Highly Undecidable Questions for Process Algebras ⋆

Petr Jančar[1] and Jiří Srba[2]

[1] Department of Computer Science, Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava - Poruba, Czech Republic
Petr.Jancar@vsb.cz

[2] **BRICS**⋆⋆, Department of Computer Science, University of Aalborg,
Fredrik Bajersvej 7B, 9220 Aalborg East, Denmark
srba@brics.dk

Abstract. We show Σ_1^1-completeness of weak bisimilarity for PA (process algebra), and of weak simulation preorder/equivalence for PDA (pushdown automata), PA and PN (Petri nets). We also show Π_1^1-hardness of weak ω-trace equivalence for the (sub)classes BPA (basic process algebra) and BPP (basic parallel processes).

1 Introduction

In the area of verification, the possibilities of checking behavioural equivalences and/or preorders of systems are a natural object to study, which includes various decidability and complexity questions. A part of research effort has been aimed at bisimulation equivalence (bisimilarity) and simulation preorder, since these had been recognized as fundamental notions. We are interested in infinite-state systems, for which recent surveys of results have been given, e.g., in [2, 11, 17].

The systems we study can be uniformly defined by means of process rewrite systems (PRS) — see Figure 1 for the PRS-hierarchy from [14]; the second and the third level from the bottom is the focus of our interest. We now provide a selection of some results relevant to our paper (all references can be found in [17]).

(Strong) bisimilarity is already well known to be decidable for the class BPA (basic process algebra, or basic sequential processes), i.e., the class of labelled transition systems generated by left-most derivations of context-free grammars in Greibach normal form; the states correspond to finite sequences of nonterminals which are composed sequentially and only the first one, say X, can be rewritten according to a rule $X \xrightarrow{a} \alpha$ while emitting an action a (so for a state $X\beta$ we have $X\beta \xrightarrow{a} \alpha\beta$).

⋆ Both authors are partly supported by the Grant Agency of the Czech Rep., grant No. 201/03/1161.

⋆⋆ **B**asic **R**esearch in **C**omputer **S**cience,
Centre of the Danish National Research Foundation.

Bisimilarity is also known to be decidable for BPP (basic parallel processes); the only difference with BPA is that nonterminals are viewed as composed in parallel, i.e., each can be rewritten. (We can mention also the recent result [10] showing the decidability for the union of BPA and BPP.) An involved result by Sénizergues (later strengthened and simplified by Stirling) showed the decidability even for PDA – labelled transition systems generated by pushdown automata (where a state (p, α) comprises a control state and a sequence of stack symbols). For PN (labelled place/transition Petri nets) bisimilarity is known to be undecidable; this even holds for the subclass PPDA (pushdown automata with stack symbols composed in parallel), which lies strictly between BPP and PN. For the class PA (where the right-hand sides of grammar rules can contain a mixture of sequential and parallel compositions), the decidability question is still open. *(Strong) simulation preorder* is undecidable (already) for both BPA and BPP – as well as classical language equivalence and its modification called *trace equivalence*.

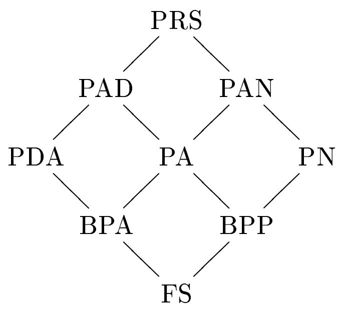

Fig. 1. PRS-hierarchy

We can naturally ask similar questions for models with silent (internal) actions, and explore weak bisimilarity and weak simulation. Decidability of *weak bisimilarity* is still open for both BPA and BPP. From [18] it is known to be highly undecidable for PDA and PN, more precisely, complete for the level Σ_1^1 of the analytical hierarchy (i.e., it can be described by a formula $\exists X.\phi(\ldots, X, \ldots)$ where ϕ is a first-order arithmetical formula containing the predicate X; we refer to [16] for further details about arithmetical and analytical hierarchies). For PA, weak bisimilarity was recently proved undecidable in [19] but the absence of a control unit seemed to prevent a reduction showing Σ_1^1-hardness; so this problem was left open. In fact, such questions might not seem very relevant from the 'practical' point of view, nevertheless we believe that categorizing undecidable problems according to their degrees of undecidability is still useful for deeper understanding of the studied problems. We can also recall the general experience that the 'natural' undecidable problems (in computer science) are either on the lowest levels of the arithmetical hierarchy or on the lowest levels of the analytical hierarchy (see, e.g., [5]).

In this paper we succeeded in modelling a sufficient fragment of the (missing) finite-control unit, which enabled us to show Σ_1^1-completeness of weak bisimilarity also for PA.

We then use some modifications of the developed reductions to show Σ_1^1-completeness of *weak simulation preorder/equivalence* for all the classes PDA, PA and PN (in fact, again even for PPDA).

Weak trace preorder/equivalence is easily shown to be in Π_1^0, i.e., (very) low in the arithmetical hierarchy. This seems to contradict the experience from the strong case (without silent actions) where the complexity increases in the direction: bisimulation – simulation – trace. We give some results indicating that when taking infinite traces (ω-traces) into account, the mentioned 'contradiction' disappears; in particular we show Π^1_1-hardness of *weak ω-trace preorder/equivalence* for both BPA and BPP.

We also show that *weak regularity checking* (checking if a given system is weakly bisimilar to some finite-state one) is 'easier', by which we mean at most hyperarithmetical, for any reasonable process algebra. Finally we add a few observations about Σ^1_1-completeness of *branching bisimilarity* for PDA and PPDA.

The last section presents a short summary of known results for weak equivalences on the studied classes.

2 Basic Definitions

A *labelled transition system* (LTS) 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*; for each $a \in \mathcal{A}ct$, we view $\stackrel{a}{\longrightarrow}$ as a relation on S where $\alpha \stackrel{a}{\longrightarrow} \beta$ iff $(\alpha, a, \beta) \in \longrightarrow$. We assume that $\mathcal{A}ct$ contains a distinguished *silent action* τ. The *weak transition relation* $\Longrightarrow$ is defined by $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^* \circ \stackrel{a}{\longrightarrow} \circ (\stackrel{\tau}{\longrightarrow})^*$ for $a \in \mathcal{A}ct \smallsetminus \{\tau\}$, and $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^*$ for $a = \tau$.

Given $(S, \mathcal{A}ct, \longrightarrow)$, a binary relation $R \subseteq S \times S$ is a *weak simulation* iff for each $(\alpha, \beta) \in R$, $a \in \mathcal{A}ct$, and α' such that $\alpha \stackrel{a}{\longrightarrow} \alpha'$ there is β' such that $\beta \stackrel{a}{\Longrightarrow} \beta'$ and $(\alpha', \beta') \in R$. A *weak bisimulation* is a weak simulation which is a symmetric relation. We say that a process α *is simulated* by a process β, denoted $\alpha \sqsubseteq_s \beta$, if there is a weak simulation containing (α, β). Processes α and β are *simulation equivalent*, denoted $\alpha =_s \beta$, if $\alpha \sqsubseteq_s \beta$ and $\beta \sqsubseteq_s \alpha$. Processes α and β are *weakly bisimilar*, denoted $\alpha \approx \beta$, if there is a weak bisimulation containing (α, β).

We shall use standard game-theoretic characterizations of the introduced notions [21, 20]. A (weak) *bisimulation game* on a pair of processes α_1 and α_2 is a two-player game between 'Attacker' and '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:

1. Attacker chooses $i \in \{1, 2\}$, $a \in \mathcal{A}ct$ and $\beta'_i \in S$ such that $\beta_i \stackrel{a}{\longrightarrow} \beta'_i$.
2. Defender responds by choosing $\beta'_{3-i} \in S$ such that $\beta_{3-i} \stackrel{a}{\Longrightarrow} \beta'_{3-i}$.
3. 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, starting from the initial states α_1 and α_2. Defender is the winner in every infinite play. A finite play is lost by the player who is stuck.

A (weak) *simulation game* is played similarly, the only change is that Attacker is always bound to choose $i = 1$ (thus playing in the "left process" only).

Proposition 1. *It holds that $\alpha_1 \approx \alpha_2$ (resp. $\alpha_1 \sqsubseteq_s \alpha_2$) iff Defender has a winning strategy in the bisimulation (resp. simulation) game starting from α_1 and α_2.*

In other words, $\alpha_1 \not\approx \alpha_2$ (resp. $\alpha_1 \not\sqsubseteq_s \alpha_2$) iff Attacker has a winning strategy.

2.1 PA-processes

Let $\mathcal{C}onst$ be a set of *process constants*. The class of *process expressions* over $\mathcal{C}onst$ is given by $E ::= \epsilon \mid X \mid E.E \mid E\|E$ where 'ϵ' is the *empty process*, X ranges over $\mathcal{C}onst$, '.' is the operator of *sequential composition*, and '$\|$' stands for a *parallel composition*. We do not distinguish between process expressions related by a *structural congruence*, which is the smallest congruence respecting that '.' is associative, '$\|$' is associative and commutative, and 'ϵ' is a unit for '.' and '$\|$'. We shall adopt the convention that the sequential operator binds tighter than the parallel one. Thus, for example, $X.Y\|Z$ means $(X.Y)\|Z$.

