Completeness Results for Undecidable Bisimilarity Problems

Jiří Srba [1,2]

BRICS, *Department of Computer Science*
University of Aalborg
Fredrik Bajersvej 7E, 9220 Aalborg, Denmark

Abstract

We establish Σ^1_1-completeness (in the analytical hierarchy) of weak bisimilarity checking for infinite-state processes generated by pushdown automata and parallel pushdown automata. The results imply Σ^1_1-completeness of weak bisimilarity for Petri nets and give a negative answer to the open problem stated by Jančar (CAAP'95): "does the problem of weak bisimilarity for Petri nets belong to Δ^1_1?"

Key words: Weak bisimilarity, undecidability, process algebra.

1 Introduction

Given two (infinite-state) processes, the *equivalence checking problem* is to decide whether the processes are equivalent with regard to some behavioral equivalence. This question has been intensively studied for various classes of infinite-state systems (see e.g. [1,8,12] for overviews). The notion of *bisimulation equivalence* is of particular interest both for the theory and practice.

Strong (and weak) bisimilarity checking of Petri nets (PN) is known to be undecidable [6]. In the case of strong bisimilarity the problem is Π^0_1-complete in the arithmetical hierarchy (see e.g. [5]) and in the weak case it is known to be highly undecidable [5] (i.e. it lies beyond the arithmetical hierarchy).

On the other hand, strong bisimilarity checking of pushdown processes (PDA) remains decidable [11] while e.g. the language equivalence is not. The weak bisimilarity problem for PDA was recently proved to be undecidable [14] and we conjectured that the problem lies beyond the arithmetical hierarchy.

[1] Email: srba@brics.dk

[2] This research was supported in part by the GACR, grant No. 201/03/1161.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In this paper we confirm this conjecture and we strengthen the results of high undecidability of weak bisimilarity for PDA and PN not only to ω-hardness in the arithmetical hierarchy but also to Σ_1^1-hardness (the first level of the analytical hierarchy). In the case of Petri nets our proof generalizes the result of Jančar [5] also in another way: the result is demonstrated for a proper subclass of PN called parallel pushdown processes (PPDA) or also multiset automata.

As for the upper bounds it is easy to observe that the weak bisimilarity problems are contained in Σ_1^1 (see [5]). Hence Σ_1^1-completeness of weak bisimilarity for PDA and PPDA (and PN) is established.

An interesting observation is that PDA, PN and PPDA are not Turing powerful (e.g. reachability remains decidable [9]) but still the weak bisimilarity problems are surprisingly highly undecidable.

These Σ_1^1-lower bounds contrast to other results in the theory. For example (weak) trace equivalence checking of PDA and PN remains Π_1^0-complete (see [5]). On the other hand for the communication-free subclass of PN called basic parallel processes strong bisimilarity is PSPACE-complete [7,13] and weak bisimilarity is very likely to be decidable, while other equivalences including (strong and weak) trace equivalence are undecidable [4]. In fact (strong and weak) trace equivalence is Π_1^0-complete [5]. Similar surprising results are valid also for stateless PDA (called basic process algebra) where strong bisimilarity is decidable in 2-EXPTIME [2] and weak bisimilarity is conjectured to be also decidable, while (strong and weak) trace equivalence is undecidable (the formalism describes exactly the class of context-free languages). Again, the problem of (strong and weak) trace equivalence can be seen to be Π_1^0-complete by using a construction from [5].

2 Basic Definitions

A *labelled transition system* is a triple $(S, \mathcal{A}ct, \longrightarrow)$ where S is a set of *states* (or *processes*), $\mathcal{A}ct$ is a set of *labels* (or *actions*), and $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*, written $\alpha \stackrel{a}{\longrightarrow} \beta$, for $(\alpha, a, \beta) \in \longrightarrow$.

Assume that the set of actions $\mathcal{A}ct$ contains a distinguished *silent action* τ. The *weak transition relation* $\Longrightarrow$ is defined by $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^* \circ \stackrel{a}{\longrightarrow} \circ (\stackrel{\tau}{\longrightarrow})^*$ if $a \in \mathcal{A}ct \smallsetminus \{\tau\}$, and $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^*$ if $a = \tau$.

Let $(S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system. A binary relation $R \subseteq S \times S$ is a *weak bisimulation* iff whenever $(\alpha, \beta) \in R$ then for each $a \in \mathcal{A}ct$: if $\alpha \stackrel{a}{\longrightarrow} \alpha'$ then $\beta \stackrel{a}{\Longrightarrow} \beta'$ for some β' such that $(\alpha', \beta') \in R$; and if $\beta \stackrel{a}{\longrightarrow} \beta'$ then $\alpha \stackrel{a}{\Longrightarrow} \alpha'$ for some α' such that $(\alpha', \beta') \in R$. Processes α and β are *weakly bisimilar* ($\alpha \approx \beta$) iff there is a weak bisimulation R such that $(\alpha, \beta) \in R$.

Weak bisimilarity has an elegant characterization in terms of *bisimulation games*. A bisimulation game on a pair of processes α_1 and α_2 is a two-player game between an 'attacker' and a 'defender'. The game is played in *rounds*.

In each round the players change the *current states* β_1 and β_2 (initially α_1 and α_2) according to the following rule.

