TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Simulation Preorder on Simple Process Algebras

Antonín Kučera, Richard Mayr

TUM-I9902
SFB-Bericht Nr. 342/01/99 A
Januar 99

TUM–INFO–01-I9902-80/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Simulation Preorder on Simple Process Algebras

Antonín Kučera*
Faculty of Informatics
Masaryk University
Botanická 68a, 60200 Brno
Czech Republic
tony@fi.muni.cz

Richard Mayr
Institut für Informatik
Technische Universität München
Arcisstr. 21, D-80290 München
Germany
mayrri@in.tum.de

Abstract

We consider the problem of simulation preorder/equivalence between infinite-state processes and finite-state ones. We prove that simulation preorder (in both directions) and simulation equivalence are *intractable* between all major classes of infinite-state 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.

The *decidability border* for the mentioned problem is also established. Simulation preorder (in both directions) and simulation equivalence are shown to be 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. The obtained 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 describe a way how to utilize decidability of bisimulation problems to solve certain instances of undecidable simulation problems. We apply this method to BPP processes.

*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 and complexity of the problem of checking simulation preorder/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). The same problem has been studied recently for strong and weak bisimilarity [14, 19], 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, giving a complete overview (see Figure 1).

The state of the art: Simulation preorder/equivalence is known to be undecidable for BPA [10] and BPP [12] processes. Consequently, it is also undecidable for any superclass of BPA and BPP, in particular for pushdown (PDA) processes and Petri nets. An interesting positive result is [1], where it is shown that simulation preorder (and hence also equivalence) is decidable for Petri nets with at most one unbounded place.

In [15] it is shown that 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.

Our contribution: In Section 3 we concentrate on 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. Those hardness results are also valid for any superclass of BPA and BPP processes, hence the main message of this section is that simulation with finite-state systems is unfortunately *intractable* for any studied class of infinite-state systems (assuming $\mathcal{P} \neq \mathcal{NP}$)—see Figure 1. 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 $\mathcal{P}$ [19].

In Section 4 we establish the decidability border of Figure 1. First we prove that simulation preorder between PDA processes 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 if we restrict ourselves to 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 process, 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* the PA or the finite-state process in each of the mentioned undecidable problems (for example, there is a fixed normed deterministic PA process P such that for a given (normed and deterministic) finite-state process F it undecidable whether P simulates F). Then we demonstrate that regularity of (normed) PA processes w.r.t. simulation equivalence is also undecidable. It contrasts sharply with regularity w.r.t. bisimilarity for normed PA processes, which is decidable in polynomial time [18]. All the obtained undecidability results also hold for trace preorder and trace equivalence, and therefore they might be also interesting from the point of view of the 'classical' automata theory. See the last section for further comments.

Finally, in Section 5 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 way 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 the 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 absolutely necessary for problems of 'real' size. We also provide a little 'case-study' where we show how to apply the proposed technique to BPP processes; in this way we obtain a large subclass of BPP processes where simulation equivalence as well as regularity w.r.t. simulation equivalence are decidable.

In the last section we give a summary of existing results in the area of comparing infinite-state systems with finite-state ones. We also extensively discuss some aspects of the obtained results.

2 Definitions

2.1 Process Rewrite Systems

Let $Act = \{a, b, c, \ldots\}$ be a countably infinite set of *actions*. Let $Const = \{X, Y, Z, \ldots\}$ be a countably infinite set of *process constants* such that $Act \cap Const = \emptyset$. The class of *process expressions* is defined by the following abstract syntax equation:

$$E \ ::= \ \epsilon \mid X \mid E \| E \mid E.E$$

Here X ranges over $Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, the '.' operator corresponds to a sequential 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 '$\|$'.

A *process rewrite system* [20] 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 $a \in Act$. We use $Const(\Delta)$ and $Act(\Delta)$ to denote the set of process constants and actions which are used in the rules of Δ, respectively (note that $Const(\Delta)$ and $Act(\Delta)$ are finite).

Each process rewrite system Δ determines 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):

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

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 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 G to denote the class of all process expressions, and 1 to denote the set of process constants. The 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. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA, BPP, and PA processes are well-known [2], PDA correspond to pushdown processes (as proved by Caucal in [5]), PN correspond to Petri nets (see e.g. [23]), etc.

In Figure 1 we also indicated the decidability/tractability border for simulation preorder and equivalence with finite-state systems which is established in the following sections.

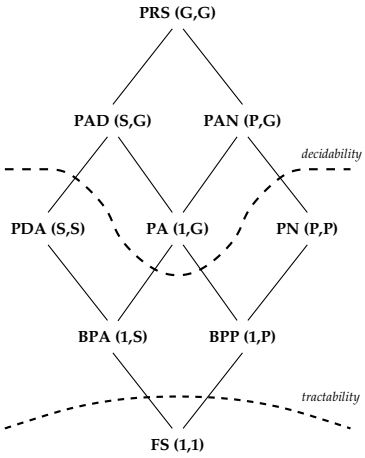

Figure 1: A hierarchy of process rewrite systems with the decidability/tractability border for simulation with finite-state processes

In the rest of this paper we consider *processes* as (being associated with) states in transition systems generated by process rewrite systems. We also assume that for each system Δ there is some distinguished process expression which is considered as a 'default' process, or *initial state* of Δ. In what follows, we often identify process rewrite systems with their initial states. A process P is said to be *deterministic* iff each reachable state of P has at most one a-successor for every $a \in Act$. A process is *normed* iff from every reachable state it has a terminating computation. It follows that a BPA, BPP, or PA process Δ is normed iff $X \rightarrow^* \epsilon$ for every $X \in Const(\Delta)$.