A *PA process rewrite system* ($(1,G)$-PRS in the terminology of [14]) Δ is a finite set of *rules* of the form $X \stackrel{a}{\longrightarrow} E$, where $X \in \mathcal{C}onst$, $a \in \mathcal{A}ct$ and E is a process expression. Let us denote the set of actions and the set of process constants that appear in Δ as $\mathcal{A}ct(\Delta)$ and $\mathcal{C}onst(\Delta)$, respectively. (Note that these sets are finite).

A PA system Δ determines a labelled transition system where the process expressions over $\mathcal{C}onst(\Delta)$ are the states and $\mathcal{A}ct(\Delta)$ is the set of labels. The *transition relation* is the least relation satisfying the following SOS rules (recall that '$\|$' is commutative):

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

A process constant $D \in \mathcal{C}onst(\Delta)$ is called a *deadlock* iff Δ contains no rule $D \stackrel{a}{\longrightarrow} E$ for any E. In the usual presentation of PA it is often assumed that Δ contains no deadlocks.

Later on we use the following obvious proposition:

Proposition 2. *Given a PA system, if $E \approx F$ then $E\|\gamma \approx F\|\gamma$ for any γ.*

2.2 PDA, PPDA, BPA and BPP processes

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 *PDA system* (or a pushdown automaton) Δ is a finite set of rewrite rules of the type $p \stackrel{a}{\longrightarrow} q\alpha$ or $pX \stackrel{a}{\longrightarrow} q\alpha$ where $a \in \mathcal{A}ct$, $p, q \in Q$, $X \in \Gamma$ and $\alpha \in \Gamma^*$. Such a PDA system generates a labelled transition system where $Q \times \Gamma^*$ is the set of states[3], $\mathcal{A}ct$ is the set of actions,

[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.

and the transition relation is defined by prefix-rewriting rules: $(p \xrightarrow{a} q\alpha) \in \Delta$ $(\,(pX \xrightarrow{a} q\alpha) \in \Delta\,)$ implies $p\gamma \xrightarrow{a} q\alpha\gamma$ $(\,pX\gamma \xrightarrow{a} q\alpha\gamma\,)$ for all $\gamma \in \Gamma^*$.

A *PPDA system* (a parallel pushdown automaton) is defined in the same way as a PDA system but the composition of stack symbols is now viewed as commutative, i.e., 'parallel'. (So each symbol stored in the stack is directly accessible and the stack can be viewed as a multiset of stack symbols.)

A PDA (resp. PPDA) system is called BPA for *basic process algebra* (resp. BPP for *basic parallel processes*) whenever the set of control states is singleton.

The classes BPA, BPP, PDA and PA correspond directly to the classes from the PRS hierarchy in Figure 1. The class PPDA is positioned strictly between BPP and PN. Hence all the lower bounds we shall prove for PPDA immediately apply also to PN.

2.3 Defender's Choice Technique

In what follows we shall frequently use a technique called 'Defender's Choice' (abbreviated by DC). The idea is that Attacker in the (bi)simulation game starting from α and β can be forced by Defender to play a certain transition in the following sense: if Attacker takes any other available transition, Defender can answer in such a way that the resulting processes are guaranteed to be (bi)similar (and hence Attacker loses).

A typical situation in the case of bisimilarity may look like in Figure 2 part a) where $\alpha_i \approx \beta_i$ for all $i \geq 1$ (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 Attacker is forced (DC) to take the transition $\alpha \xrightarrow{a} \alpha'$. In all other possible moves he loses.

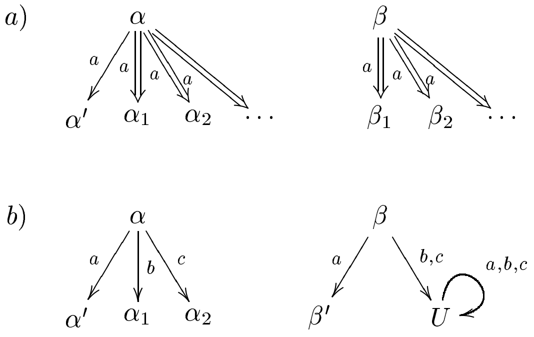

Fig. 2. Defender's Choice

In the case of simulation game, Defender can also use another way to force Attacker to perform a certain move. Defender can threaten to enter a *universal state*, i.e., a state where all available actions are constantly enabled. The situation may look like in Figure 2 part b).

Obviously Attacker who is playing in the left process is forced (DC) to perform the action a to which Defender can answer only by the same action; the players then continue from the pair α' and β'. Should Attacker play b or c in the first round, Defender answers by the same action and enters the universal state U. From now on Defender can answer to all Attacker's moves and clearly wins.

3 Σ_1^1-completeness of weak (bi)similarity problems

From [18] we know that weak bisimilarity is Σ_1^1-complete on PDA and PPDA. For PA only undecidability was known [19] and it was not clear how to simulate "finite-control unit features" which would allow to derive high undecidability as well. Here we answer this question by showing Σ_1^1-completeness also for PA. We then add the Σ_1^1-completeness results for weak simulation preorder (and equivalence) on all the classes PDA, PA and PPDA. Finally we sketch an extension of the results to branching bisimilarity on PDA and PPDA.

We first observe that the mentioned problems are in Σ_1^1: the expression "there exists a set of pairs which contains (P_1, P_2) and is a weak bisimulation (a weak simulation)" can be routinely transformed into a Σ_1^1-formula. For this, it is sufficient that the relations $\xrightarrow{a}$ and $\xRightarrow{a}$ are arithmetical (which is obviously true for any reasonable process algebra like PRS); in fact, these relations are even decidable for the classes PDA, PA and PPDA which we are primarily interested in.

The Σ_1^1-hardness results are achieved by (algorithmic) reductions from suitable problems which are known to be Σ_1^1-complete. One of them is the following:

> Problem: *Recurrent Post's correspondence problem (rPCP)*
> *Instance:* Two sequences $A = [u_1, u_2, \ldots, u_n]$, $B = [v_1, v_2, \ldots, v_n]$ ($n \geq 1$) of nonempty words over an alphabet Σ such that $|u_i| \leq |v_i|$ for all i, $1 \leq i \leq n$.
> *Question:* Is there an infinite sequence of indices $i_1, i_2, i_3, \ldots$ from the set $\{1, 2, \ldots, n\}$ in which the index 1 appears infinitely often and for which the infinite words $u_{i_1} u_{i_2} u_{i_3} \cdots$ and $v_{i_1} v_{i_2} v_{i_3} \cdots$ are equal ?

Such an infinite sequence $i_1, i_2, i_3, \ldots$ is called a *solution* of the instance (A, B). Any finite sequence $i_1, i_2, \ldots, i_m$ is called a *partial solution* of (A, B) iff $u_{i_1} u_{i_2} \cdots u_{i_m}$ is a prefix of $v_{i_1} v_{i_2} \cdots v_{i_m}$.

Remark 1. The problem rPCP is usually defined without the condition $|u_i| \leq |v_i|$; we have included this additional requirement since it is technically convenient and can be easily shown not to affect the following theorem.

Theorem 1. *[5] Problem rPCP is Σ_1^1-complete.*

Let us now fix an instance (A, B) of rPCP, over an alphabet Σ, where $A = [u_1, \ldots, u_n]$ and $B = [v_1, \ldots, v_n]$. A solution of (A, B), if it exists, can be naturally represented by an infinite sequence of process constants from $\{V_1, V_2, \ldots, V_n\}$; the sequence can be divided into finite segments, where a *segment* is defined as a sequence from $\{V_2, V_3, \ldots, V_n\}^* \cdot \{V_1\}$. We note that an infinite sequence composed from segments represents a solution of (A, B) iff all its finite prefixes represent partial solutions, which is equivalent to saying that infinitely many of its finite prefixes represent partial solutions.

A general idea behind our reductions can be described as the following game (which is then concretely implemented in the particular cases we study). Starting from the empty sequence (viewed as a partial solution), Attacker can repeatedly request Defender to prolong the so far constructed partial solution by adding a further segment (for which the implementations will use sequences of τ-moves). Besides the mentioned request, Attacker has also a possibility to enter a checking phase to verify that the (so far) constructed sequence indeed represents a partial solution – if it does not then Attacker wins, and if it does then Defender wins. This means that Defender has a winning strategy if and only if there is an (infinite) solution of the (A, B)-instance.

We now describe a concrete implementation for weak bisimilarity of PA. We show an (algorithmic) construction of a PA system Δ with a pair of processes P_1 and P_2 such that

$$(A, B) \text{ has a solution} \iff P_1 \approx P_2\,. \tag{1}$$

We present Δ in a stepwise manner, always giving a piece of it together with several useful observations (which should make the verification of the desired property straightforward).

In the construction we use a distinguished process constant D which is a *deadlock*, i.e., there are no rules with D on the left-hand side. Particularly useful for us is to note that $\alpha.D.\beta \parallel \gamma \;\approx\; \alpha \parallel \gamma$. Later on we show that using the deadlock is not essential (just technically convenient).

Our first intention is to arrange that the bisimulation game will start from the pair (X, X') and continue through some pairs $(X.\alpha_1, X'.\alpha_1)$, $(X.\alpha_2.\alpha_1, X'.\alpha_2.\alpha_1)$, $(X.\alpha_3.\alpha_2.\alpha_1, X'.\alpha_3.\alpha_2.\alpha_1)$, ... where α_i's are reversed segments which are chosen by Defender (using DC, i.e. Defender's Choice technique). Let us look at the rules in the groups I and II. The bisimulation game starting from $(X.\alpha, X'.\alpha)$ is depicted in Figure 3.

