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.

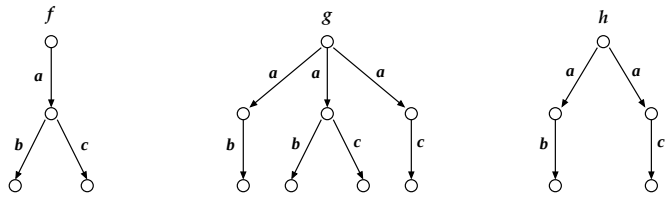

FIG. 1. Processes f, g, and h

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 [4]. 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 $\mathcal{G}$, 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 $E, F \in \mathcal{G}$, $E \neq \varepsilon$ 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.)

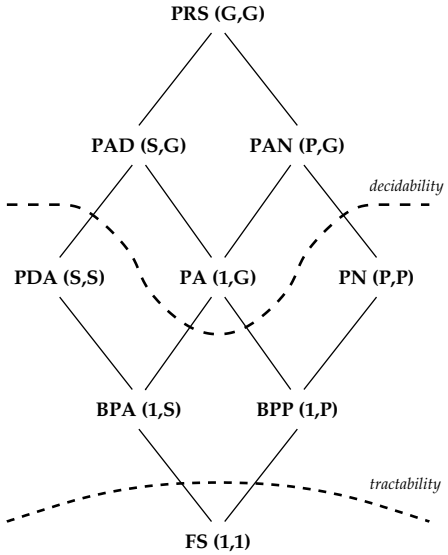

FIG. 2. A hierarchy of process rewrite systems with the decidability/tractability border for simulation with finite-state processes

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 $s \xrightarrow{a} t$ 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 $T = (S, \mathcal{A}, \rightarrow)$ 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 $t \stackrel{a}{\rightarrow} t'$, 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 $T_1 = (S_1, \mathcal{A}, \rightarrow)$, $T_2 = (S_2, \mathcal{A}, \rightarrow)$ 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, 3] 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, 3].

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 $F' \stackrel{a}{\rightarrow} F$. 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

- $X_i \xrightarrow{b} \varepsilon$ for each $0 \leq i \leq n-1$
- $\overline{X}_i \xrightarrow{q_j} \varepsilon$ 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
- $\overline{X}_i \xrightarrow{b} \varepsilon$ 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

- $F \xrightarrow{a} F$
- $F \xrightarrow{check} F_i$ for each $1 \leq i \leq k$
- $F_i \xrightarrow{q_j} F_i$ for all $1 \leq i \leq k$, $1 \leq j \leq k$ such that $i \neq j$
- $F_i \xrightarrow{b} F_i$ 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 $F \xrightarrow{b} U$ and $F \xrightarrow{q_i} U$ for every $1 \leq i \leq k$, and $U \xrightarrow{x} U$ 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 $P' \xrightarrow{a} P$ and $P' \xrightarrow{a} F$ and F' be defined by the rule $F' \xrightarrow{a} F$. 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 $Q_j \xrightarrow{q_j} \varepsilon$ for all $1 \leq j \leq k$ (in the case of Theorem 5.1), and $F_i \xrightarrow{b} \varepsilon$ 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]	
$\sqsubseteq_s$ FS	**YES**		yes [16]		**NO**		**YES**		yes [16]	
FS $\sqsubseteq_s$	**YES**		yes [16]		**NO**		**YES**		yes [16]	
$=_s$ FS	**YES**		yes [16]		**NO**		**YES**		yes [16]	
reg. $=_s$	?		?		**NO**		?		no [16]	
$\sqsubseteq_t$ FS	yes	yes	yes [16]	yes [16]	**NO**	**NO**	yes	yes	yes [16]	yes [16]
FS $\sqsubseteq_t$	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. 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, F. Moller, and B. Steffen. Verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.
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.