(i) The attacker chooses an $i \in \{1,2\}$, $a \in \mathcal{A}ct$ and $\beta'_i \in S$ such that $\beta_i \stackrel{a}{\longrightarrow} \beta'_i$.

(ii) The defender responds by choosing a $\beta'_{3-i} \in S$ such that $\beta_{3-i} \stackrel{a}{\Longrightarrow} \beta'_{3-i}$.

(iii) The states β'_1 and β'_2 become the current states.

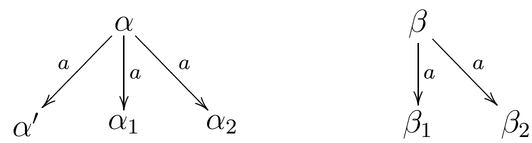
A *play* is a maximal sequence of pairs of states formed by the players according to the rule described above, and starting from the initial states α_1 and α_2. The defender is the winner in every infinite play. A finite play is lost by the player who is stuck.

The following theorem is a standard one (see e.g. [16,17]).

Theorem 2.1 *Processes α_1 and α_2 are weakly bisimilar iff the defender has a winning strategy (and nonbisimilar iff the attacker has a winning strategy).*

In what follows we shall frequently use a technique called 'Defender's Choice' (abbriviated by DC). The idea is that the attacker in the bisimulation game starting from α and β can be forced by the defender to play a certain transition in the following sense: if the attacker takes any other available transition (either from α or β), the defender can always answer in such a way that the resulting processes are weakly bisimilar (and hence the attacker loses).

A typical situation may look like



where $\alpha_i \approx \beta_i$ for $1 \leq i \leq 2$ (very often α_i and β_i will be even syntactically equal). It is easy to see that in the bisimulation game starting from α and β the attacker is forced (DC) to take the transition $\alpha \stackrel{a}{\longrightarrow} \alpha'$. In all other possible moves he immediately loses.

Let $Q = \{p, q, \ldots\}$, $\Gamma = \{X, Y, \ldots\}$ and $\mathcal{A}ct = \{a, b, \ldots\}$ be finite sets of *control states*, *stack symbols* and *actions*, respectively, such that $Q \cap \Gamma = \emptyset$ and $\tau \in \mathcal{A}ct$ is the distinguished *silent action*. A *pushdown automaton* (PDA) is a finite set Δ of rewrite rules of the type $p \stackrel{a}{\longrightarrow} q\alpha$ or $pX \stackrel{a}{\longrightarrow} q\alpha$ where $a \in \mathcal{A}ct$, $p, q \in Q$, $\alpha \in \Gamma^*$ and $X \in \Gamma$.

A pushdown automaton Δ generates a labelled transition system $T(\Delta) \stackrel{\text{def}}{=} (Q \times \Gamma^*, \mathcal{A}ct, \longrightarrow)$ where $Q \times \Gamma^*$ is the set of states [3], $\mathcal{A}ct$ is the set of actions, and the transition relation $\longrightarrow$ is defined by prefix-rewriting rules: $p\gamma \stackrel{a}{\longrightarrow} q\alpha\gamma$ if $(p \stackrel{a}{\longrightarrow} q\alpha) \in \Delta$, and $pX\gamma \stackrel{a}{\longrightarrow} q\alpha\gamma$ if $(pX \stackrel{a}{\longrightarrow} q\alpha) \in \Delta$ for all $\gamma \in \Gamma^*$.

[3] We write $p\alpha$ instead of $(p, \alpha) \in Q \times \Gamma^*$ where p is a control state and α is the stack content. A state $p\epsilon \in Q \times \Gamma^*$, where ϵ stands for the *empty stack*, is written as p.

A *parallel pushdown automaton* (PPDA) is defined in the same way as PDA. The only difference is that the states of the transition system generated by a PPDA system are considered modulo commutativity of the operator for the composition of stack symbols. Hence rather than a sequential access to the stack (as in the case of PDA) we have a parallel access to all the symbols stored in the stack and the stack can be viewed as a multiset of stack symbols.

Example 2.2 Let $\Delta \stackrel{\text{def}}{=} \{pX \stackrel{a}{\longrightarrow} q\}$. For PPDA there is a transition $pYX \stackrel{a}{\longrightarrow} qY$ but there is no such a transition when Δ is interpreted as PDA.

Remark 2.3 For technical convenience (and w.l.o.g.) the rewrite rules for PDA and PPDA were introduced in a slightly more general form than usual. Different definitions use only rules of the form $pX \stackrel{a}{\longrightarrow} q\alpha$. Nevertheless, the rules of the form $p \stackrel{a}{\longrightarrow} q\alpha$ can be converted into this form by standard techniques.

Let $\mathbb{N}_0 \stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots\}$. In what follows we will use the notation A^i for a sequence of $i \in \mathbb{N}_0$ occurrences of $A \in \Gamma$, i.e., $A^0 \stackrel{\text{def}}{=} \epsilon$ and $A^{i+1} \stackrel{\text{def}}{=} A^i A$. By $\#_A(\gamma)$ we denote the number of occurrences of $A \in \Gamma$ in the sequence $\gamma \in \Gamma^*$.

3 High Undecidability of Weak Bisimilarity

In this section we prove that weak bisimilarity checking of PPDA and PDA is Σ^1_1-hard. The proofs are by reduction from the recurrence problem of nondeterministic Minsky machines. We describe first a general idea of the reduction and then show how it applies to PPDA and PDA.