I	$X \xrightarrow{a} X_1'$	$X' \xrightarrow{a} X_1'$	
		$X_1' \xrightarrow{\tau} X_1'.V_i$	for each $i \in \{2, 3, \ldots, n\}$
	$X \xrightarrow{a} Y$	$X_1' \xrightarrow{\tau} Y'.V_1$	
II	$Y \xrightarrow{a} Y_1.D$	$Y' \xrightarrow{a} Y_1.D$	
	$Y_1 \xrightarrow{\tau} Y_1.V_i$		for each $i \in \{2, 3, \ldots, n\}$
	$Y_1 \xrightarrow{\tau} X.V_1$	$Y' \xrightarrow{a} X'$	

According to these rules, when starting from the pair $(X.\alpha, X'.\alpha)$, Attacker is forced (DC) to perform $X.\alpha \xrightarrow{a} Y.\alpha$, otherwise Defender can reach a syntactic equality. Defender can be then viewed as forced to respond by $X' \stackrel{a}{\Longrightarrow} Y'.\beta.\alpha$ for a (reversed) segment β of his choice. If he does not finish by using the rule $X_1' \xrightarrow{\tau} Y'.V_1$, Attacker can perform a move according to this rule in the next round — thus installing a pair $(Y.\alpha, Y'.\beta.\alpha)$ anyway.

Rules in II make clear that Attacker is now forced (DC) to move $Y'.\beta.\alpha \xrightarrow{a} X'.\beta.\alpha$ and Defender can respond by $Y.\alpha \stackrel{a}{\Longrightarrow} X.\beta.\alpha.D.\alpha$; since D is a deadlock, we can view the installed pair as $(X.\beta.\alpha, X'.\beta.\alpha)$. Similarly as above, Defender cannot gain by not using the rule $Y_1 \xrightarrow{\tau} X.V_1$. As we shall see later, he neither can gain by installing $X.\gamma$ for $\gamma \neq \beta.\alpha$.

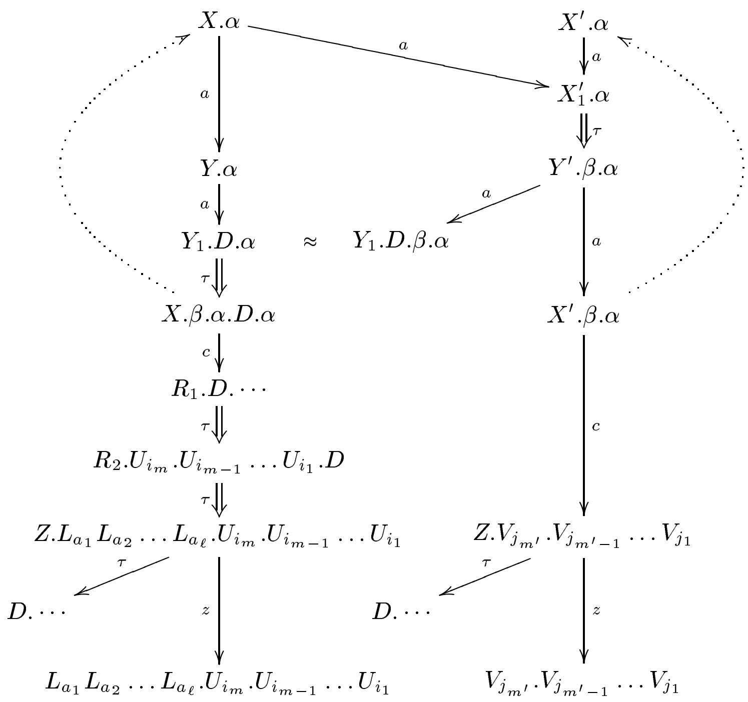

Fig. 3. Generation of a partial solution, assuming that $\beta.\alpha \equiv V_{j_{m'}}.V_{j_{m'-1}}\dots V_{j_1}$

To enable Attacker to enter the checking phase, we add the following rules.

III	$X \xrightarrow{c} R_1.D$	$X' \xrightarrow{c} R_1.D$	
		$X' \xrightarrow{c} Z$	
	$R_1 \xrightarrow{\tau} R_1.U_i$		for all $i \in \{1,2,\dots,n\}$
	$R_1 \xrightarrow{\tau} R_2$		
	$R_2 \xrightarrow{\tau} R_2.L_s$		for all $s \in \Sigma$
	$R_2 \xrightarrow{\tau} Z$		

Having a pair $(X.\alpha, X'.\alpha)$, Attacker can thus also choose to play a c-action (instead of an a-action); in this case he is obviously forced (DC) to play $X'.\alpha \xrightarrow{c} Z.\alpha$. Defender can respond by $X.\alpha \stackrel{c}{\Longrightarrow} Z.\gamma.D.\alpha$ for some $\gamma \in \{L_s \mid s \in \Sigma\}^* \cdot \{U_1, U_2, \dots, U_n\}^*$ where L_s and U_i are new process constants (we recall that Σ is the alphabet of the instance (A, B)). In the whole PA system Δ, there will be only one rule with the action d, namely $Z \xrightarrow{d} Z$ (in group V). By inspecting the rules it is easy to verify that if Defender chooses not to finish his move by using the rule $R_2 \xrightarrow{\tau} Z$, Attacker can play $Z \xrightarrow{d} Z$ in the next round and thus, in fact, force reaching a pair $(Z.\gamma.D.\alpha,\ Z.\alpha)$.

We now want to arrange that the above mentioned Defender's response $(X.\alpha \overset{c}{\Longrightarrow} Z.\gamma.D.\alpha)$ can be successful if and only if $\alpha = V_{i_m}.V_{i_{m-1}}.\dots.V_{i_1}$ represents a partial solution; and in this case the response must be such that $\gamma = L_{a_\ell}.L_{a_{\ell-1}}.\dots.L_{a_1}.U_{i_m}.U_{i_{m-1}}.\dots.U_{i_1}$ where

$$u_{i_1}u_{i_2}\dots u_{i_m}a_1a_2\dots a_\ell = v_{i_1}v_{i_2}\dots v_{i_m}\,. \tag{2}$$

In order to achieve that, we define the set $\mathcal{T} \overset{\text{def}}{=} \{\, T_w \mid w \text{ is a suffix of some } (u_i)^R$ or $(v_i)^R \,\}$ of new process constants (where $(.)^R$ denotes the reversal operation), and we add the following rules.

IV	$U_k \xrightarrow{\iota_k} \epsilon$	$V_k \xrightarrow{\iota_k} \epsilon$	for each $k \in \{1,2,\dots,n\}$
	$U_k \xrightarrow{\tau} T_{(u_k)^R}$	$V_k \xrightarrow{\tau} T_{(v_k)^R}$	for each $k \in \{1,2,\dots,n\}$
	$T_{sw} \xrightarrow{s} T_w$	$T_{sw} \xrightarrow{\tau} T_w$ $T_\epsilon \xrightarrow{\tau} \epsilon$	for $T_{sw} \in \mathcal{T}$ and $s \in \Sigma$
	$L_s \xrightarrow{s} \epsilon$	$L_s \xrightarrow{\tau} \epsilon$	for all $s \in \Sigma$

We can easily verify that a necessary condition for the processes $L_{a_\ell}.L_{a_{\ell-1}}.\dots.L_{a_1}.U_{i_m}.U_{i_{m-1}}.\dots.U_{i_1}$ and $V_{j_{m'}}.V_{j_{m'-1}}.\dots.V_{j_1}$ to be weakly bisimilar is that $m = m'$, $i_1 = j_1, i_2 = j_2, \dots, i_m = j_m$, and (2) holds. But due to the possible mixing of 'letter-actions' and 'index-actions', the condition is not sufficient. That is why the above processes are preceded by Z in our bisimulation game. If Z can be somehow used to implement a 'switch' for Attacker by which he binds himself to checking either only the index-actions or only the letter-actions then our goal is reached.

We first note that the outcomes of such switching can be modeled by composing in parallel either a process constant C_1 (which masks all letter-actions) or C_2 (which masks all index-actions). So we add the rules for C_1, C_2, and also all the rules for Z (whose meaning will become clear later).

V	$C_1 \xrightarrow{s} C_1$	for each $s \in \Sigma$
	$C_2 \xrightarrow{\iota_k} C_2$	for each $k \in \{1,2,\dots,n\}$
	$Z \xrightarrow{z} \epsilon$	
	$Z \xrightarrow{\tau} D$	
	$Z \xrightarrow{d} Z$	

The following propositions are now easy to verify.

Proposition 3. *It holds that* $Z.L_{a_\ell}.L_{a_{\ell-1}}\dots L_{a_1}.U_{i_m}.U_{i_{m-1}}\dots U_{i_1} \parallel C_1 \approx Z.V_{j_{m'}}.V_{j_{m'-1}}\dots V_{j_1} \parallel C_1$ *if and only if* $m = m'$ *and* $i_k = j_k$ *for all* k, $1 \le k \le m$.

Proposition 4. *It holds that* $Z.L_{a_\ell}.L_{a_{\ell-1}}\dots L_{a_1}.U_{i_m}.U_{i_{m-1}}\dots U_{i_1} \parallel C_2 \approx Z.V_{j_{m'}}.V_{j_{m'-1}}\dots V_{j_1} \parallel C_2$ *if and only if* $u_{i_1}u_{i_2}\dots u_{i_m}a_1a_2\dots a_\ell = v_{j_1}v_{j_2}\dots v_{j_{m'}}$.

In order to realize the above discussed 'switch', we add the final group of rules.