2.2 Behavioral Equivalences

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 on the class of 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 1. *We say that $w \in Act^*$ is a* trace *of a process E iff $E \stackrel{w}{\rightarrow} E'$ for some E'. Let $Tr(E)$ be the set of all traces of E. We write $E \sqsubseteq_t F$ iff $Tr(E) \subseteq Tr(F)$. Moreover, we say that E and F are* trace equivalent, *written $A =_t B$, iff $Tr(E) = Tr(F)$.*

In concurrency theory, trace equivalence is usually considered as being too coarse. A plethora of finer 'behavioral' equivalences have been proposed [24]. It seems that *simulation* and *bisimulation* equivalence are of special importance, as their accompanying theory has been developed very intensively.

Definition 2. *A binary relation R over process expressions is a* simulation *if whenever $(E, F) \in R$ then for each $a \in Act$*

$$\text{if } E \xrightarrow{a} E', \text{ then } F \xrightarrow{a} F' \text{ for some } F' \text{ such that } (E', F') \in R$$

A symmetric simulation is called bisimulation. *A process E is* simulated *by a process F, written $E \sqsubseteq_s F$, if there is a simulation R s.t. $(E, F) \in R$. We say that E and F are* simulation equivalent*, written $E =_s F$, iff $E \sqsubseteq_s F$ and $F \sqsubseteq_s E$. Similarly, we say that E and F are* bisimilar *(or* bisimulation equivalent*), written $E \sim F$, iff there is a bisimulation relating them.*

Another natural (and studied) problem is decidability of *regularity* (i.e. 'semantical finiteness') of processes w.r.t. certain behavioral equivalences.

Definition 3. *A process E is* regular *w.r.t. bisimulation (or simulation, trace) equivalence iff there is a finite-state process F such that $E \sim F$ (or $E =_s F$, $E =_t F$, respectively).*

2.3 Minsky Machines

Almost all undecidability results in this paper are obtained by reduction of the halting problem for Minsky counter machines.

Definition 4. A counter machine $\mathcal{M}$ *with nonnegative counters $c_1, c_2, \cdots, c_m$ is a sequence of instructions*

$$\begin{array}{ll} 1: & INS_1 \\ 2: & INS_2 \\ \vdots & \\ k-1: & INS_{k-1} \\ k: & \texttt{halt} \end{array}$$

where each INS_i ($i = 1, 2, ..., k-1$) is in one of the following two forms (assuming $1 \le l, l', l'' \le k$, $1 \le j \le m$)

- $c_j := c_j + 1$; $\texttt{goto}\ l$
- $\texttt{if}\ c_j = 0\ \texttt{then goto}\ l'\ \texttt{else}\ (c_j := c_j - 1;\ \texttt{goto}\ l'')$

The halting problem is undecidable even for Minsky machines with two counters initialized to zero [21].

3 The Tractability Border

In this section we show that the problem whether a BPA process simulates a finite-state one is *PSPACE*-hard. The other 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 Figure 1 is established.

Theorem 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 $n \in \mathbf{N}$ and $x_0, \ldots, x_{n-1}$ 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

$$Q := \forall x_0 \exists x_1 \ldots \forall x_{n-2} \exists x_{n-1} (Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid.

We reduce this problem to the simulation problem. A finite-state system Γ with initial state s_0 is defined as follows: $s_{2i} \stackrel{x_{2i}}{\rightarrow} s_{2(i+1)}$ and $s_{2i} \stackrel{\bar{x}_{2i}}{\rightarrow} s_{2(i+1)}$ for $0 \leq i \leq n/2-1$, $s_n \stackrel{check}{\longrightarrow} q_j$ for $1 \leq j \leq k$, and $q_j \stackrel{q_j}{\rightarrow} q_j$ for $1 \leq j \leq k$. A BPA system Δ with initial state Z is defined by the following rules:

$Z \stackrel{x_{2i}}{\rightarrow} Z.X_{2i+1}.X_{2i}$	$Z \stackrel{x_{2i}}{\rightarrow} Z.\bar{X}_{2i+1}.X_{2i}$	for $0 \leq i \leq n/2-1$
$Z \stackrel{\bar{x}_{2i}}{\rightarrow} Z.X_{2i+1}.\bar{X}_{2i}$	$Z \stackrel{\bar{x}_{2i}}{\rightarrow} Z.\bar{X}_{2i+1}.\bar{X}_{2i}$	for $0 \leq i \leq n/2-1$
$Z \stackrel{check}{\longrightarrow} \epsilon$		
$X_i \stackrel{q_j}{\rightarrow} X_i$	if X_i occurs in Q_j	
$X_i \stackrel{q_j}{\rightarrow} \epsilon$	if X_i does not occur in Q_j	
$\bar{X}_i \stackrel{q_j}{\rightarrow} \bar{X}_i$	if $\bar{X}_i$ occurs in Q_j	
$\bar{X}_i \stackrel{q_j}{\rightarrow} \epsilon$	if $\bar{X}_i$ does not occur in Q_j	

The process s_0 guesses the assignment for variables with even index. Z 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. If Z can simulate s_0 then every clause Q_i is true. If Z cannot simulate s_0 then this is because Z cannot do some action q_j and thus Q_j is not true. Thus Q is valid iff $s_0 \sqsubseteq_s Z$. □

Theorem 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 1 carries over directly. We use the same rules for Δ with parallel composition instead of sequential composition. □

Theorem 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 $n \in \mathbf{N}$ 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

$$Q := \exists x_0, x_1, \ldots, x_{n-1}(Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid.