3.1 General Idea

A *nondeterministic Minsky machine* R with two non-negative counters c_1 and c_2 is a finite sequence of *instructions* $R = (I_1,\ I_2,\ \ldots,\ I_n)$ such that $n \geq 1$ and every instruction I_i, $1 \leq i \leq n$, is one of the following three types:

increment	$i:\ c_r := c_r + 1;$ `goto` j
test and decrement	$i:$ `if` $c_r = 0$ `then goto` j `else` $c_r := c_r - 1;$`goto` k
nondet. branching	$i:$ `goto (`j `or` k`)`

where $1 \leq r \leq 2$ and $1 \leq j, k \leq n$.

Remark 3.1 W.l.o.g. we can assume that I_1 is of the type 'increment'.

A *configuration* of R is a triple $(i, v_1, v_2) \in \{1, \ldots, n\} \times \mathbb{N}_0 \times \mathbb{N}_0$ where i is the label of the instruction to be executed, and v_1 and v_2 are the values of the counters c_1 and c_2, respectively. A *computational step* $\hookrightarrow$ between configurations is defined in the natural way.

The following recurrence problem $\mathcal{P}_{rec}$ is Σ^1_1-complete [3]: "given a nondeterministic Minsky machine R, is there an infinite computation of R starting at the instruction label 1 with both counters zero such that the instruction I_1 is executed infinitely many times?"

We reduce the problem $\mathcal{P}_{rec}$ to weak bisimilarity checking of PPDA and PDA. Given an instance P of $\mathcal{P}_{rec}$ we construct a PPDA (PDA) system Δ and a pair of processes p_1 and p'_1 such that the answer to the problem P is yes if and only if $p_1 \approx p'_1$.

The intuition is that a configuration (i, v_1, v_2) corresponds to a pair of processes $p_i\gamma$ and $p'_i\gamma'$ where $\gamma, \gamma' \in \{C_1, C_2, A\}^*$ such that $\#_{C_1}(\gamma) = \#_{C_1}(\gamma') = v_1$, $\#_{C_2}(\gamma) = \#_{C_2}(\gamma') = v_2$, and $\#_A(\gamma) = \#_A(\gamma')$ is the upper bound on the number of steps before the instruction I_1 is executed. In order to check whether γ and γ' contain the same number of occurrences of C_1, C_2 and A we shall introduce the following rules.

$$
\begin{array}{lll}
\mathsf{equal} \xrightarrow{c_1} \mathsf{equal}_{C_1} & \mathsf{equal} \xrightarrow{c_2} \mathsf{equal}_{C_2} & \mathsf{equal} \xrightarrow{a} \mathsf{equal}_{A} \\
\mathsf{equal}_{C_1} C_1 \xrightarrow{c} \mathsf{equal}_{C_1} & \mathsf{equal}_{C_1} C_2 \xrightarrow{\tau} \mathsf{equal}_{C_1} & \mathsf{equal}_{C_1} A \xrightarrow{\tau} \mathsf{equal}_{C_1} \\
\mathsf{equal}_{C_2} C_1 \xrightarrow{\tau} \mathsf{equal}_{C_2} & \mathsf{equal}_{C_2} C_2 \xrightarrow{c} \mathsf{equal}_{C_2} & \mathsf{equal}_{C_2} A \xrightarrow{\tau} \mathsf{equal}_{C_2} \\
\mathsf{equal}_{A} C_1 \xrightarrow{\tau} \mathsf{equal}_{A} & \mathsf{equal}_{A} C_2 \xrightarrow{\tau} \mathsf{equal}_{A} & \mathsf{equal}_{A} A \xrightarrow{c} \mathsf{equal}_{A}
\end{array}
$$

Lemma 3.2 *Let $\gamma, \gamma' \in \{C_1, C_2, A\}^*$. It holds that* $\mathsf{equal}\ \gamma \approx \mathsf{equal}\ \gamma'$ *if and only if $\#_{C_1}(\gamma) = \#_{C_1}(\gamma')$, $\#_{C_2}(\gamma) = \#_{C_2}(\gamma')$ and $\#_A(\gamma) = \#_A(\gamma')$. It is irrelevant whether the rules are interpreted as PPDA or PDA.*

Proof. In the first round the attacker selects a symbol to be tested by performing the action c_1, c_2 or a. In the successive rounds every occurrence of the selected symbol becomes visible under the action c. The τ rules simply remove the remaining symbols (these rules are needed only in the case of PDA). □

Our aim is to design a set of rewrite rules Δ such that both the attacker and the defender have the possibility to force the opponent to faithfully simulate the computation of R.

A single computational step $(i, v_1, v_2) \hookrightarrow (i', v'_1, v'_2)$ of the machine R is simulated by a finite number of rounds in the bisimulation game starting from $p_i\gamma$ and $p'_i\gamma'$ such that $\gamma, \gamma' \in \{C_1, C_2, A\}^*$ where $\#_{C_1}(\gamma) = \#_{C_1}(\gamma') = v_1$, $\#_{C_2}(\gamma) = \#_{C_2}(\gamma') = v_2$ and $\#_A(\gamma) = \#_A(\gamma')$. Such a simulation consists of two phases: a *counting phase* and an *execution phase*.