VI	$C \xrightarrow{c_1} C_1$	$C \xrightarrow{c_2} C_2$	$C \xrightarrow{z} C \Vert W$
	$W \xrightarrow{\tau} W.U_k$	$W \xrightarrow{\tau} W.V_k$	for each $k \in \{1, 2, \ldots, n\}$
	$W \xrightarrow{\tau} W.L_s$		for all $s \in \Sigma$
	$W \xrightarrow{\tau} \epsilon$		

Now the pair of processes $(X \Vert C,\ X' \Vert C)$ is the pair (P_1, P_2) we were aiming to construct according to equation (1). This is confirmed by the following two lemmas.

Lemma 1. *If the rPCP instance (A, B) has no solution then $X \Vert C \not\approx X' \Vert C$.*

Proof. Assume that (A, B) has no solution. We show a winning strategy for Attacker from the pair $(X \Vert C, X' \Vert C)$.

As described above, Attacker can force the game to go through some pairs $(X \Vert C,\ X' \Vert C)$, $(X.\beta_1 \Vert C,\ X'.\alpha_1 \Vert C)$, $(X.\beta_2 \Vert C,\ X'.\alpha_2.\alpha_1 \Vert C)$, etc., where α_i's are (reversed) segments selected by Defender. (Sequences β_i, also chosen by Defender, play no role in the current analysis.) Since (A, B) has no solution, eventually a pair

$$(X.\beta \Vert C,\ X'.V_{j_{m'}}.V_{j_{m'-1}} \ldots V_{j_1} \Vert C)$$

must be reached where $j_1, j_2, \ldots, j_{m'}$ is *not* a partial solution. Attacker can now force reaching a pair

$$(Z.L_{a_\ell} L_{a_{\ell-1}} \ldots L_{a_1}.U_{i_m}.U_{i_{m-1}} \ldots U_{i_1} \Vert C,\ Z.V_{j_{m'}}.V_{j_{m'-1}} \ldots V_{j_1} \Vert C)$$

(as was described previously, also with help of the rule $Z \xrightarrow{d} Z$).

Now Attacker chooses either $C \xrightarrow{c_1} C_1$ or $C \xrightarrow{c_2} C_2$ (to which Defender has to respond by the same move in the other process) so that the installed pair is not weakly bisimilar (due to Propositions 3 or 4). □

Lemma 2. *If the rPCP instance (A, B) has a solution then $X \Vert C \approx X' \Vert C$.*

Proof. Assuming that (A, B) has a solution, we describe Defender's winning strategy starting from the pair $(X \Vert C, X' \Vert C)$.

First we note that every process reachable from $X \Vert C$ can be naturally viewed as a parallel composition $\alpha \Vert \gamma$ where α is an "X-successor" and γ is a "C-successor"; similarly every process reachable from $X' \Vert C$ can be written as $\alpha' \Vert \gamma'$ where α' is an "X'-successor" and γ' is a "C-successor". Defender's strategy responds to each Attacker's move from the C-successor on one side by the same move (from the C-successor) on the other side, which means that Attacker cannot win by moving only in C-successors (Defender keeps $\gamma = \gamma'$). For the moves in X- and X'-successors our previous observations easily confirm that Defender has a strategy to generate a representation of (an increasing prefix of) an (A, B)-solution, which we assume to exist.

So we have only to examine what happens when Attacker decides to check the (so far) generated partial solution. Attacker has to use (DC) the rule $X' \xrightarrow{c} Z$ to which Defender responds by installing a pair

$$(Z.L_{a_\ell} L_{a_{\ell-1}} \ldots L_{a_1}.U_{i_m}.U_{i_{m-1}} \ldots U_{i_1} \Vert \gamma,\ Z.V_{i_m}.V_{i_{m-1}} \ldots V_{i_1} \Vert \gamma)$$

such that the condition (2) holds. (We omitted the parts starting with D.)

We now observe that γ necessarily contains exactly one of the constants C, C_1 or C_2 (as a parallel component). If C_1 (or C_2) occurs in γ then we reached a weakly bisimilar pair due to Propositions 2 and 3 (or 4).

So it remains to examine the situation $\gamma = C\|\gamma'$. The only interesting case is when Attacker performs $Z.\sigma\|C\|\gamma' \xrightarrow{z} \sigma\|C\|\gamma'$; here σ denotes the relevant sequence (composed from constants U_i and L_s, or from V_i). The other possible moves are safely handled by Defender's playing the same move in the other process.

Defender's response to the above move uses the rules $Z \xrightarrow{\tau} D$, $C \xrightarrow{z} C\|W$ and the (τ-)rules for W so that he performs $Z.\omega\|C\|\gamma' \overset{z}{\Longrightarrow} D.\omega\|C\|\sigma\|\gamma'$; ω denotes the other relevant sequence (composed from U_i and L_s, or from V_i). Hence a weakly bisimilar pair is reached. ◻

Now we state the main theorem, which assumes the usual class PA, i.e., without deadlocks.

Theorem 2. *Weak bisimilarity on PA is Σ_1^1-complete.*

Proof. The membership in Σ_1^1 was already discussed; Σ_1^1-hardness follows from the construction we described and from Lemmas 1 and 2 – on condition that we handle the question of deadlocks. However, there is a straightforward (polynomial-time) reduction from weak bisimilarity of PA with deadlocks to PA without deadlocks (described in [19]). ◻

Combining with the results of [18] (for PDA and PPDA), we can conclude that weak bisimilarity problems for all PRS-classes on the third level of the hierarchy (and above) are Σ_1^1-complete. Using a similar general strategy, we can show the same results also for weak simulation preorder and equivalence:

Theorem 3. *Weak simulation preorder/equivalence on PDA, PA and PPDA is Σ_1^1-complete.*

The constructions are more straightforward in this case, where each player is given a fixed system to play in. Here Defender can influence Attacker's moves by threatening to enter a 'universal' process, which enables all actions forever. Problem rPCP is convenient for reductions in the cases of PDA and PA; in the case of PPDA, the recurrent problem for nondeterministic Minsky machines is more suitable. (It asks whether there is an infinite computation which uses a distinguished instruction infinitely often.) A detailed proof is given in the appendix.

A natural conjecture is now that all relations subsuming weak bisimilarity and being subsumed in weak simulation preorder are also Σ_1^1-hard. Such claims, for general relations $R_1 \subseteq R_2$ are usually proven by reduction (from a suitable problem $\mathcal{P}$) constructing two processes P_1 and P_2 such that $(P_1, P_2) \in R_1$ if the answer (for the instance of $\mathcal{P}$ being reduced) is YES and $(P_1, P_2) \notin R_2$ if the answer is NO.

So far we do not see how to modify our constructions to satisfy this. However, in the case of PDA and PPDA, we could in this way derive Σ_1^1-hardness for all relations between weak bisimilarity and branching bisimilarity. A branching bisimulation (as introduced by van Glabbeek and Weijland, see, e.g., [24]) is a symmetric relation R where, for each $(\alpha, \beta) \in R$, each (Attacker's) move $\alpha \xrightarrow{a} \alpha'$ can be matched by a (Defender's) move $\beta \xrightarrow{\tau}^* \beta_1 \xrightarrow{a} \beta_2 \xrightarrow{\tau}^* \beta'$ where we require $(\alpha', \beta') \in R$ and also $(\alpha, \beta_1) \in R$, $(\alpha', \beta_2) \in R$; Defender's move can be empty in the case $a = \tau$ (then $(\alpha', \beta) \in R$).

Claim. All relations subsuming branching bisimilarity and being subsumed in weak bisimilarity are Σ_1^1-hard on PDA and PPDA.

We do not provide a detailed proof since it would require to repeat the constructions used in [18], with some slight modifications. The point is that the long τ-moves (of Defender) can be made reversible (e.g., for setting a counter value there are τ-actions for both increasing and decreasing). This can be achieved easily in the presence of a finite-control unit (like in case of PDA and PPDA). Such a reversibility is not present in our construction for PA, and it is unclear whether PA processes can model these features in an alternative way.

4 Other semantic equivalences

A natural question to ask is about the complexity of other well-known semantic equivalences (like those in [23] or, more relevantly for us, in [22]). Of particular interest is the question whether some other equivalences are also highly undecidable (i.e., beyond (hyper)arithmetical hierarchy). We provide a few results and notes about this.

For a finite or infinite $w = a_1 a_2 \ldots$ we write $\alpha_0 \stackrel{w}{\Longrightarrow}$ iff there are $\alpha_1, \alpha_2, \ldots$ such that $\alpha_i \stackrel{a_{i+1}}{\Longrightarrow} \alpha_{i+1}$ for all $i = 0, 1, 2, \ldots$. The coarsest equivalence among the studied action-based semantic equivalences is the trace equivalence: two processes α and β are *weakly trace equivalent* iff $\forall w \in (\mathcal{A}ct \smallsetminus \{\tau\})^* : \alpha \stackrel{w}{\Longrightarrow} \Leftrightarrow \beta \stackrel{w}{\Longrightarrow}$ (i.e., α and β enable the same *finite* observable traces).

We can immediately see that the problem is at a very low level in the arithmetical hierarchy even for very general classes of labelled transition systems. We call a labelled transition system (LTS) *recursively enumerable* if the set of states S and the set of actions $\mathcal{A}ct$ are both (represented as) recursively enumerable sets of strings in some finite alphabets and the set $\{ (\alpha, a, \beta) \mid \alpha, \beta \in S, a \in \mathcal{A}ct, \alpha \xrightarrow{a} \beta \}$ is also recursively enumerable. The respective algorithms (Turing machines) can serve as finite descriptions of such an LTS.