A BPA system Δ with initial state G_0 is defined as follows: $G_i \xrightarrow{g} G_{i+1}.X_i$ and $G_i \xrightarrow{g} G_{i+1}.\bar{X}_i$ for $0 \leq i \leq n-1$ and $G_n \xrightarrow{check} \epsilon$ and $X_i \xrightarrow{q_j} \epsilon$ if X_i is in Q_j and $\bar{X}_i \xrightarrow{q_j} \epsilon$ if $\bar{X}_i$ is in Q_j. Furthermore we add rules $X_i \xrightarrow{d} \epsilon$ and $\bar{X}_i \xrightarrow{d} \epsilon$ for every i. A finite-state system Γ with initial state s is defined as follows: $s \xrightarrow{g} s$ and $s \xrightarrow{check} s_i$ for $1 \leq i \leq k$ and $s_i \xrightarrow{q_j} s_i$ for any i, j with $i \neq j$ and $s_i \xrightarrow{d} s_i$ for every i. If Q is valid then there is an assignment that satisfies all clauses Q_j. Then s cannot simulate G_0, because no s_i can do every action q_j. If Q is not valid then in every assignment some Q_j is not true. Then s can simulate G_0 by going to the state s_j. Thus Q is valid iff $G_0 \not\sqsubseteq_s s$. □

Theorem 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 Theorem 3. The rules for Δ are like in Theorem 3 with parallel composition instead of sequential composition. Γ is defined in the same way, but we also add the rules $s \xrightarrow{q_i} u$ for every $1 \leq i \leq k$, and $u \xrightarrow{a} u$ for every $a \in \{q_1, \ldots, q_k, g, check\}$. Intuitively, if some q_i is emitted before G_0 completes the guess (i.e. before *check* is emitted), s goes to u where it can simulate everything. Again we have that Q is valid iff $G_0 \not\sqsubseteq_s s$. □

Corollary 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 3 and 4. □

The obtained hardness results are of course valid for any superclass of BPA and BPP (in particular for PDA and PN).

4 The Decidability Border

In this section we establish the decidability border of Figure 1. 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[1] (as we do not need this powerful logic in the rest of our paper, we have omitted its definition; we refer to [17] for details).

Then we turn our attention to PA processes. We prove that simulation preorder is *undecidable* between PA processes and finite-state ones in both directions. It is somewhat surprising, as for the subclasses BPP and BPA we have positive decidability results. 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 [18]. However, just one nondeterministic state in PA suffices to make the undecidability proof possible.

Theorem 5. *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

[1] We would like to thank Javier Esparza who observed the idea of the presented proof.

Δ', two processes A, B of Δ', and a formula φ of the 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 $\alpha \in Stack(\Delta)^*$. 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 new bottom symbol Z_0 is added so that F cannot 'get stuck' just due to the emptiness of the stack. Let

$$\varphi \equiv \nu X.(\bigwedge_{a \in Act} [a]\langle a \rangle X)$$

where Act is the set of actions which are used in Δ and Γ (note that Act is finite). The problem whether a PDA process satisfies φ is decidable in *EXPTIME*.
Let P be of the form $p\alpha$. Now 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 2. *Simulation equivalence between PDA and finite-state processes is decidable in EXPTIME.*

Theorem 6. *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 two-counter machine with counters initialized to m_1, m_2. We construct a deterministic PA process $P(\mathcal{M})$ and a deterministic finite-state process $F(\mathcal{M})$ s.t. $P(\mathcal{M}) \sqsubseteq_s F(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt.
Let $Act := \{zero_1, inc_1, dec_1, zero_2, inc_2, dec_2\}$. The PA process $P(\mathcal{M})$ is defined by the following rules:

$$\begin{array}{llll} Z_1 \stackrel{zero_1}{\longrightarrow} Z_1 & Z_1 \stackrel{inc_1}{\longrightarrow} C_1.Z_1 & C_1 \stackrel{inc_1}{\longrightarrow} C_1.C_1 & C_1 \stackrel{dec_1}{\longrightarrow} \epsilon \\ Z_2 \stackrel{zero_2}{\longrightarrow} Z_2 & Z_2 \stackrel{inc_2}{\longrightarrow} C_2.Z_2 & C_2 \stackrel{inc_2}{\longrightarrow} C_2.C_2 & C_2 \stackrel{dec_2}{\longrightarrow} \epsilon \end{array}$$

The initial state is $(C_1^{m_1}.Z_1) \,\|\, (C_2^{m_2}.Z_2)$.

The process $F(\mathcal{M})$ 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 an arc $n \xrightarrow{inc_i} n'$. For every instruction of the form

$$n: \ \texttt{if} \ c_i = 0 \ \texttt{then goto} \ n' \ \texttt{else} \ c_i := c_i - 1; \ \texttt{goto} \ n'' \ \texttt{fi}$$

we have arcs $n \xrightarrow{zero_i} n'$ and $n \xrightarrow{dec_i} n''$. Then we add a new state *all* and arcs $all \xrightarrow{a} all$ for every $a \in Act$. Finally, we complete the process $F(\mathcal{M})$ in the following way: For every node n, except for the one which corresponds to the final state *halt* of $\mathcal{M}$, and every $a \in Act$, if there is no arc $n \xrightarrow{a} n'$ for any n', then add an arc $n \xrightarrow{a} all$. The initial state of $F(\mathcal{M})$ corresponds to the initial state of $\mathcal{M}$.

The state of $P(\mathcal{M})$ corresponds to the contents of the counters of $\mathcal{M}$ and the state of $F(\mathcal{M})$ corresponds to the state of the finite control of $\mathcal{M}$. A round in the simulation game corresponds to a computation step of $\mathcal{M}$.