In the counting phase the players move from p-control states $p_i\gamma$ and $p'_i\gamma'$ to q-control states $q_i\delta$ and $q'_i\delta'$ such that the number of occurrences of the symbol A is altered while the number of occurrences of C_1 and C_2 is preserved. This phase depends on whether $i = 1$ (in this case the defender has the possibility to add an arbitrary number of the symbols A to both γ and γ') or whether $i > 1$ (in this case one occurrence of A is deleted from γ and γ').

In the execution phase starting from the q-control states $q_i\delta$ and $q'_i\delta'$ the players execute the corresponding instruction I_i and modify the number of occurrences of C_1 and C_2 accordingly (hence reaching a new pair of p-control states $p_{i'}\omega$ and $p'_{i'}\omega'$ such that $\#_{C_1}(\omega) = \#_{C_1}(\omega') = v'_1$, $\#_{C_2}(\omega) = \#_{C_2}(\omega') = v'_2$ and $\#_A(\omega) = \#_A(\omega') = \#_A(\delta) = \#_A(\delta')$). In the case of nondeterministic branching the continuation of the game is determined by the defender (using DC).

This concludes the simulation of one computational step of R and the same game repeats starting from $p_{i'}\omega$ and $p'_{i'}\omega'$ (the instruction $I_{i'}$ is going to be executed in this step).

Since the players can force one another to follow the two phases described above, we are able to argue for the correctness of our reduction as follows.

- If there is an infinite computation of R where I_1 is executed infinitely many times then let us fix such a computation. The defender can now force the attacker to simulate this computation in the bisimulation game from p_1 and p'_1 (initially both counters are empty). Moreover the defender is able to add a sufficient number of the symbols A whenever the instruction I_1 is executed and hence it is always possible to delete one occurrence of A in the counting phase of instructions different from I_1. The bisimulation game becomes infinite and hence winning for the defender.
- If there is no infinite computation of R where I_1 occurs infinitely often then the attacker can force the defender to simulate a particular computation (selected by the defender) and after finitely many rounds it is the case that the instruction I_1 cannot be executed from that point (irrelevant of the choices for nondeterministic branching). Now the attacker continues to simulate the computation of R. Every computational step decreases the number of occurrences of the symbol A. Hence after finitely many rounds the attacker wins (all occurrences of A are removed and this can be checked).

3.2 Σ^1_1-Completeness of Weak Bisimilarity for PPDA

We shall now provide the reader with the necessary details of the construction described above. The PPDA rewrite rules are constructed in such a way that they enable a quick adaptation into PDA rules later on.

We start with the counting phase (i.e. moving from p-control states to q-control states). The rules that prepare the execution of I_1 are as follows.

$$p_1 \xrightarrow{a} r_1 \qquad\qquad p'_1 \xrightarrow{a} t'_1$$

$$p_1 \xrightarrow{a} t'_1 \qquad\qquad t'_1 \xrightarrow{\tau} t'_1A \qquad t'_1 \xrightarrow{\tau} r'_1$$

$$r_1 \xrightarrow{a} s_1 \qquad\qquad r'_1 \xrightarrow{a} q'_1$$

$$s_1 \xrightarrow{\tau} s_1A \qquad s_1A \xrightarrow{\tau} s_1 \qquad s_1 \xrightarrow{\tau} q_1 \qquad\qquad r'_1 \xrightarrow{a} s_1$$

$$q_1 \xrightarrow{check} \mathsf{equal} \qquad\qquad q'_1 \xrightarrow{check} \mathsf{equal}$$

Consider a bisimulation game starting from $p_1\gamma$ and $p'_1\gamma'$ for some $\gamma, \gamma' \in \{C_1, C_2, A\}^*$ such that the number of occurrences of C_1, C_2 and A in γ and γ' are equal.

In the first round the attacker is forced to play $p_1\gamma \xrightarrow{a} r_1\gamma$ (DC) and the defender can answer by $p'_1\gamma' \stackrel{a}{\Longrightarrow} r'_1A^{\ell'}\gamma'$ for some $\ell' \in \mathbb{N}_0$ and hence add an arbitrary number of the symbols A. If the defender stays in a state of the form $t'_1A^{\ell'}\gamma'$ the attacker simply continues by using the rule $t'_1 \xrightarrow{\tau} r'_1$ and since there are no τ-moves from $r_1\gamma$ both players can force the other one to reach a pair of states $r_1\gamma$ and $r'_1A^{\ell'}\gamma'$ and it is the defender who chose $\ell' \in \mathbb{N}_0$.

In the next round the attacker is forced to play $r'_1A^{\ell'}\gamma' \xrightarrow{a} q'_1A^{\ell'}\gamma'$ (DC – here the rule $s_1A \xrightarrow{\tau} s_1$ is necessary) and the defender can answer by $r_1\gamma \stackrel{a}{\Longrightarrow} q_1A^{\ell}\gamma$ for some $\ell \in \mathbb{N}_0$ (in fact even some number of symbols A from γ can be deleted but in this case the attacker wins as argued later on in this paragraph). As before, if the defender stays in the s_1-state the attacker uses the rule $s_1 \xrightarrow{\tau} q_1$ and since there are no τ rules out of the state $q'_1A^{\ell'}\gamma'$ the game continues from the pair of q-control states $q_1A^{\ell}\gamma$ and $q'_1A^{\ell'}\gamma'$. If $\ell \neq \ell'$ then the attacker has the possibility to perform the action *check* and he wins because of Lemma 3.2.