We can easily observe that given a recursively enumerable LTS (where $\mathcal{A}ct$ includes τ), the set $\{ (\alpha, w) \mid w \in (\mathcal{A}ct \smallsetminus \{\tau\})^*, \alpha \stackrel{w}{\Longrightarrow} \}$ is also recursively enumerable. More generally, the set of all triples (L, α, w), where L is (a description of) a recursively enumerable LTS, α one of its states and w a finite sequence of its (observable) actions such that $\alpha \stackrel{w}{\Longrightarrow}$ (in L), can be defined by some Σ_1^0-formula $\exists x.\phi(L, \alpha, w, x)$ where ϕ is recursive (with the parameters coded by natural numbers).

Proposition 5. *The set of all triples (L, α, β), where L is (a description of) a recursively enumerable LTS and α, β two weakly trace equivalent states, is in Π^0_2.*

Proof. Having the above mentioned formula $\exists x.\phi(L, \alpha, w, x)$, we can define the formula

$$\psi(L, \alpha, \beta) \Leftrightarrow_{df}$$

$$\forall w.\big(\exists x.\phi(L, \alpha, w, x) \wedge \exists x.\phi(L, \beta, w, x)\big) \vee \big(\forall x.\neg\phi(L, \alpha, w, x) \wedge \forall x.\neg\phi(L, \beta, w, x)\big)$$

which can be easily transformed into the Π^0_2-form. $\square$

Remark 2. In fact, for the classes like PDA, PA and PN the set $\{(L, \alpha, w) \mid \alpha \stackrel{w}{\Longrightarrow}\}$ is even recursive. For PDA and PA this follows, e.g., from [1] and [13] and for PN it can be decided by standard constructions from Petri net theory (reducing to the coverability problem). This means that weak trace equivalence for such classes is in Π^0_1.

For other equivalences based on trace-like finite behaviours (sometimes called 'decorated traces'), i.e., failure equivalence, ready equivalence, ready-trace equivalence etc., we can make similar observations. This means that in fact all these (weak) equivalences are very low in the arithmetical hierarchy.

In some sense, this might seem as a surprising fact. In the strong case (without τ-actions), complexity of the equivalence problems is decreasing in the direction: trace – simulation – bisimulation. On the other hand in the weak case the situation now seems to look different. However, the right way for such a comparison is to take also infinite traces (i.e., ω-traces) into account. Then the above complexity-decreasing chain is restored as illustrated below.

Remark 3. For image-finite labelled transition systems (like those generated by PRS systems in the strong case), the finite-trace equivalence implies also the ω-trace equivalence. This is, however, not true for non-image-finite systems, which are easily generated by PRS systems in the weak case.

We shall focus on the classes BPP and BPA. For BPP weak bisimilarity is known to be semidecidable [3], so it belongs to the class Σ^0_1. In fact, it seems even well possible that the problem is decidable (see [8] where PSPACE-completeness of strong bisimilarity is established). Simulation preorder/equivalence (as well as trace preorder/equivalence) is undecidable even in the strong case [7]. Weak simulation preorder/equivalence is surely in Σ^1_1 (the best estimate we can derive at the moment) while we can prove that weak ω-trace preorder/equivalence is Π^1_1-hard:

Theorem 4. *Weak ω-trace preorder/equivalence on BPP is Π^1_1-hard.*

Given a nondeterministic Minsky machine, the nonexistence of an infinite computation using instruction 1 infinitely often can be reduced to the weak ω-trace preorder (equivalence) problem. In order to prove this we modify a known construction showing undecidability of trace preorder in the strong case (which can be found in [6]). A more detailed sketch of the proof is in the appendix.

For BPA, the situation is roughly similar though a bit more unclear. Both weak bisimilarity and weak similarity are surely in Σ_1^1 but otherwise we only know that weak bisimilarity is EXPTIME-hard [15] and weak similarity undecidable; the latter follows from undecidability of (even) strong similarity [4]. There are some reasons to conjecture that weak bisimilarity of BPA might be decidable. The (obvious) membership in Σ_1^1 thus seems to be a very rough upper bound, and one might start to try to strenghten this by showing that the problem is in the hyperarithmetical hierarchy, i.e., in the intersection of Σ_1^1 and Π_1^1. Nevertheless, it seems that a deeper insight would be needed even for this less ambitious goal.

The undecidability of strong trace equivalence for BPA follows easily from classical results for context-free langauges. Moreover, similarly as in the case of BPP, we can show:

Theorem 5. *Weak ω-trace preorder/equivalence on BPA is Π_1^1-hard.*

The theorem holds even when one BPA-process is a fixed finite-state process. The proof uses the recurrent problem for nondeterministic Turing machines and builds on the classical context-free grammar generating all words which do not correspond to correct computations of a Turing machine (where all even configurations are written in the reverse order). More details are in the appendix.

We also add a straightforward analogy to Proposition 5:

Proposition 6. *The set of all triples (L, α, β), where L is (a description of) a recursively enumerable LTS and α, β two weakly ω-trace equivalent states, is in Π_2^1.*

Proof. An infinite sequence of (observable) actions can be coded as a set W of pairs (i, a) ($i \in \mathbb{N}$, $a \in \mathcal{A}ct$) where each $i \in \mathbb{N}$ appears exactly once; similarly we can code infinite sequences of states. The set of triples (L, α, W), where α enables the ω-trace coded by W, can be then obviously defined by some formula $\exists X.\phi(L, \alpha, W, X)$ where ϕ is surely arithmetical (when X, W are taken as predicates). Now we can take ψ as in the proof of Proposition 5 while replacing the number variables x and w with the set variables X and W, respectively. $\square$

5 Regularity is in the hyperarithmetical hierarchy

Here we look at some more specialized problems, namely the question of equivalence (of a general process) with a given finite-state process, and the question of regularity, which asks whether a given (general) process is equivalent (weakly bisimilar in our case) to an (unspecified) finite-state process. These question were studied, e.g., in [9], from where the next (easy) lemma follows.

To state the lemma, we need to define weak bisimilarity approximants $\approx_i$ ($i = 0, 1, 2, \ldots$):

- $\alpha \approx_0 \beta$ for all α, β ;
- $\alpha \approx_{i+1} \beta$ iff $\alpha \approx_i \beta$ and for all $a \in \mathcal{A}ct$:
  - if $\alpha \stackrel{a}{\Longrightarrow} \alpha'$ then $\beta \stackrel{a}{\Longrightarrow} \beta'$ for some β' such that $\alpha' \approx_i \beta'$, and
  - if $\beta \stackrel{a}{\Longrightarrow} \beta'$ then $\alpha \stackrel{a}{\Longrightarrow} \alpha'$ for some α' such that $\alpha' \approx_i \beta'$.

Lemma 3 ([9]). *Assume that g is a state in a general (infinite) labelled transition system and f is a state in a finite-state system F with k states. Then $g \approx f$ iff*

- *$g \approx_k f$, and*
- *for every g' which is reachable from g there is a state f' in F such that $g' \approx_k f'$.*

Let us now consider recursively enumerable labelled transition systems. In this case the reachability relation as well as the relations $\stackrel{a}{\Longrightarrow}$ are clearly semidecidable.

In such a case, for given g and f from Lemma 3, we can construct a formula ϕ in Π^0_{2k+2} such that ϕ is true iff $g \approx f$. The idea is that $g \approx_k f$ can be shown to be in Π^0_{2k} by writing it as a formula $g \approx_{k-1} f \ \wedge \ \forall a \forall g' \exists f' : \ (g \stackrel{a}{\Longrightarrow} g') \Rightarrow \big(f \stackrel{a}{\Longrightarrow} f' \wedge g' \approx_{k-1} f'\big) \wedge \forall a \forall f' \exists g' : \ (f \stackrel{a}{\Longrightarrow} f') \Rightarrow \big(g \stackrel{a}{\Longrightarrow} g' \wedge g' \approx_{k-1} f'\big)$. From the inductive assumption that $\approx_{k-1}$ is in $\Pi^0_{2(k-1)}$ and from the fact that $\stackrel{a}{\Longrightarrow}$ is semidecidable, the containment in Π^0_{2k} immediately follows. The other condition of Lemma 3 can be expressed by a formula $\forall g' \exists f' : \ (g \longrightarrow^* g') \Rightarrow (g' \approx_k f')$. As shown above, $g' \approx_k f'$ is in Π^0_{2k} and hence the second formula is in Π^0_{2k+2}.

It thus follows easily that the problem $g \approx f$ is recursive in (reducible to) the set *TA* which consists of (the codes of) all true first-order sentences in arithmetic. Since the question of weak regularity of g can be formulated as "is there a number x coding a finite-state system F and its state f such that $g \approx f$?", this problem is recursively enumerable in *TA* (which is taken as oraculum), and hence (at most) hyperarithmetical.

Denoting the collection of all sets which are recursive (recursively enumerable) in *TA* by Σ^0_ω (by $\Sigma^0_{\omega+1}$, respectively), we have shown:

Proposition 7. *The problem of weak regularity of recursively enumerable labelled transition systems is in $\Sigma^0_{\omega+1}$.*

Though the stated result is not too practical, it still separates weak bisimilarity checking from weak regularity checking for the classes like PDA, PA and PPDA (because $\Sigma^0_{\omega+1}$ is a proper subclass of $\Sigma^1_1 \cap \Pi^1_1$). Recalling the general experience that natural problems (in computer science) are either at low levels of the arithmetical hierarchy or at low levels of the analytical hierarchy, we have at least some indication in what direction the results for regularity in the summary table of the next section can be possibly strengthened.