The only problem is that $P(\mathcal{M})$ may do steps that do not correspond to steps of the counter machine, e.g. $P(\mathcal{M})$ does a step dec_1 when the current state in $F(\mathcal{M})$ expects inc_1. In all these cases the construction of $F(\mathcal{M})$ ensures that $F(\mathcal{M})$ can (and must) respond by a step that ends in the state *all*. After such a step $F(\mathcal{M})$ can simulate anything. It is easy to see that $P(\mathcal{M}) \not\sqsubseteq_s F(\mathcal{M})$ iff $P(\mathcal{M})$ can force $F(\mathcal{M})$ to enter the state *halt* via a sequence of moves which correspond to the correct simulation of $\mathcal{M}$. Thus $P(\mathcal{M}) \sqsubseteq_s F(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt. □

Remark 1. Theorem 8 still holds under the additional condition that both the PA process and the finite-state one are normed. We can make the PA process normed by adding the following rules:

$$\begin{array}{ll} Z_1 \xrightarrow{x_1} \epsilon & C_1 \xrightarrow{x_1} \epsilon \\ Z_2 \xrightarrow{x_2} \epsilon & C_2 \xrightarrow{x_2} \epsilon \end{array}$$

Observe that the resulting process is still deterministic. To make sure that $F(\mathcal{M})$ can simulate the actions x_1, x_2, we add the rules $n \xrightarrow{x_1} all$ and $n \xrightarrow{x_2} all$ for every state n of $F(\mathcal{M})$ (this also includes the rules $all \xrightarrow{x_1} all$ and $all \xrightarrow{x_2} all$). The process $F(\mathcal{M})$ is made normed by introducing a new state *terminated* where no action is enabled, and a rule $all \xrightarrow{x} terminated$. It is easy to see that these new systems $P'(\mathcal{M})$ and $F'(\mathcal{M})$ are deterministic and normed, and still satisfy the property that $P'(\mathcal{M}) \sqsubseteq_s F'(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt.

The halting problem is undecidable even for two-counter machines with counters initialized to zero. The construction of $P(\mathcal{M})$ 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 $F(\mathcal{M}')$ is independent of those input values. Hence we can conclude:

Theorem 7. *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.*

Theorem 8. *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 two-counter machine with counters initialized to m_1, m_2. We construct a deterministic PA process $P(\mathcal{M})$ and a deterministic finite-state system $F(\mathcal{M})$ s.t. $F(\mathcal{M}) \sqsubseteq_s P(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt. Let $Act := \{zero_1, inc_1, dec_1, zero_2, inc_2, dec_2, \tau\}$. For the construction of $P(\mathcal{M})$ we start with the same PA process as in Theorem 6 and extend it by the following rules, which handle all the behaviors that are 'illegal' in a given state of $P(\mathcal{M})$ w.r.t. the counter values it represents.

$$\begin{array}{ll} Z_1 \xrightarrow{dec_1} A_1 & C_1 \xrightarrow{zero_1} A_1 \\ Z_2 \xrightarrow{dec_2} A_2 & C_2 \xrightarrow{zero_2} A_2 \\ A_1 \xrightarrow{a} A_1 & \text{for every } a \in \{zero_1, inc_1, dec_1, \tau\} \\ A_2 \xrightarrow{a} A_2 & \text{for every } a \in \{zero_2, inc_2, dec_2, \tau\} \end{array}$$

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. The initial state is $(C_1^{m_1}.Z_1) \,\|\, (C_2^{m_2}.Z_2)$. Note that $P(\mathcal{M})$ is deterministic; a term that contains both A_1 and A_2 can do the action τ in two different ways, but the result is always the same.

The system $F(\mathcal{M})$ 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 an arc $n \xrightarrow{inc_i} n'$. For every instruction of the form

$$n: \; \texttt{if}\ c_i = 0\ \texttt{then goto}\ n'\ \texttt{else}\ c_i := c_i - 1; \; \texttt{goto}\ n''\ \texttt{fi}$$

we have arcs $n \xrightarrow{zero_i} n'$ and $n \xrightarrow{dec_i} n''$. For the unique final state *halt* of the finite control of $\mathcal{M}$ we add the rule $halt \xrightarrow{\tau} halt$. Note that a reachable state of

$P(\mathcal{M})$ cannot do τ, unless it contains A_1 or A_2. Every step in the simulation game corresponds to a computation step of $\mathcal{M}$. It follows that $F(\mathcal{M}) \not\sqsubseteq_s P(\mathcal{M})$ iff $F(\mathcal{M})$ can reach the state *halt* via a sequence of legal steps that correspond to steps of the counter machine (and do not introduce the symbol A_1 or A_2 in $P(\mathcal{M})$). Thus $F(\mathcal{M}) \sqsubseteq_s P(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt. □

Remark 2. Theorem 8 still holds under the additional condition that both the PA process and the finite-state one are normed. The system $F(\mathcal{M})$ is made normed as follows: We introduce a new state *terminated* where no action is enabled, and rules $n \xrightarrow{x}$ *terminated* for every other state n of $F(\mathcal{M})$. To assure that $P(\mathcal{M})$ can always simulate the action x, we add the rules

$$Z_1 \xrightarrow{x} \epsilon, \quad C_1 \xrightarrow{x} \epsilon, \quad A_1 \xrightarrow{x} \epsilon$$

To make $P(\mathcal{M})$ normed, it now suffices to add the following:

$$Z_2 \xrightarrow{y} \epsilon, \quad C_2 \xrightarrow{y} \epsilon, \quad A_2 \xrightarrow{y} \epsilon$$

It is easy to see that these new processes $P'(\mathcal{M})$ and $F'(\mathcal{M})$ are deterministic and normed, and still satisfy the property that $F'(\mathcal{M}) \sqsubseteq_s P'(\mathcal{M})$ iff the machine $\mathcal{M}$ does not halt.