This means that after finitely many rounds the players can force the opponent to reach a pair of states $q_1A^{\ell}\gamma$ and $q'_1A^{\ell}\gamma'$ and it is the defender who is allowed to choose the number of occurrences of A.

The following rules decrease the number of occurrences of A by one and prepare the execution of the instructions $I_2, \ldots, I_n$. In the following rules let i range over $\{2, \ldots, n\}$ and let stop be a particular control state from which no transitions are possible.

$$p_iA \xrightarrow{count} q_i \qquad\qquad p'_iA \xrightarrow{count} q'_i$$

$$p_i \xrightarrow{check} \mathsf{stop} \qquad\qquad p'_iA \xrightarrow{check} \mathsf{stop}$$

Consider a bisimulation game starting from $p_i\gamma$ and $p'_i\gamma'$ for some $\gamma, \gamma' \in \{C_1, C_2, A\}^*$ and $1 < i \leq n$ such that the number of occurrences of C_1, C_2 and A in γ and γ' are equal.

If $\#_A(\gamma) = \#_A(\gamma') > 1$ then after one round the players perform the action *count* and reach the pair $q_i\delta$ and $q'_i\delta'$ such that $A\delta = \gamma$ and $A\delta' = \gamma'$ as desired. Should the attacker choose the action *check* the defender wins immediately.

On the other hand if $\#_A(\gamma) = \#_A(\gamma') = 0$ then the attacker wins by using the rule $p_i \xrightarrow{check} \mathsf{stop}$ to which the defender has no answer.

We proceed by the execution phase (i.e. moving from q-control states to p-control states).

For every i, $1 \leq i \leq n$, such that I_i is of the type

$$i:\ c_r := c_r + 1;\ \texttt{goto}\ j$$

we have the following rules.

$$q_i \xrightarrow{inc} p_jC_r \qquad\qquad q'_i \xrightarrow{inc} p'_jC_r$$

In one round of the game starting from $q_i\delta$ and $q'_i\delta'$ the players have only one way to continue and reach the pair $p_jC_r\delta$ and $p'_jC_r\delta'$. Hence they faithfully simulate the corresponding computational step of the machine R.

For every i, $1 \leq i \leq n$, such that I_i is of the type

$$i: \texttt{goto } (j \texttt{ or } k)$$

we have the following rules.

$$q_i \xrightarrow{a} q_i^{choice} \qquad q'_i \xrightarrow{a} q_i^{left}$$
$$q_i \xrightarrow{a} q_i^{left} \qquad q'_i \xrightarrow{a} q_i^{right}$$
$$q_i \xrightarrow{a} q_i^{right}$$

$$q_i^{choice} \xrightarrow{left} p_j \qquad q_i^{left} \xrightarrow{left} p'_j \qquad q_i^{left} \xrightarrow{right} p_k$$
$$q_i^{choice} \xrightarrow{right} p_k \qquad q_i^{right} \xrightarrow{right} p'_k \qquad q_i^{right} \xrightarrow{left} p_j$$

Consider a bisimulation game starting from $q_i\delta$ and $q'_i\delta'$ for some $\delta, \delta' \in \{C_1, C_2, A\}^*$ such that the number of occurrences of C_1, C_2 and A in δ and δ' are equal. We claim that after two rounds of the game the players can force the opponent to reach either $p_j\delta$ and $p'_j\delta'$ or $p_k\delta$ and $p'_k\delta'$ and it is the defender who decides between these two alternatives.

In the first round the attacker is forced to play $q_i\delta \xrightarrow{a} q_i^{choice}\delta$ (DC) and the defender answers by (i) $q'_i\delta' \xrightarrow{a} q_i^{left}\delta'$ or (ii) $q'_i\delta' \xrightarrow{a} q_i^{right}\delta'$. In the second round starting from (i) $q_i^{choice}\delta$ and $q_i^{left}\delta'$ or (ii) $q_i^{choice}\delta$ and $q_i^{right}\delta'$ the attacker is forced (DC) to play the action *left* in case (i) or the action *right* in case (ii). This means that after two rounds the players reach the pair (i) $p_j\delta$ and $p'_j\delta'$ or (ii) $p_k\delta$ and $p'_k\delta'$ according to the defender's choice.

The rules for the 'test and decrement' instructions start with similar rules as those for nondeterministic branching. First, the defender has the choice to determine whether the relevant counter is empty or not and the game continues according to this decision. After the defender's move, the attacker has the possibility to check the correctness of the defender's decision.

