Refining the Undecidability Border of Weak Bisimilarity

by

Mojmír Křetínský
Vojtěch Řehák
Jan Strejček

FI MU Report Series FIMU-RS-2005-06

Refining the Undecidability Border of Weak Bisimilarity[*]

Mojmír Křetínský[†] Vojtěch Řehák[‡] Jan Strejček[§]

Faculty of Informatics, Masaryk University in Brno

Botanická 68a, 602 00 Brno, Czech Republic

{kretinsky,rehak,strejcek}@fi.muni.cz

August 19, 2005

Abstract

Weak bisimilarity is one of the most studied behavioural equivalences. This equivalence is undecidable for *pushdown processes* (PDA), *process algebras* (PA), and *multiset automata* (MSA, also known as *parallel pushdown processes*, PPDA). Its decidability is an open question for *basic process algebras* (BPA) and *basic parallel processes* (BPP). We move the undecidability border towards these classes by showing that the equivalence remains undecidable for weakly extended versions of BPA and BPP. Further, we show the results hold for even more restricted classes of normed BPA with finite constraint system and normed BPP with finite constraint system.

1 Introduction

Equivalence checking is one of the main streams in verification of concurrent systems. It aims at demonstrating some semantic equivalence between two systems, one of which is usually considered as representing the specification, the other as its implementation

[*]This is a full version of INFINITY'05 paper [12].

[†]Supported by the Grant Agency of the Czech Republic, grant No. 201/03/1161.

[‡]Supported by the research centre "Institute for Theoretical Computer Science (ITI)", project No. 1M0021620808.

[§]Supported by the Academy of Sciences of the Czech Republic, grant No. 1ET408050503.

or refinement. The semantic equivalences are designed to correspond to the system behaviours at the desired level of abstraction; the most prominent ones being strong and weak bisimulations.

Current software systems often exhibit an evolving structure and/or operate on unbounded data types. Hence automatic verification of such systems usually requires modelling them as infinite-state ones. Various specification formalisms have been developed with their respective advantages and limitations. *Petri nets* (PN), *pushdown processes* (PDA), and process algebras like BPA, BPP, or PA all serve to exemplify this. Here we employ the classes of infinite-state systems defined by term rewrite systems and called *Process Rewrite Systems* (PRS) as introduced by Mayr [13]. PRS subsume a variety of the formalisms studied in the context of formal verification (e.g. all the models mentioned above). The relevance of various subclasses of PRS for modelling and analysing programs is shown, for example, in [5]; for automatic verification we refer to surveys [2, 23].

The relative expressive power of various process classes has been studied, especially with respect to strong bisimulation; see [3, 17], also [13] showing the hierarchy of PRS classes. Adding a finite-state control unit to the PRS rewriting mechanism results in so-called *state-extended PRS* (sePRS) classes, see for example [8]. We have extended the PRS hierarchy by sePRS classes and refined this extended hierarchy by introducing restricted state extensions of two types: PRS equipped with a weak finite-state unit (wPRS, inspired by weak automata [18]) [11, 10] and PRS with finite constraint unit (fcPRS) [24].

Research on the expressive power of process classes has been accompanied by exploring algorithmic boundaries of various verification problems. In this paper we focus on the equivalence checking problem taking weak bisimilarity as the notion of behavioral equivalence.

State of the art: Regarding sequential systems, i.e. those without parallel composition, the weak bisimilarity problem is undecidable for PDA even for the normed case [20]. However, it is conjectured [14] that weak bisimilarity is decidable for BPA; the best known lower bound is *EXPTIME*-hardness [14].

Considering parallel systems, even strong bisimilarity is undecidable for MSA [17] using the technique introduced in [6]. However, it is conjectured [7] that the weak bisimilarity problem is decidable for BPP; the best known lower bound is *PSPACE*-hardness [21].

For the simplest systems combining both parallel and sequential operators, called PA processes [1], the weak bisimilarity problem is undecidable [22]. It is an open question for the normed PA; the best known lower bound is *EXPTIME*-hardness [14].