The proof of the following theorem is the same as of Theorem 7:

Theorem 9. *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 [24]) is *decidable* for such a pair of processes, because it coincides with bisimilarity which is known to be decidable [14]. It is thus interesting that simulation equivalence becomes *undecidable* as soon as we consider PA processes with just one nondeterministic state; this is proved in the following theorem:

Theorem 10. *There is a fixed normed deterministic finite-state process F s.t. the problem whether $P =_s F$ for a given normed PA process P is undecidable.*

Proof: We reduce the second undecidable problem of Theorem 7 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 counter machine as in Theorem 7. 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 $F \stackrel{a}{\rightarrow} \overline{F}$, and P by $P \stackrel{a}{\rightarrow} P'$, $P \stackrel{a}{\rightarrow} \overline{F}$. 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 initial state) where an action can be done in two different ways. □

On the other hand, simulation equivalence remains decidable between deterministic PA and *arbitrary* (possibly nondeterministic) finite-state systems. This is a consequence of a more general result—see the next section.

Remark 3. *All undecidability results which have been proved in this section immediately extend to trace preorder and trace equivalence, because trace preorder and trace equivalence coincide with simulation preorder and simulation equivalence in the class of deterministic processes, respectively.*

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 [24] is easily decidable in polynomial time, as it coincides with regularity w.r.t. bisimilarity which is known to have this property [18].

Theorem 11. *Regularity w.r.t. simulation and trace equivalence is undecidable for normed PA processes.*

Proof: Let $\mathcal{M}$ be an arbitrary Minsky machine. We construct a normed PA process P' s.t. P' is regular w.r.t. simulation (and trace) equivalence iff $\mathcal{M}$ does not halt.

Let $P(\mathcal{M})$ and $F(\mathcal{M})$ be the processes constructed in the proof of Theorem 6, modified in the same way as in Remark 1. P' is given by $P' \stackrel{a}{\rightarrow} P(\mathcal{M})$, $P' \stackrel{a}{\rightarrow} F(\mathcal{M})$ (note that P' is normed). If $\mathcal{M}$ does not halt (i.e., if $P(\mathcal{M}) \sqsubseteq_s F(\mathcal{M})$), then P' is regular w.r.t. simulation and trace equivalence, since $P' =_s F'$ where F' is the finite-state process defined by $F' \stackrel{a}{\rightarrow} F(\mathcal{M})$. To complete the proof, we need to show that if $\mathcal{M}$ halts, then P' 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 $P(\mathcal{M})$. The process $F(\mathcal{M})$ can perform the sequence w, but it has to enter the state *halt* from which there are no transitions (cf. the proof of Theorem 6). In other words, $F(\mathcal{M})$ does not have any trace of the form $w\,v$ where

$v \neq \epsilon$. On the other hand, $P(\mathcal{M})$ can perform any trace of the form $w\, inc_1^n\, dec_1^n$ where $n \in \mathbf{N}$. Suppose there is a finite state system G with k states such that $P' =_t G$. 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 $m \in \mathbf{N}$ (here we use a well-known argument from the theory of finite automata). However, P' does not have this property—each trace of P' which is of the form $a\, w\, v$ where $v \neq \epsilon$ must satisfy the condition that $w\, v$ is a trace of $P(\mathcal{M})$. If we choose $m = length(w) + k + 1$, then obviously $P(\mathcal{M})$ 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 P', and we have a contradiction. □

5 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 [6, 4, 13]. We have even provided some new examples in the previous section.

The next theorem suggests a possible way how to use existing algorithms for 'bisimulation problems' to solve the corresponding 'simulation problems'.

Theorem 12. *For every finitely branching transition systems T_1, T_2 there are finitely branching transition systems $\mathcal{B}(T_1), \mathcal{B}(T_2)$ such that $T_1 =_s \mathcal{B}(T_1)$, $T_2 =_s \mathcal{B}(T_2)$, and $\mathcal{B}(T_1), \mathcal{B}(T_2)$ are simulation equivalent iff they are bisimilar, i.e. $\mathcal{B}(T_1) =_s \mathcal{B}(T_2) \Leftrightarrow \mathcal{B}(T_1) \sim \mathcal{B}(T_2)$.*

Proof: For any finitely branching transition system $T = (S, Act, \rightarrow, r)$ we define the system $\mathcal{B}(T) = (S, Act, \rightarrow', r)$, where $\rightarrow'$ is defined as follows:

$$s \stackrel{a}{\rightarrow}' t \;\text{ iff }\; s \stackrel{a}{\rightarrow} t \text{ and } \forall u \in S : (s \stackrel{a}{\rightarrow} u \wedge t \sqsubseteq_s u) \Longrightarrow u \sqsubseteq_s t$$

In other words, we eliminate from T those transitions $s \stackrel{a}{\rightarrow} t$ for which there is a transition $s \stackrel{a}{\rightarrow} u$ with $t \sqsubseteq_s u$ but $u \not\sqsubseteq_s t$, i.e. we preserve only 'maximal' transitions w.r.t. simulation preorder. Note that if there are some a-transitions from a state s, then at least one of those a-transitions *must* be maximal; here we need the assumption that T is finitely branching.