Hence for every i, $1 \leq i \leq n$, such that I_i is of the type

$$i: \texttt{if } c_r = 0 \texttt{ then goto } j \texttt{ else } c_r := c_r - 1\texttt{; goto } k$$

we have the following rules.

$$q_i \xrightarrow{a} q_i^{choice} \qquad q'_i \xrightarrow{a} q_i^{left}$$
$$q_i \xrightarrow{a} q_i^{left} \qquad q'_i \xrightarrow{a} q_i^{right}$$
$$q_i \xrightarrow{a} q_i^{right}$$

$$q_i^{choice} \xrightarrow{left} \mathsf{zero}_i \qquad q_i^{left} \xrightarrow{left} \mathsf{zero}'_i \qquad q_i^{left} \xrightarrow{right} \mathsf{nonzero}_i$$
$$q_i^{choice} \xrightarrow{right} \mathsf{nonzero}_i \qquad q_i^{right} \xrightarrow{right} \mathsf{nonzero}'_i \qquad q_i^{right} \xrightarrow{left} \mathsf{zero}_i$$

$$\mathsf{zero}_i \xrightarrow{zero} p_j \qquad \mathsf{zero}'_i \xrightarrow{zero} p'_j$$

$$\mathsf{nonzero}_i\ C_r \xrightarrow{dec} p_k \qquad \mathsf{nonzero}'_i\ C_r \xrightarrow{dec} p'_k$$

$$\mathsf{zero}_i \xrightarrow{check} z_r \qquad \mathsf{zero}'_i \xrightarrow{check} z'_r$$

$$z_r C_r \xrightarrow{b} \mathsf{stop} \qquad z'_r C_r \xrightarrow{c} \mathsf{stop}$$

$$z_r C_{3-r} \xrightarrow{\tau} z_r \qquad z_r A \xrightarrow{\tau} z_r \qquad z'_r C_{3-r} \xrightarrow{\tau} z'_r \qquad z'_r A \xrightarrow{\tau} z'_r$$

$$\mathsf{nonzero}_i \xrightarrow{check} n_r \qquad \mathsf{nonzero}'_i \xrightarrow{check} n'_r$$

$$n_r \xrightarrow{b} \mathsf{stop} \qquad n_r C_r \xrightarrow{c} \mathsf{stop} \qquad n'_r C_r \xrightarrow{b} \mathsf{stop} \qquad n'_r \xrightarrow{c} \mathsf{stop}$$

Consider a bisimulation game starting from $q_i\delta$ and $q'_i\delta'$ for some $\delta, \delta' \in \{C_1, C_2, A\}^*$ such that the number of occurrences of C_1, C_2 and A in δ and δ' are equal.

The same arguments as before apply to show that after two rounds of the game the players can force the opponent to reach the pair (i) $\mathsf{zero}_i\ \delta$ and $\mathsf{zero}'_i\ \delta'$ or (ii) $\mathsf{nonzero}_i\ \delta$ and $\mathsf{nonzero}'_i\ \delta'$ according to the defender's choice.

The attacker can now verify the correctness of the defender's decision by playing the action *check* and thus forcing the defender to reach a pair of states starting with either (i) z_r and z'_r or (ii) n_r and n'_r. The following two lemmas show that in this case the attacker wins if and only if the defender cheated.

Lemma 3.3 *Let $\delta, \delta' \in \{C_1, C_2, A\}^*$ and $r \in \{1, 2\}$. It holds that $z_r\delta \approx z'_r\delta'$ if and only if $0 = \#_{C_r}(\delta) = \#_{C_r}(\delta')$. It is irrelevant whether the rules are interpreted as PPDA or PDA.*

Proof. Obvious. □

Lemma 3.4 *Let $\delta, \delta' \in \{C_1, C_2, A\}^*$ and $r \in \{1, 2\}$. It holds (in the case of PPDA) that $n_r\delta \approx n'_r\delta'$ if and only if $1 \leq \#_{C_r}(\delta), \#_{C_r}(\delta')$. It also holds (in the case of PDA) that $n_r\delta \approx n'_r\delta'$ if and only if both δ and δ' begin with C_r.*

Proof. Obvious. □

In order to finish the simulation of the 'test and decrement' instruction the players have only one continuation of the game from (i) $\mathsf{zero}_i\ \delta$ and $\mathsf{zero}'_i\ \delta'$ or (ii) $\mathsf{nonzero}_i\ \delta$ and $\mathsf{nonzero}'_i\ \delta'$. In case (i) they perform the action *zero* and reach a new pair of states $p_j\delta$ and $p'_j\delta'$. In case (ii) they perform the action *dec* and decrease the number of occurrences of C_r by one. After this they continue from the pair $p_k\omega$ and $p'_k\omega'$ such that $C_r\omega = \delta$ and $C_r\omega' = \delta'$.

In order to conclude the proof recall that both players can force the opponent to faithfully simulate computational steps of the machine R and the defender has the choice in the case of nondeterministic branching. Moreover, whenever the instruction I_1 is performed, the defender can generate enough of the symbols A to ensure that he does not lose before the instruction I_1 is executed again.

If there is a computation of the machine R which visits I_1 infinitely many times, the defender simulates such a computation in the bisimulation game and he wins (either because the attacker decides not to cooperate during the simulation, or because the game becomes infinite). On the other hand, if along every computational path the instruction I_1 is executed only finitely many times, the attacker wins since the symbols A generated by the defender will be exhausted eventually.

Hence the answer to the given recurrence problem of nondeterministic Minsky machines is positive if and only if $p_1 \approx p'_1$.