6 Summary

In the following table we provide a summary of the known upper and lower bounds for weak equivalence problems on BPA, BPP, PDA, PA and PPDA/PN. For more references and updated overviews of the results we refer the reader to [17].

weak	BPA	BPP	PDA	PA	PPDA/PN
bisimilarity	in Σ^1_1 EXPTIME-hard	in Σ^0_1 PSPACE-hard	Σ^1_1-complete	Σ^1_1-complete	Σ^1_1-complete
simulation	in Σ^1_1 Π^0_1-hard	in Σ^1_1 Π^0_1-hard	Σ^1_1-complete	Σ^1_1-complete	Σ^1_1-complete
(finite) trace	Π^0_1-complete	Π^0_1-complete	Π^0_1-complete	Π^0_1-complete	Π^0_1-complete
ω-trace	in Π^1_2 Π^1_1-hard	in Π^1_2 Π^1_1-hard	in Π^1_2 Π^1_1-hard	in Π^1_2 Π^1_1-hard	in Π^1_2 Π^1_1-hard
regularity	in $\Sigma^0_{\omega+1}$ EXPTIME-hard	in $\Sigma^0_{\omega+1}$ PSPACE-hard	in $\Sigma^0_{\omega+1}$ EXPTIME-hard	in $\Sigma^0_{\omega+1}$ EXPTIME-hard	in $\Sigma^0_{\omega+1}$ Σ^0_1-hard Π^0_1-hard

References

[1] J.R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.

[2] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[3] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31:13–26, 1997.

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

[5] D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *Journal of the ACM (JACM)*, 33(1):224–248, 1986.

[6] Y. Hirshfeld. Deciding equivalences in simple process algebras. Technical report ECS-LFCS-94-294, Department of Computer Science, University of Edinburgh, 1994.

[7] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *LNCS*, pages 454–464. Springer-Verlag, 1994.

[8] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 218–227. IEEE Computer Society Press, 2003.

[9] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2):409–433, 2001.

[10] P. Jančar, A. Kučera, and F. Moller. Deciding bisimilarity between bpa and bpp processes. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, pages 159–173. Springer-Verlag, 2003.

[11] A. Kučera 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)*, volume 2540 of *LNCS*, pages 41–73. Springer-Verlag, 2002.

[12] H.R. Lewis and Ch.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, 1997.

[13] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on pa-processes. *Theoretical Computer Science*, 274(1–2):89–115, 2002.

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

[15] R. Mayr. Weak bisimilarity and regularity of BPA is EXPTIME-hard. In *Proceedings of the 10th International Workshop on Expressiveness in Concurrency (EXPRESS'03)*, pages 160–143, 2003. To appear in ENTCS.

[16] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.

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

[18] J. Srba. Completeness results for undecidable bisimilarity problems. In *Proceedings of the 5th International Workshop on Verification of Infinite-State Systems (INFINITY'03)*, pages 9–22, 2003. To appear in ENTCS.

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

[20] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11. Springer-Verlag, 1995.

[21] W. Thomas. 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)*, volume 668 of *LNCS*, pages 559–568. Springer-Verlag, 1993.

[22] R.J. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *LNCS*, pages 66–81. Springer-Verlag, 1993.

[23] R.J. van Glabbeek. The linear time - branching time spectrum I: The semantics of concrete, sequential processes. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier Science, 2001.

[24] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

A Appendix

A *nondeterministic Minsky machine* R with two non-negative counters c_1 and c_2 is a finite sequence of *labelled instructions* $R = (1 : I_1,\ 2 : I_2,\ \ldots,\ n : I_n)$ such that $n \geq 1$ and every instruction I_i, $1 \leq i \leq n$, is in one of the following three forms:

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

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

A *configuration* of R is a triple $(i, v_1, v_2) \in \{1, \ldots, n\} \times \mathbb{N} \times \mathbb{N}$ 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.

The following recurrence problem is Σ_1^1-complete [5].

> Problem: *Recurrence Problem for Minsky Counter Machines (rMCM)*
> *Instance:* A nondeterministic Minsky machine R.
> *Question:* 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 often ?

Theorem 3. *Weak simulation preorder/equivalence on PDA, PA and PPDA is Σ_1^1-complete.*

Proof. The membership of the problems in Σ_1^1 was already discussed in the main text. We shall focus on showing Σ_1^1-hardness of weak simulation preorder on PDA, PA and PPDA. By standard techniques weak simulation preorder can be then reduced to weak simulation: let $P \xrightarrow{\tau} P_1$ and $P \xrightarrow{\tau} P_2$, then $P_1 \sqsubseteq_s P_2$ if and only if $P =_s P_2$.

We start by reducing the rPCP problem to weak simulation preorder on PDA processes. We pay a special attention to the fact that this construction should be easily reusable in order to work also for PA. Finally, we establish Σ_1^1-hardness of weak simulation preorder for PPDA by reduction from the rMCM problem.

Let us fix an (A, B)-instance of rPCP. We construct a PDA system Δ and a pair of processes $X.S$ (in fact only a BPA process) and pX' such that $X.S \sqsubseteq_s pX'$ iff the (A, B)-instance has a solution. The aims of the players in the weak simulation game will be the same as in the case of weak bisimilarity (see Section 3). Without loss of generality we may assume that if an infinite solution of the (A, B)-instance exists then it starts with the index 1.

The system Δ is provided in Table 1 (the rules $V_i \xrightarrow{v_i^R} \epsilon$ and $p_2U_i \xrightarrow{u_i^R} p_2$ in fact represent several rewrite rules which enable to perform exactly the whole visible sequences v_i^R and u_i^R respectively; no τ-rules are involved here and $(.)^R$ denotes the reversal operation as before). The rules are divided into Attacker's rules (playing in the left process) and Defender's rules (playing in the right process); the state pU is called a *universal state*, which means that all actions

Attacker's rules	(BPA process)	
$X \xrightarrow{a} Y$		
$Y \xrightarrow{\iota_i} X.V_i$		for all i, $1 \leq i \leq n$
$X \xrightarrow{b} Z$		
$Z \xrightarrow{c_1} \epsilon$	$Z \xrightarrow{c_2} \epsilon$	
$S \xrightarrow{stop} S$		
$V_i \xrightarrow{\iota_i} \epsilon$	$V_i \xrightarrow{v_i^R} \epsilon$	for all i, $1 \leq i \leq n$
Defender's rules	**(PDA process)**	
$pX' \xrightarrow{a} pY_i'$		for all i, $2 \leq i \leq n$
$pX' \xrightarrow{a} pX_1'$		
$pX_1' \xrightarrow{\tau} pX_1'X'$	$pX_1' \xrightarrow{\tau} pY_1'$	
$pY_i' \xrightarrow{\iota_i} p$		for all i, $1 \leq i \leq n$
$pY_i' \xrightarrow{x} pU$		for all $x \in \mathcal{A}ct(\Delta) \smallsetminus \{\iota_i, \tau\}$
$pX' \xrightarrow{\tau} pX_2'S$		
$pX_2' \xrightarrow{\tau} pX_2'U_i$		for all i, $1 \leq i \leq n$
$pX_2' \xrightarrow{\tau} pX_3'$		
$pX_3' \xrightarrow{\tau} pX_3'L_s$		for all $s \in \Sigma$
$pX_3' \xrightarrow{b} p$		
$p \xrightarrow{c_1} p_1$	$p \xrightarrow{c_2} p_2$	
$p_1 \xrightarrow{s} pU$		for all $s \in \Sigma$
$p_2 \xrightarrow{\iota_i} pU$		for all i , $1 \leq i \leq n$
$p_1S \xrightarrow{stop} p_1S$	$p_2S \xrightarrow{stop} p_2S$	
$p_1U_i \xrightarrow{\iota_i} p_1$	$p_2U_i \xrightarrow{u_i^R} p_2$	for all i, $1 \leq i \leq n$
$p_1L_s \xrightarrow{\tau} p_1$	$p_2L_s \xrightarrow{s} p_2$	for all $s \in \Sigma$
$pU \xrightarrow{x} pU$		for all $x \in \mathcal{A}ct(\Delta)$

Table 1. Rewrite rules for weak simulation of PDA

from $\mathcal{A}ct(\Delta) \stackrel{\text{def}}{=} \{a, b, c_1, c_2, stop, \tau\} \cup \{\iota_i \mid 1 \leq i \leq n\} \cup \{s \mid s \in \Sigma\}$ are forever enabled when starting in pU.

The simulation game starts from the processes $X.S$ and pX'. Whenever Attacker's process starts with the process constant X, he can either perform the action b (this is discussed later on) or the action a. In case that he decides for the action a and moves to a process starting with Y, Defender can answer by moving to a state starting with pY_i'. In this move one process constant X' is removed from the stack, apart from the case where Defender enters a state starting with pY_1' because by using the τ-rules he can generate an arbitrary number of process constants X'. (This means that after finitely many rounds Defender is forced to reach a state starting with pY_1', otherwise he removes all constants X' from the stack and loses.) Also note that Defender's answer to Attacker's first move under the action a cannot use the τ-rule $pX' \xrightarrow{\tau} pX_2'S$ because no a-action can be performed after this choice.

So after the first round the players are in the states starting with Y and pY_i'. Attacker is now forced (DC) to play using the rule $Y \xrightarrow{\iota_i} X.V_i$ and Defender can answer only by $pY_i' \xrightarrow{\iota_i} p$. This means that Defender forced Attacker to add the process constant V_i to Attacker's stack. Should Attacker play again the action a, the whole game repeats according to the schema above.