We prove that $T =_s \mathcal{B}(T)$. The fact $\mathcal{B}(T) \sqsubseteq_s T$ is obvious; for the other direction, let us consider the *greatest* simulation R on T (i.e. $R = \{(s,u) \in S \times S \mid s \sqsubseteq_s u\}$). We prove that T is simulated by $\mathcal{B}(T)$ in R. Clearly $(r,r) \in R$; it remains to show that whenever $(s,u) \in R$ and $s \stackrel{a}{\to} s'$, then there is a transition $u \stackrel{a}{\to}' u'$ of $\mathcal{B}(T)$ with $(s',u') \in R$. As $s \sqsubseteq_s u$, there is at least one a-successor of u which simulates s'. Let u' be the maximal one of those a-successors w.r.t. simulation preorder (see above); then $u \stackrel{a}{\to}' u'$ and $(s',u') \in R$ as required.

Now let $T_1 = (S_1, Act, \to, r_1)$, $T_2 = (S_2, Act, \to, r_2)$ be finitely branching transition systems. We prove that $\mathcal{B}(T_1), \mathcal{B}(T_2)$ are simulation equivalent iff they are bisimilar. As bisimilarity is finer than simulation equivalence, the 'if' part is obvious. For the 'only if' part, we show that the following relation is a bisimulation:

$$R = \{(s,u) \mid s \in S_1, u \in S_2, s =_s u\}$$

It clearly suffices, because $(r_1, r_2) \in R$. Let $(s,u) \in R$. By definition of bisimulation, we must show that for each $s \stackrel{a}{\to} s'$ there is $u \stackrel{a}{\to} u'$ with $(s',u') \in R$ and vice versa (we only show the first part; the proof of the second one is symmetric). Let $s \stackrel{a}{\to} s'$. As $s =_s u$, we also have $s \sqsubseteq_s u$ and hence u must be able to 'match' the move $s \stackrel{a}{\to} s'$ by performing some $u \stackrel{a}{\to} u'$ with $s' \sqsubseteq_s u'$. Now it suffices to show that $u' \sqsubseteq_s s'$. As $s =_s u$, we also have $u \sqsubseteq_s s$ and hence the move $u \stackrel{a}{\to} u'$ must be matched by some $s \stackrel{a}{\to} s''$ with $u' \sqsubseteq_s s''$. To sum up, we have $s' \sqsubseteq_s u' \sqsubseteq_s s''$ and hence $s' \sqsubseteq_s s''$ — but it also means that $s'' \sqsubseteq_s s'$ (by definition of $\mathcal{B}(T_1)$). We obtain $s' \sqsubseteq_s u' \sqsubseteq_s s'' \sqsubseteq_s s'$, hence $u' \sqsubseteq_s s'$ as required. □

The construction of transition system $\mathcal{B}(T)$ from the previous proof is not effective in general. However, it provides us with a very general 'strategy' or a 'proof-method' which can be used to solve certain instances of an undecidable 'simulation problem' provided that the corresponding 'bisimulation problem' is decidable—if we are to decide simulation equivalence between T_1, T_2, we can try to construct $\mathcal{B}(T_1), \mathcal{B}(T_2)$ and decide bisimilarity between them. Similarly, if we are interested whether T is regular w.r.t. simulation equivalence, we can construct $\mathcal{B}(T)$ and check its regularity w.r.t. bisimilarity.

As simulation preorder between finite-state processes is decidable, the system $\mathcal{B}(T)$ can be effectively constructed for any finite-state system T. Moreover, if T is deterministic then $\mathcal{B}(T) = T$. Thus, as a consequence of Theorem 12 we obtain:

Theorem 13. *Simulation equivalence is decidable between deterministic PA processes and (arbitrary) finite-state ones.*

Theorem 12 can also be applied in a nontrivial way. In Appendix A we provide a little 'case-study'. We design a rich subclass of BPP processes where $\mathcal{B}(T)$ is effectively constructible; consequently, simulation equivalence as well as regularity w.r.t. simulation equivalence are decidable in this subclass.

6 Summary and Conclusions

The following table 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 [11] preserves trace equivalence.

	BPA		BPP		PA		PDA		PN	
$\sim$ FS	yes [8]		yes [6]		yes [14]		yes [22]		yes [15]	
reg. $\sim$	yes [4]		yes [13]		?		?		yes [13]	
$\sqsubseteq_s$ FS	**YES**		yes [15]		**NO**		**YES**		yes [15]	
FS $\sqsubseteq_s$	**YES**		yes [15]		**NO**		**YES**		yes [15]	
$=_s$ FS	**YES**		yes [15]		**NO**		**YES**		yes [15]	
reg. $=_s$	?		?		**NO**		?		no [15]	
$\sqsubseteq_t$ FS	yes	yes	yes [15]	yes [15]	**NO**	**NO**	yes	yes	yes [15]	yes [15]
FS $\sqsubseteq_t$	yes	no	yes [15]	yes [15]	**NO**	no	yes	no	yes [15]	yes [15]
$=_t$ FS	yes	no	yes [15]	yes [15]	yes [14]	no	yes	no	yes [15]	yes [15]
reg. $=_t$	yes	no	yes [13]	?	?	no	yes	no	yes [13]	no [15]

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 immediate consequences of the 'classical' ones for language equivalence (see [11]). 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 [11]. Thus, our results bring some new 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 the '?' problems are at least semidecidable, as it is possible to enumerate all finite-state systems and decide equivalence with them.

References

[1] P.A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 253–268. Springer-Verlag, 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 *LNCS*, pages 135–150. Springer-Verlag, 1997.

[4] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 247–262. Springer-Verlag, 1996.

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

[6] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR'93*, volume 715 of *LNCS*, pages 143–157. Springer-Verlag, 1993.

[7] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of LICS'93*. IEEE Computer Society Press, 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. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Proceedings of FCT'95*, volume 965 of *LNCS*, pages 221–232. Springer-Verlag, 1995.

[10] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):353–371, 1994.