Theorem 3.5 *Weak bisimilarity checking of PPDA (and PN) is Σ^1_1-complete.*

Proof. The hardness of the problem follows from the construction described above, and the containment of the problem for PN (and PPDA) in Σ^1_1 was established in [5]. □

3.3 Σ^1_1-Completeness of Weak Bisimilarity for PDA

We will now proceed by showing how to adapt the PPDA rules for the case of PDA. Obviously, all the PPDA rules defined above that do not remove any symbol from the stack can be used also for PDA. There are, however, three situations where a symbol is removed from the stack. First, there is the rule $s_1 A \xrightarrow{\tau} s_1$ in the counting phase of the instruction I_1. Since it is sufficient that the rule removes only the occurrences of A added in the previous round (and hence on the top of the stack), no change is needed in this case. The second place where a symbol is removed is in the counting phase of the instructions $I_2, \ldots, I_n$ and the third place are the rules for 'test and decrement' instructions in the case of nonzero value of the corresponding counter (recall that I_1 is of the type 'increment' by Remark 3.1).

Fortunately, the outlined problems can be solved by one modification in the construction. The idea is that before each counting phase, the defender is allowed to rearrange the content of the stacks (while preserving the number of occurrences of C_1, C_2 and A) in such a way that the stacks are equal (in order to apply DC) and the symbol A is on the top of them. If the execution phase continues by a 'test and decrement' instruction and the corresponding counter c_r is nonempty, the defender makes sure that the symbol C_r follows immediately after A.

This is formally described as follows. For all i, $1 < i \leq n$, we remove the PPDA rules for the counting phase (i.e. the rules $p_i A \xrightarrow{count} q_i$, $p'_i A \xrightarrow{count} q'_i$, $p_i \xrightarrow{check} \mathsf{stop}$ and $p'_i A \xrightarrow{check} \mathsf{stop}$) and replace them with the following rules where X ranges over the set $\{C_1, C_2, A\}$.

$$p_i \xrightarrow{a} r_i \qquad\qquad p'_i \xrightarrow{a} t'_i$$
$$p_i \xrightarrow{a} t'_i \qquad\qquad t'_i \xrightarrow{\tau} t'_iX \qquad t'_iX \xrightarrow{\tau} t'_i$$
$$t'_i \xrightarrow{\tau} r'_i$$

$$r_i \xrightarrow{check} \mathsf{equal} \qquad\qquad r'_i \xrightarrow{check} \mathsf{equal}$$

$$r_i \xrightarrow{a} s_i \qquad\qquad r'_i \xrightarrow{a} u'_i$$
$$s_i \xrightarrow{\tau} s_iX \qquad s_iX \xrightarrow{\tau} s_i \qquad r'_i \xrightarrow{a} s_i$$
$$s_i \xrightarrow{\tau} u_i$$

$$u_i \xrightarrow{check} \mathsf{equal} \qquad\qquad u'_i \xrightarrow{check} \mathsf{equal}$$

$$u_iA \xrightarrow{count} q_i \qquad\qquad u'_iA \xrightarrow{count} q'_i$$
$$u_i \xrightarrow{b} \mathsf{stop} \qquad u_iA \xrightarrow{c} \mathsf{stop} \qquad u'_iA \xrightarrow{b} \mathsf{stop} \qquad u'_i \xrightarrow{c} \mathsf{stop}$$

Consider a bisimulation game played from $p_i\gamma$ and $p'_i\gamma'$ such that $\#_{C_1}(\gamma) = \#_{C_1}(\gamma')$, $\#_{C_2}(\gamma) = \#_{C_2}(\gamma')$ and $\#_A(\gamma) = \#_A(\gamma')$. We claim that after two rounds of the game the players can force the opponent to reach a pair of states $u_i\delta$ and $u'_i\delta'$ such that $\#_{C_1}(\delta) = \#_{C_1}(\delta') = \#_{C_1}(\gamma) = \#_{C_1}(\gamma')$, $\#_{C_2}(\delta) = \#_{C_2}(\delta') = \#_{C_2}(\gamma) = \#_{C_2}(\gamma')$ and $\#_A(\delta) = \#_A(\delta') = \#_A(\gamma) = \#_A(\gamma')$ and it is the defender who selects such δ and δ'.

In the first round starting from $p_i\gamma$ and $p'_i\gamma'$ the attacker is forced (DC) to play $p_i\gamma \xrightarrow{a} r_i\gamma$. The defender answers by $p'_i\gamma' \overset{a}{\Longrightarrow} r'_i\delta'$ (should the defender answer only by $p'_i\gamma' \overset{a}{\Longrightarrow} t'_i\delta'$, the attacker plays $t'_i\delta' \xrightarrow{\tau} r'_i\delta'$ and the defender can only respond by staying in $r_i\gamma$).

The game continues by the second round from the states $r_i\gamma$ and $r'_i\delta'$. (If $\#_{C_1}(\gamma) \neq \#_{C_1}(\delta')$, or $\#_{C_2}(\gamma) \neq \#_{C_2}(\delta')$, or $\#_A(\gamma) \neq \#_A(\delta')$ the attacker plays the action *check* and wins from the states $\mathsf{equal}\ \gamma$ and $\mathsf{equal}\ \delta'$ because of Lemma 3.2.)

In the second round the attacker is forced (DC) to play $r'_i\delta' \xrightarrow{a} u'_i\delta'$ and the defender answers by $r_i\gamma \overset{a}{\Longrightarrow} u_i\delta$ (should the defender answer only by $r_i\gamma \overset{a}{\Longrightarrow} s_i\delta$, the attacker plays $s_i\delta \xrightarrow{\tau} u_i\delta$ and the defender can only respond by staying in $u'_i\delta'$).

Again, by performing the action *check* the attacker can validate that the number of occurrences of C_1, C_2 and A in δ and δ' are the same.