To sum up, Attacker can repeatedly invite Defender to select V_i for some i, $1 \leq i \leq n$, and Defender's selection is remembered on Attacker's stack. Moreover, frequently (i.e. after finitely many steps) Defender has to include the process constant V_1, which concludes the generation of one segment.

From a pair of processes starting with X and pX' Attacker can also perform a move according to the rule $X \xrightarrow{b} Z$ and enter the phase where he checks whether the (so far) generated partial solution is valid. In this case Defender answers by removing the content of the current stack by pushing the unnormed process constant S and continues by generating a sequence of process constants U_i, followed by a sequence of process constants L_s. Finally he performs the action b and the players continue from the processes $Z.V_{i_m}.V_{i_{m-1}}.\cdots.V_{i_1}.S$ and $pL_{a_\ell}L_{a_{\ell-1}}\cdots L_{a_1}U_{j_{m'}}U_{j_{m'-1}}\cdots U_{j_1}S$. The intuition is that Defender should win from this pair if and only if both processes can generate exactly the same sequence of indices and the same sequence of letters from Σ. In the next round Attacker performs the action (i) c_1 or (ii) c_2 in order to verify this property.

In case (i), after performing the action c_1, Attacker is forced to play only the actions corresponding to indices, otherwise Defender threatens (DC) to enter a universal state. None of the players may skip any ι_i action and only after performing all of them, the action *stop* becomes enabled. Hence Defender wins in case (i) if and only if the processes represent the same sequence of indices.

In case (ii), after performing the action c_2, Attacker can only play the actions from Σ, otherwise Defender threatens (DC) to enter a universal state. Defender is first responding by removing the process constants L_s, and then performs the sequence of actions $u^R_{j_{m'}}u^R_{j_{m'-1}}\cdots u^R_{j_1}$ and only after performing all of them, the action *stop* becomes enabled. Defender wins in case (ii) if and only if the processes represent the same sequences over Σ, followed by the action *stop*.

This all together gives that if the rPCP instance has a solution, the game is either infinite or Attacker must (after a certain generated prefix of the infinite solution) perform the action b but then he loses anyway because Defender can make sure that it is a valid partial solution. On the other hand, if the rPCP instance has no solution, after finitely many rounds Defender cannot extend the so far generated partial solution any further and he loses in the checking phase later on. This proves the correctness of the reduction in case of PDA.

We shall now slightly modify the presented construction to show Σ^1_1-hardness of weak simulation also for PA.

Since Attacker's process uses no control states (it is a BPA process), no modification is needed here. Defender's process uses the control state p during the generation phase and later on in the checking phase two extra states p_1 and p_2 are used. We simply keep all the rules containing only the control state p and the rest of the rules is replaced by

$$\begin{array}{lll}
C \xrightarrow{c_1} C_1 & C \xrightarrow{c_2} C_2 & \\
C_1 \xrightarrow{s} U & & \text{for all } s \in \Sigma \\
C_2 \xrightarrow{\iota_i} U & & \text{for all } i,\ 1 \leq i \leq n \\
 & & \\
S \xrightarrow{stop} S & & \\
U_i \xrightarrow{\iota_i} \epsilon & U_i \xrightarrow{u_i^R} \epsilon & \text{for all } i,\ 1 \leq i \leq n \\
L_s \xrightarrow{\tau} \epsilon & L_s \xrightarrow{s} \epsilon & \text{for all } s \in \Sigma.
\end{array}$$

The weak simulation game now starts from the pair X and $X'\|C$. The parallel component C serves as a switch during the checking phase of the game and replaces the control states p_1 and p_2. The correctness of such a modification is easy to verify.

We finish the proof of this theorem by showing that weak simulation preorder of PPDA is also Σ_1^1-complete. In this case we provide a reduction from the recurrence problem of nondeterministic Minsky counter machines (rMCM).

Given an instance R of rMCM we construct a PPDA system Δ and a pair of processes p and p_1' such that the answer to the problem R is yes if and only if $p \sqsubseteq_s p_1'$.

Remark 4. Let A be a process constant and i be a non-negative integer. We shall use the notation A^i for a parallel composition of i occurrences of A, i.e., $A^0 \stackrel{\text{def}}{=} \epsilon$ and $A^{i+1} \stackrel{\text{def}}{=} A\|A^i$.

The intuition is that a configuration (i, v_1, v_2) of R corresponds to a pair of PPDA processes $p(C_1^{v_1}\|C_2^{v_2})$ and $p_i'(C_1^{v_1}\|C_2^{v_2}\|X^\ell)$ where the values of counters are represented by the number of occurrences of C_1 and C_2, and ℓ (the number of occurrences of the process constant X) represents the upper bound on the number of computational steps before the first instruction is executed. Attacker is playing in the process $p(C_1^{v_1}\|C_2^{v_2})$ and he stores only the information about the current counter values. Defender answers in the process $p_i'(C_1^{v_1}\|C_2^{v_2}\|X^\ell)$; apart from the values stored in the counters he also remembers the label i of the instruction to be executed.

In the rules from Table 2 the indices i and r range over the sets $\{1, \ldots, n\}$ and $\{1, 2\}$, respectively. The state sU is the universal state, which means that all actions from $\mathcal{A}ct(\Delta) \stackrel{\text{def}}{=} \{a, b, inc_1, inc_2, dec_1, dec_2, z_1, z_2, \tau\}$ are constantly enabled in sU.

The game starts from the states p and p_1'. During the game the players are simultaneously either in the control states p and p_i', or q and "q_i-like" states, and the numbers of occurrences of C_1 and C_2 on both sides are equal.

Attacker's rules	
$p \xrightarrow{a} q$	
$q \xrightarrow{inc_r} pC_r$	
$qC_r \xrightarrow{dec_r} p$	
$q \xrightarrow{b} p$	
$qC_r \xrightarrow{z_r} qC_r$	
Defender's rules	
$p'_iX \xrightarrow{a} q_j^{inc_r}$	if $I_i \equiv$ $c_r := c_r + 1$; `goto` j
$p'_iX \xrightarrow{a} q_j^{zero_r}$	if $I_i \equiv$ `if` $c_r = 0$ `then goto` j `else` $c_r := c_r - 1$;`goto` k
$p'_iX \xrightarrow{\tau} p_k^{dec_r}$	if $I_i \equiv$ `if` $c_r = 0$ `then goto` j `else` $c_r := c_r - 1$;`goto` k
$p_k^{dec_r}C_r \xrightarrow{a} q_k^{dec_r}$	
$p'_iX \xrightarrow{a} q'_j$	if $I_i \equiv$ `goto` (j `or` k)
$p'_iX \xrightarrow{a} q'_k$	if $I_i \equiv$ `goto` (j `or` k)
$q'_i \xrightarrow{b} p'_i$	
$q'_i \xrightarrow{x} sU$	for all $x \in \mathcal{A}ct(\Delta) \smallsetminus \{b, \tau\}$
$q_i^{inc_r} \xrightarrow{inc_r} p'_iC_r$	
$q_i^{inc_r} \xrightarrow{x} sU$	for all $x \in \mathcal{A}ct(\Delta) \smallsetminus \{inc_r, \tau\}$
$q_i^{dec_r} \xrightarrow{dec_r} p'_i$	
$q_i^{dec_r} \xrightarrow{x} sU$	for all $x \in \mathcal{A}ct(\Delta) \smallsetminus \{dec_r, \tau\}$
$q_i^{zero_r} \xrightarrow{b} p'_i$	
$q_i^{zero_r} \xrightarrow{x} sU$	for all $x \in \mathcal{A}ct(\Delta) \smallsetminus \{b, z_r, \tau\}$
$p'_1 \xrightarrow{\tau} p'_1X$	
$sU \xrightarrow{x} sU$	for all $x \in \mathcal{A}ct(\Delta)$

Table 2. Rewrite rules for weak simulation of PPDA

Let us assume a simulation game starting from a general pair $p(C_1^{v_1} \| C_2^{v_2})$ and $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell)$ where v_1, v_2 and ℓ are non-negative integers.

Attacker can only start by playing according to the rule $p \xrightarrow{a} q$. Should $\ell = 0$ and $i \neq 1$, Defender immediately loses. If $i = 1$, Defender can answer by $\stackrel{a}{\Longrightarrow}$: first using the τ-moves he can generate an arbitrary large number of the process constants X (as parallel components), followed by the execution of an appropriate rule under the visible action a. On the other hand, whenever Defender plays the action a from a process in control state p'_i where $2 \leq i \leq n$, the number of occurrences of X is decreased by one. This ensures that Defender must repeatedly visit the control state p'_1, otherwise he loses.

Let us so analyze the weak simulation game starting from the pair $p(C_1^{v_1} \| C_2^{v_2})$ and $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell)$ such that ℓ is a positive integer and $i \in \{2, 3, \ldots, n\}$. In the first round, Attacker has only one possible move, namely $p(C_1^{v_1} \| C_2^{v_2}) \xrightarrow{a}$

$q(C_1^{v_1} \| C_2^{v_2})$ to which Defender can answer according to the type of the instruction I_i.