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

[12] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of TACS'94*, volume 789 of *LNCS*, pages 454–464. Springer-Verlag, 1994.

[13] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimilarity. In *Proceedings of ICALP'96*, volume 1099 of *LNCS*, pages 478–489. Springer-Verlag, 1996.

[14] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 200–211. Springer-Verlag, 1998.

[15] P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proceedings of CONCUR'95*, volume 962 of *LNCS*, pages 348–362. Springer-Verlag, 1995.

[16] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[17] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[18] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proceedings of FST&TCS'96*, volume 1180 of *LNCS*, pages 111–122. Springer-Verlag, 1996.

[19] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. Technical report TUM-I9830, Institut für Informatik, TU-München, 1998.

[20] R. Mayr. Process rewrite systems. *To appear in Information and Computation.*

[21] M.L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[22] 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.

[23] W. Reisig. *Petri Nets—An Introduction.* Springer-Verlag, 1985.

[24] R.J. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of CONCUR'90*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, 1990.

A The BPP^k Classes

The aim of this appendix is to 'prove' our previous claims about generality and usefulness of Theorem 12. We demonstrate that it can also have nontrivial applications, and nontrivial decidability results can be obtained in this way.

We have chosen the class of BPP processes as the subject of our 'case-study'. It is known that e.g. the simulation equivalence problem is not even semidecidable for BPP processes—see [12], while the bisimulation equivalence problem as well as the regularity problem w.r.t. bisimilarity are decidable [7, 13].

For each $k \in \mathbf{N}$ we design the subclass BPP^k of BPP processes which has the following properties:

- the process $\mathcal{B}(\Delta)$ is effectively constructible for any process Δ of BPP^k; moreover, $\mathcal{B}(\Delta)$ is always a BPP process.
- it is decidable whether a given BPP process Δ belongs to BPP^k.
- the class BPP^{k+1} is strictly greater than BPP^k for every $k \in \mathbf{N}$.

Consequently, the simulation equivalence problem as well as the regularity problem w.r.t. simulation equivalence are *decidable* for processes of $\bigcup_{k=1}^{\infty}$ BPP^k.

First we need to introduce some BPP-specific notation. States of a BPP process Δ are parallel expressions which can be viewed as multisets over $Const(\Delta)$, because the '$\|$' operator is commutative. Hence we can use the standard multiset relations on parallel expressions, writing e.g. $X \in X\|Y$, or $X\|X \subseteq X\|X\|Y$. The set of all parallel expressions over $Const(\Delta)$ is denoted by $Const(\Delta)^{\otimes}$. We let Greek letters $\alpha, \beta, \ldots$ to range over $Const(\Delta)^{\otimes}$.

For each $k \in \mathbf{N}$ we define the function Cut_k on $Const(\Delta)^{\otimes}$ as follows (we assume that $X_1, \ldots, X_n$ are pairwise different):

$$Cut_k(X_1^{k_1}\| \cdots \|X_n^{k_n}) = X_1^{\min\{k_1,k\}}\| \cdots \|X_n^{\min\{k_n,k\}}$$

Intuitively, $Cut_k(\alpha)$ 'cuts' from α exactly that information which is needed to determine the behavior of α within the first k steps.

Furthermore, for each $\alpha \in Const(\Delta)^{\otimes}$ we define the set

$$Context_k(\alpha) = \{\, Cut_k(\gamma) \mid \alpha\|\gamma \text{ is a reachable state of } \Delta\}$$

Note that $Context_k(\alpha)$ is finite and effectively constructible for each $\alpha \in Const(\Delta)^{\otimes}$; here we use the fact that BPP processes form a (proper) subclass of Petri nets [9] and hence we can immediately apply some well-known results from this area. In

this case, we can employ e.g. the 'standard' technique by Karp and Miller [16] to construct the set $Context_k(\alpha)$.

Finally, we say that $X, Y \in Const(\Delta)$, $X \neq Y$ are *concurrent* iff Δ has a reachable state of the form $X \| Y \| \alpha$.

Definition 5. *Let Δ be a BPP process, $X \in Const(\Delta)$. We say that X is* useless *iff whenever $X \| \gamma$ is a reachable state of Δ, then for each basic transition $X \stackrel{a}{\rightarrow} \alpha$ there is $Y \in \gamma$, $X \neq Y$ and a basic transition $Y \stackrel{a}{\rightarrow} \beta$ such that $\alpha \| Y \subseteq X \| \beta$.*

Note that useless constants can be easily recognized; for a given $X \in Const(\Delta)^{\otimes}$ we just construct the set $Context_1(X)$ a check whether each of its elements has the property indicated in the previous definition.

Theorem 14. *Let Δ be a BPP process, $X \in Const(\Delta)$ a useless constant. Let Δ' be a BPP process obtained from Δ by deleting all basic transitions of the form $X \stackrel{a}{\rightarrow} \alpha$ and transforming all basic transitions of the form $Y \stackrel{a}{\rightarrow} \beta \| X^j$, where $X \neq Y$ and $X \notin \beta$, to $Y \stackrel{a}{\rightarrow} \beta$. Then $\Delta =_s \Delta'$.*

Proof: Clearly $\Delta' \sqsubseteq_s \Delta$. To prove $\Delta \sqsubseteq_s \Delta'$, it suffices to show that the relation

$$R = \{(X^i \| \alpha, \beta) \mid i \in \mathbf{N_0},\ X^i \| \alpha \text{ is a reachable state of } \Delta, \alpha \subseteq \beta\}$$

is a simulation. However, it follows directly from the definition of useless variable. □

The previous theorem says that useless constants can be safely and effectively removed. Thus, each process Δ can be transformed to a simulation equivalent process Δ' which does not contain any useless constants. We can observe that this transformation is generally ambiguous—if the process Δ contains two useless constants X, Y, we can remove any of them in the first step. It is possible that e.g. Y is not useless anymore after we remove X, hence the order is significant and different final results can be obtained; however, the set of all possible results is finite and effectively constructible.

