern. After reading w_i (A_i has to be in an accepting state), the machine reads a sequence of symbols p_i or n_i and stores their length in corresponding reversal bounded counters. After the input has been completely read, the Presburger formula can be tested by using a finite number of other reversal bounded counters. $\Box$

Now, we define a notion of satisfaction between BPA($\mathbb{Z}$)'s and ISL formulae.

Definition 33 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ. Let F be an ISL-formula. We define that (w_0, Δ) satisfies the formula F iff it has a reachable configuration that satisfies F. Formally*

$$(w_0, \Delta) \models F \iff \exists w \in Post^*_\Delta(w_0).\, w \models F$$

To prove the decidability of the verification problem $(w_0, \Delta) \models F$, for a given

BPA($\mathbb{Z}$) (w_0, Δ) and a formula F we need the following definition and a lemma.

Definition 34 *Let L be a set of ISS. Then, $L|_k$ is the set of sequences w such that there exists a sequence $w' \in L$ with $k' \geq k$ integers such that w is obtained from w' by removing $k' - k$ integers and encoding the remaining integers in unary.*

Lemma 35 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ. Then we can construct a PCA M such that $L(M) = Post^*_\Delta(w_0)|_k$.*

PROOF. First by Theorem 21 we construct a $\mathbb{Z}$-input 1-CM M that accepts $Post^*_\Delta(w_0)$. We construct a PCA from M by (1) using the pushdown store to encode the counter (2) choosing non-deterministically exactly k input values which are compared to the counter. For these comparisons we need k additional reversal bounded counters (to avoid losing the counter value). □

Theorem 36 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ and $F = (A_1, \ldots, A_n, P)$ an ISL-formula. The problem $(w_0, \Delta) \models F$ is decidable.*

PROOF. Clearly, we have $Post^*_\Delta(w_0) \cap [\![F]\!] \neq \emptyset$ iff $\widehat{Post^*_\Delta(w_0)} \cap \widehat{[\![F]\!]} \neq \emptyset$, which is also equivalent to $\widehat{Post^*_\Delta(w_0)}|_{n-1} \cap \widehat{[\![F]\!]} \neq \emptyset$ since F cannot constrain more than $n-1$ integers. Then we show that $\widehat{Post^*_\Delta(w_0)}|_{n-1} \cap \widehat{[\![F]\!]} \neq \emptyset$ is decidable. This follows from Lemma 35, Lemma 32, the fact that the intersection of a CA language with a PCA language is a PCA language (Lemma 5.1 of [9]), and Theorem 5.2 of [9] which states that the emptiness problem of PCA is decidable. □

Finally, we consider another interesting problem concerning the analysis of BPA($\mathbb{Z}$)'s. When used to model recursive procedures, a natural question is to know the set of all the possible values for which a given procedure can be called. More generally, we are interested in knowing all the possible values of the vectors $(k_1, \ldots, k_n)$ such that there is a reachable configuration which satisfies some given ISL formula $F = (A_1, \ldots, A_{n+1}, P)$. We show that this set is effectively semilinear.

Theorem 37 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ, and let F be an ISL formula. Then, the set*

$$\{(k_1, \ldots, k_n) \in \mathbb{Z}^n \mid \exists w = w_1 Y_1(k_1) \ldots w_n Y_n(k_n) w_{n+1} \in Post^*_\Delta(w_0).\, w \models F\}$$

is effectively semilinear.

PROOF. As in the proof of Theorem 36, we can construct a PCA which recognizes the language $\widehat{Post^*_\Delta(w_0)|_n} \cap \widehat{[\![F]\!]}$. Then, the result follows from the fact that the Parikh image of a PCA language is semilinear (see Theorem 5.1 of [9]). □

7 Conclusion

We have shown that BPA($\mathbb{Z}$) is a more expressive and more realistic model for recursive procedures than BPA. The price for this increased expressiveness is that a stronger automata theoretic model ($\mathbb{Z}$-input 1-CM) is needed to describe sets of configurations, while simple finite automata suffice for BPA. As a consequence, the set of predecessors is no longer effectively constructible for BPA($\mathbb{Z}$) in general. However, the set of successors is still effectively constructible in BPA($\mathbb{Z}$) and thus many verification problems are decidable for BPA($\mathbb{Z}$), e.g., model checking with ISL. Thus, BPA($\mathbb{Z}$) can be used for verification problems like dataflow analysis, when BPA is not expressive enough. We expect that our results can be generalized to more expressive models (e.g., pushdown automata with an integer parameter), but some details of the constructions will become more complex.

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[2] A. Boudet and H. Comon. Diophantine equations, presburger arithmetic and finite automata. In *Proc. of CAAP'96*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer-Verlag, 1996.

[3] O. Burkart and B. Steffen. Pushdown processes: Parallel composition and model checking. In *CONCUR'94*, volume 836 of *LNCS*, pages 98–113. Springer Verlag, 1994.

[4] O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*. Springer Verlag, 1997.

[5] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. of CAV 2000*, volume 1855 of *LNCS*. Springer Verlag, 2000.

[6] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. of FoSSaCS'99*, volume 1578 of *LNCS*, pages 14–30. Springer Verlag, 1999.

[7] M. Furer. The complexity of presburger arithmetic with bounded quantifier alternation depth. *Theoretical Computer Science*, 18:105–111, 1982.

[8] E. Gradel. Subclasses of presburger arithmetic and the polynomial-time hierarchy. *Theoretical Computer Science*, 56:289–301, 1988.

[9] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[10] O. Ibarra, T. Bultan, and J. Su. Reachability analysis for some models of infinite-state transition systems. In *Proc. of CONCUR 2000*, volume 1877 of *LNCS*. Springer Verlag, 2000.

[11] R. Mayr. Undecidable problems in unreliable computations. In *Proc. of LATIN 2000*, volume 1776 of *LNCS*. Springer Verlag, 2000. Journal version to appear in TCS.

[12] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[13] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity*. Elsevier, 1990.

[14] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.