- If $I_i \equiv$ $c_r := c_r + 1$; `goto` j then Defender has to play $p_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \xrightarrow{a} q_j^{inc_r}(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$. In the second round starting from $q(C_1^{v_1} \| C_2^{v_2})$ and $q_j^{inc_r}(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$ Attacker is forced (DC) to play the move $q(C_1^{v_1} \| C_2^{v_2}) \xrightarrow{inc_r} p(C_r^{v_r+1} \| C_{3-r}^{v_{3-r}})$, otherwise Defender threatens to enter the universal state sU. Defender can answer only by $q_j^{inc_r}(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1}) \xrightarrow{inc_r} p'_j(C_r^{v_r+1} \| C_{3-r}^{v_{3-r}} \| X^{\ell-1})$.
- If $I_i \equiv$ `if` $c_r = 0$ `then goto` j `else` $c_r := c_r - 1$; `goto` k then Defender has two possibilities:
  - If it is the case that $v_r \geq 1$ then Defender can play $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \overset{a}{\Longrightarrow} q_k^{dec_r}(C_r^{v_r-1} \| C_{3-r}^{v_{3-r}} \| X^{\ell-1})$. In the next round Attacker is forced (DC) to play $q(C_1^{v_1} \| C_2^{v_2}) \xrightarrow{dec_r} p(C_r^{v_r-1} \| C_{3-r}^{v_{3-r}})$ and Defender can only answer by the move $q_k^{dec_r}(C_r^{v_r-1} \| C_{3-r}^{v_{3-r}} \| X^{\ell-1}) \xrightarrow{dec_r} p'_k(C_r^{v_r-1} \| C_{3-r}^{v_{3-r}} \| X^{\ell-1})$.
    Another possibility for Defender in the first move (while still assuming that $v_r \geq 1$) is to answer by $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \xrightarrow{a} q_j^{zero_r}(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$. This does not correspond to a faithful simulation of the Minsky machine R and Attacker can punish such a move by playing $q(C_1^{v_1} \| C_2^{v_2}) \xrightarrow{z_r} q(C_1^{v_1} \| C_2^{v_2})$ to which Defender has no answer.
  - If it is the case that $v_r = 0$, the only possible answer for Defender is $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \xrightarrow{a} q_j^{zero_r}(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$. In the next round Attacker is forced (DC) to take the action b, Defender has only one answer to this move and hence the players necessarily continue from the pair $p(C_1^{v_1} \| C_2^{v_2})$ and $p'_j(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$.
- If $I_i \equiv$ `goto` (j `or` k) then Defender can choose either $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \xrightarrow{a} q'_j(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$ or $p'_i(C_1^{v_1} \| C_2^{v_2} \| X^\ell) \xrightarrow{a} q'_k(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$. In the next round Attacker is forced (DC) to play $q(C_1^{v_1} \| C_2^{v_2}) \xrightarrow{b} p(C_1^{v_1} \| C_2^{v_2})$ and Defender answers either by playing $q'_j(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1}) \xrightarrow{b} p'_j(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$ or by $q'_k(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1}) \xrightarrow{b} p'_k(C_1^{v_1} \| C_2^{v_2} \| X^{\ell-1})$. Hence Defender decided whether the game continues from the label j or k.

The situation is completely similar when the game starts from $p(C_1^{v_1} \| C_2^{v_2})$ and $p'_1(C_1^{v_1} \| C_2^{v_2} \| X^\ell)$ for a non-negative integer ℓ, apart from the fact that Defender can add an arbitrary large number of process constants X in the first round.

To sum up, the players can force one another to faithfully simulate one particular computation of the Minsky machine R and Defender decides all the nondeterministic choices. Because of the process constant X, Defender is forced during every infinite game to reach the control state p'_1 infinitely often, otherwise he loses.

It is now easy to see that the Minsky machine R has an infinite computation where the first instruction is executed infinitely many times if and only if Defender has a winning strategy in the weak simulation game from p and p'_1. $\square$

Theorem 4. *Weak ω-trace preorder/equivalence on BPP is Π^1_1-hard.*

Proof. It is enough to show the result for ω-trace preorder. This immediately implies the hardness also for the equivalence as discussed before.

The idea is to start from an instance R of rMCM (recurrent Minsky counter machine), and to construct two BPP processes α and β such that if there is an infinite computation of R (starting with zeros in the counters c_1 and c_2) which uses the instruction 1 infinitely often then α has an ω-trace (corresponding to that computation) which β does not have; otherwise all ω-traces (as well as finite traces) of α are contained in the (ω-)trace set of β.

Hirshfeld [6] describes in detail, given a deterministic Minsky machine R', how to construct BPP-processes α and β (which model R' in a sense). Process α has a (corresponding) trace for each prefix of the (correct) computation of R' and also some other ('incorrect') traces; if the computation halts, α can finish the corresponding trace with a special 'halting' action. Process β has also such a corresponding trace for each prefix of the (correct) computation (but without the halting action), and *all* 'incorrect' traces.

In fact, the construction can be directly applied also to our nondeterministic machine R (omitting the mentioned halting action). We just add the following modification into the construction: we model a third counter c_3 in β, which can be in the beginning set to any (finite) number by a sequence of τ-moves. We also arrange that after every execution of the instruction 1 the counter c_3 is necessarily decreased by one. This can be done by a special action which (the appropriate derivative of) α can perform in the relevant moment.

So if R has an infinite computation performing instruction 1 infinitely often then the corresponding ω-trace is performable from α but not from β. If there is no such computation, then each (ω-)trace of α is either incorrect (i.e., has a prefix with an 'error') – and so it is also performable from β (via the universal state) – or it is correct but contains the instruction 1 only finitely many times – and so it is performable from β as well. $\square$

Theorem 5. *Weak ω-trace preorder/equivalence on BPA is Π^1_1-hard.*

Proof. We shall adapt a classical idea from language theory used in showing the undecidability of the universality problem for context-free grammars (as can be found, e.g., in [12]).

Let $\mathcal{A}ct \stackrel{\text{def}}{=} \{\rightarrow, \leftarrow, a, b, 0, 1, \#, \tau\}$. One process in our preorder/equivalence checking will be the 'universal' one-state process U :

$$U \xrightarrow{x} U \text{ for all } x \in \mathcal{A}ct.$$

Given now a nondeterministic Turing machine with alphabet $\{a, b\}$, the initial state denoted 0 and additional k control states denoted $1, 11, 111, \ldots, 1^k$, each configuration C can be naturally represented as a word $u0v$ or $u1^iv$ where $u, v \in \{a, b\}^*$ and $1 \leq i \leq k$. Each (infinite) computation can be then described as a sequence $\# \rightarrow C_1\# \leftarrow (C_2)^R\# \rightarrow C_3\# \leftarrow (C_4)^R \ldots$ of configurations where every even configuration is written in the reverse order; the arrow preceding a configuration indicates if this is odd or even.

Following the idea of a context-free grammar (in Greibach normal form) generating all incorrect descriptions of computations, we can construct a process constant F and the appropriate rules so that the following holds:

- each action sequence enabled from F is of the form $\#w$ where w contains at most one $\#$
- $F \stackrel{u}{\longrightarrow} \epsilon$ exactly for the sequences u of the form $\#u_1\#u_2$ which witness an 'error', i.e.:

  there is no v s.t. $\#u_1\#u_2v = \# \to C\# \leftarrow (C')^R$ or $\#u_1\#u_2v = \# \leftarrow (C)^R\# \to C'$ for some (valid) configurations C and C' such that C' is a successor of C.

We also construct F_0 which is a (simple) variant of F where each such sequence is bound to start with $\# \to 0\#$. Let us now define the following BPA rewrite rules (using actions from $\mathcal{A}ct$ and constants U and F, F_0 from above).

$$
\begin{array}{llll}
X \stackrel{\tau}{\longrightarrow} F_0.U & X \stackrel{\#}{\longrightarrow} X_1 & X \stackrel{x}{\longrightarrow} U & \text{for all } x \in \mathcal{A}ct \smallsetminus \{\#, \tau\} \\
 & X_1 \stackrel{\to}{\longrightarrow} X_2 & X_1 \stackrel{x}{\longrightarrow} U & \text{for all } x \in \mathcal{A}ct \smallsetminus \{\to, \tau\} \\
X_2 \stackrel{0}{\longrightarrow} F.U & X_2 \stackrel{0}{\longrightarrow} X_3 & X_2 \stackrel{x}{\longrightarrow} U & \text{for all } x \in \mathcal{A}ct \smallsetminus \{0, \tau\} \\
 & X_3 \stackrel{\#}{\longrightarrow} X'.F.U & X_3 \stackrel{x}{\longrightarrow} U & \text{for all } x \in \mathcal{A}ct \smallsetminus \{\#, \tau\} \\
 & X' \stackrel{\tau}{\longrightarrow} X'.Y & X' \stackrel{\tau}{\longrightarrow} \epsilon & \\
 & Y \stackrel{x}{\longrightarrow} \epsilon & \text{for all } x \in \mathcal{A}ct & \\
 & Y \stackrel{x}{\longrightarrow} Y & \text{for all } x \in \mathcal{A}ct \smallsetminus \{0\} &
\end{array}
$$

We now compare the processes U and X. It can be easily verified that X enables all ω-traces which either do not start with $\# \to 0\#$ or contain only finitely many occurrences of 0 or contain an 'error', i.e., do not correspond to the correct description of an infinite computation of the given nondeterministic Turing machine. So X enables all ω-traces unless there is an infinite computation, starting with the blank tape, which visits the initial state infinitely often. This question is, however, Σ_1^1-complete [5]; therefore both weak ω-trace preorder and equivalence between a fixed (universal) finite-state process and a BPA process are Π_1^1-hard. $\square$