Example 1. *Let Δ be the following BPP process:*

$$\Delta = \{\ A \stackrel{a}{\rightarrow} B \| C \| D, \quad B \stackrel{a}{\rightarrow} B, \quad C \stackrel{a}{\rightarrow} C, \quad D \stackrel{b}{\rightarrow} D \| E \| E, \quad E \stackrel{c}{\rightarrow} \epsilon$$
$$C \stackrel{b}{\rightarrow} C, \quad D \stackrel{b}{\rightarrow} D \| E, \qquad \}$$

The variables B and C are useless. If we remove B, we obtain

$$\Delta_1 = \{\ A \stackrel{a}{\rightarrow} C \| D, \quad C \stackrel{a}{\rightarrow} C, \quad D \stackrel{b}{\rightarrow} D \| E \| E, \quad E \stackrel{c}{\rightarrow} \epsilon$$
$$C \stackrel{b}{\rightarrow} C, \quad D \stackrel{b}{\rightarrow} D \| E, \qquad \}$$

and the variable C is not useless anymore. Similarly, if we remove C we get

$$\Delta_2 = \{\ A \stackrel{a}{\to} B\|D, \quad B \stackrel{a}{\to} B, \quad D \stackrel{b}{\to} D\|E\|E, \quad E \stackrel{c}{\to} \epsilon \quad D \stackrel{b}{\to} D\|E, \quad \}$$

and B is no more useless. As we shall see, there is a substantial difference between Δ_1 and Δ_2.

Definition 6. *Let $k \in \mathbf{N}$. The BPP^k class is composed of all BPP processes Δ from which useless constants can be eliminated in such a way that the resulting process Δ' satisfies the following conditions:*

- *whenever $X \stackrel{a}{\to} \alpha$, $X \stackrel{a}{\to} \beta$, $\alpha \neq \beta$, are basic transitions of Δ, then one of the following conditions hold:*
  - $\alpha \subseteq \beta$
  - $\beta \subseteq \alpha$
  - $\alpha\|\gamma \not\sqsubseteq_s^k \beta\|\gamma$ *and* $\beta\|\gamma \not\sqsubseteq_s^k \alpha\|\gamma$ *for all* $\gamma \in Context_k(X)$
- *whenever $X \stackrel{a}{\to} \alpha$, $Y \stackrel{a}{\to} \beta$, $X \neq Y$ are basic transitions of Δ and X, Y are concurrent, we have that $\alpha\|Y\|\gamma \not\sqsubseteq_s^k X\|\beta\|\gamma$ and $X\|\beta\|\gamma \not\sqsubseteq_s^k \alpha\|Y\|\gamma$ for all $\gamma \in Context_k(X\|Y)$*

Theorem 15. *Let Δ be a BPP process, $k \in \mathbf{N}$. It is decidable whether $\Delta \in BPP^k$ and if the answer is positive, the process $\mathcal{B}(\Delta)$ can be effectively defined in BPP syntax.*

Proof: We have already mentioned that useless variables can be effectively eliminated in finitely many ways. The two conditions of Definition 6 can be checked effectively, because the set $Context_k(\alpha)$ is effectively constructible for each $k \in \mathbf{N}$, $\alpha \in Const(\Delta)^{\otimes}$, and $\sqsubseteq_s^k$ is decidable for each $k \in \mathbf{N}$.

If we can eliminate useless variables from Δ in such a way that the resulting process Δ' satisfies the conditions specified in Definition 6, the process $\mathcal{B}(\Delta)$ can be obtained from Δ' just by omitting all transitions $X \stackrel{a}{\to} \alpha$ for which there is a transition $X \stackrel{a}{\to} \beta$ with $\alpha \subseteq \beta$. □

For example, the process Δ of Example 1 is in BPP^1 and $\mathcal{B}(\Delta)$ is easy to construct; first we remove the useless variable C, obtaining the process Δ_2 which clearly satisfies the requirements of Definition 6 for $k = 1$. Now it suffices to remove the basic transition $D \stackrel{a}{\to} D\|E$ and the resulting process is exactly $\mathcal{B}(\Delta)$. However, note that if we remove the useless variable B from Δ, the resulting process Δ_1 does not satisfy the second condition of Definition 6 for *any* $k \in \mathbf{N}$; Δ_1 has

basic transitions $C \stackrel{b}{\rightarrow} C$, $D \stackrel{b}{\rightarrow} D\|E$ and as C, D are concurrent, we should have $C\|D\|\gamma \not\sqsubseteq_s^k C\|D\|E\|\gamma$ for each $\gamma \in \mathit{Context}_k(C\|D)$. However, clearly $C\|D\|\gamma \sqsubseteq_s C\|D\|E\|\gamma$ and hence this requirement cannot be satisfied for any $k \in \mathbf{N}$.

It is easy to see that the class BPP^{k+1} is always strictly greater than BPP^k. However, there are also BPP processes which do not belong to $\bigcup_{k=1}^{\infty} \mathrm{BPP}^k$.

It is of course possible to use more advanced methods and design richer subclasses of BPP where the process $\mathcal{B}(\Delta)$ is effectively constructible. One could also concentrate on more powerful models, e.g. on PA processes or Petri nets, but this is not the aim of our paper. We just want to show that Theorem 12 really provides a solid base for attacking undecidable 'simulation problems'.