Hence after two rounds the players can force the opponent to reach the states $u_i\delta$ and $u'_i\delta'$ such that $\#_{C_1}(\delta) = \#_{C_1}(\delta') = \#_{C_1}(\gamma) = \#_{C_1}(\gamma')$, $\#_{C_2}(\delta) = \#_{C_2}(\delta') = \#_{C_2}(\gamma) = \#_{C_2}(\gamma')$ and $\#_A(\delta) = \#_A(\delta') = \#_A(\gamma) = \#_A(\gamma')$ and it was the defender who rearranged the stack contents. In particular the defender can ensure that the stacks are equal in order to apply DC.

This also means that the defender had the chance to place the symbol A on the top of the stacks δ and δ' and hence the game continues by performing the action *count* as in the case of PPDA. If the defender didn't place the symbol A on the top of both stacks or there were no A's in γ and γ', it is easy to see that the attacker wins by performing either the action b or c.

Moreover, if the instruction I_i is of the type 'test and decrement' and the tested counter c_r is nonempty, the defender was forced to place the symbols C_r on both stacks as the second from the top so that the rewrite rules $\mathsf{nonzero}_i\, C_r \xrightarrow{dec} p_k$ and $\mathsf{nonzero}'_i\, C_r \xrightarrow{dec} p'_k$ are applicable later on. If not then the defender loses because the attacker can use the rules $\mathsf{nonzero}_i \xrightarrow{check} n_r$ and $\mathsf{nonzero}'_i \xrightarrow{check} n'_r$ and he wins because of Lemma 3.4.

To sum up, after the presented modification, the arguments for the correctness of our reduction are valid also for PDA.

Theorem 3.6 *Weak bisimilarity checking of PDA is Σ^1_1-complete.*

Proof. The hardness of the problem was shown above, and the containment of the problem in Σ^1_1 follows from [5]. □

Remark 3.7 A general reduction from weak bisimilarity of PDA to weak bisimilarity of normed (from every reachable configuration it is possible to empty the stack) PDA described in [14] implies that weak bisimilarity of normed PDA is also Σ^1_1-complete. A similar reduction works also for PPDA which are normed in the same sense as PDA.

4 Conclusion

We have proved that weak bisimilarity problems for PDA and PPDA (and also for PN) are Σ^1_1-hard and hence Σ^1_1-completeness of the problems is established.

We believe that the ideas of the presented reductions will find their applications also in other classes of infinite-state systems. A particular challenge is to show high undecidability or even Σ^1_1-completeness of weak bisimilarity checking for process formalisms like PA-processes and one-counter processes. These problems are known to be undecidable [15,10] but their classification in the hierarchy of undecidable problems is open.

Acknowledgment
I would like to thank Moshe Vardi for drawing my attention to the topic, Petr Jančar for many comments and suggestions, and the anonymous referees for their remarks.

References

[1] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, Elsevier Science, 2001 pp. 545–623.

[2] Burkart, O., D. Caucal and B. Steffen, *An elementary decision procedure for arbitrary context-free processes*, in: *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, LNCS **969** (1995), pp. 423–433.

[3] Harel, D., *Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness*, Journal of the ACM (JACM) **33** (1986), pp. 224–248.

[4] Hüttel, H., *Undecidable equivalences for basic parallel processes*, in: *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, LNCS **789** (1994), pp. 454–464.

[5] Jančar, P., *High undecidability of weak bisimilarity for Petri nets*, in: *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*, LNCS **915** (1995), pp. 349–363.

[6] Jančar, P., *Undecidability of bisimilarity for Petri nets and some related problems*, Theoretical Computer Science **148** (1995), pp. 281–301.

[7] Jančar, P., *Strong bisimilarity on basic parallel processes is PSPACE-complete*, in: *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, LNCS (2003), to appear.

[8] Kučera, A. and P. Jančar, *Equivalence-checking with infinite-state systems: Techniques and results*, in: *Proceedings of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'02)*, LNCS **2540** (2002), pp. 41–73.

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

[10] Mayr, R., *Undecidability of weak bisimulation equivalence for 1-counter processes*, in: *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, LNCS **2719** (2003), pp. 570–583.

[11] Sénizergues, G., *Decidability of bisimulation equivalence for equational graphs of finite out-degree*, in: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS'98)* (1998), pp. 120–129.

[12] Srba, J., *Roadmap of infinite results*, Bulletin of the European Association for Theoretical Computer Science (Columns: Concurrency) **78** (2002), pp. 163–175, updated online version: http://www.brics.dk/∼srba/roadmap.

[13] Srba, J., *Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard*, in: *Proceedings of the 19th International Symposium on*

Theoretical Aspects of Computer Science (STACS'02), LNCS **2285** (2002), pp. 535–546.

[14] Srba, J., *Undecidability of weak bisimilarity for pushdown processes*, in: *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, LNCS **2421** (2002), pp. 579–593.

[15] Srba, J., *Undecidability of weak bisimilarity for PA-processes*, in: *Proceedings of the 6th International Conference on Developments in Laguage Theory (DLT'02)*, LNCS **2450** (2003), pp. 197–208.

[16] Stirling, C., *Local model checking games*, in: *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, LNCS **962** (1995), pp. 1–11.

[17] Thomas, W., *On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract)*, in: *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, LNCS **668** (1993), pp. 559–568.