Our contribution: We move the undecidability border of the weak bisimilarity problem towards the classes of BPA and BPP, where the problem is conjectured to be decidable. Section 4 contains relatively simple proofs of undecidability of the considered problem for the weakly extended versions of BPA (wBPA) and BPP (wBPP). In Section 4, we strengthen the result for even more restricted systems such as normed fcBPA and normed fcBPP systems. In fact, the result is not new for wBPA due to the following reasons: Mayr [14] has shown that adding a finite-state unit of the minimal non-trivial size 2 to the BPA process already makes weak bisimilarity undecidable. Our inspection of his proof shows that the result is valid for wBPA as well.

2 Preliminaries

We recall the definitions of labelled transition system and weak bisimilarity. Then we define the syntax of process rewrite systems, (weak) finite-state unit extensions of PRS, and PRS with finite constraint systems. Their semantics is given in terms of labelled transition systems.

Definition 2.1. *Let $Act = \{a, b, \ldots\}$ be a set of* actions *such that Act contains a distinguished* silent action τ. *A* labelled transition system *is a pair $(S, \longrightarrow)$, where S is a set of* states *and* $\longrightarrow \subseteq S \times Act \times S$ *is a* transition relation.

We write $s_1 \xrightarrow{a} s_2$ instead of $(s_1, a, s_2) \in \longrightarrow$. The transition relation is extended to finite words over *Act* in the standard way. Further, we extend the relation to language $L \subseteq Act^*$ such that $s_1 \xrightarrow{L} s_2$ if $s_1 \xrightarrow{w} s_2$ for some $w \in L$. Moreover, we write $s_1 \longrightarrow^* s_2$ instead of $s_1 \xrightarrow{Act^*} s_2$. The *weak transition relation* $\Longrightarrow \subseteq S \times Act \times S$ is defined as $\overset{\tau}{\Longrightarrow} = \xrightarrow{\tau^*}$ and $\overset{a}{\Longrightarrow} = \xrightarrow{\tau^* a \tau^*}$ for all $a \neq \tau$.

Definition 2.2. *A binary relation R on states of a labelled transition system is a* weak bisimulation *iff whenever $(s_1, s_2) \in R$ then for any $a \in Act$:*

- *if $s_1 \xrightarrow{a} s_1'$ then $s_2 \overset{a}{\Longrightarrow} s_2'$ for some s_2' such that $(s_1', s_2') \in R$ and*
- *if $s_2 \xrightarrow{a} s_2'$ then $s_1 \overset{a}{\Longrightarrow} s_1'$ for some s_1' such that $(s_1', s_2') \in R$.*

States s_1 *and* s_2 *are* weakly bisimilar, *written* $s_1 \approx s_2$, *iff* $(s_1, s_2) \in R$ *for some weak bisimulation* R.

We use a characterization of weak bisimilarity in terms of a *bisimulation game*. This is a two-player game between an *attacker* and a *defender* played in rounds on pairs of states of a considered labelled transition system. In a round starting at a pair of states (s_1, s_2), the attacker first chooses $i \in \{1, 2\}$, an action $a \in Act$, and a state s'_i such that $s_i \xrightarrow{a} s'_i$. The defender then has to choose a state s'_{3-i} such that $s_{3-i} \overset{a}{\Longrightarrow} s'_{3-i}$. The states s'_1, s'_2 form a pair of starting states for the next round. A *play* is a maximal sequence of pairs of states chosen by players in the given way. The defender is the winner of every infinite play. A finite game is lost by the player who cannot make any choice satisfying the given conditions. It can be shown that two states s_1, s_2 of a labelled transition system are not weakly bisimilar if and only if the attacker has a winning strategy for the bisimulation game starting in these states.

Let $Const = \{X, \ldots\}$ be a set of *process constants*. The set of *process terms* (ranged over by $t, \ldots$) is defined by the abstract syntax

$$t ::= \varepsilon \mid X \mid t.t \mid t \| t$$

where ε is the *empty term*, $X \in Const$ is a process constant; and '.' and '$\|$' mean *sequential* and *parallel composition* respectively. We always work with equivalence classes of terms modulo commutativity and associativity of '$\|$', associativity of '.', and neutrality of ε, i.e. $\varepsilon.t = t = t.\varepsilon = t\|\varepsilon = t$. We distinguish four *classes of process terms* as:

1 – terms consisting of a single process constant only, in particular $\varepsilon \notin 1$,

S – *sequential* terms - terms without parallel composition, e.g. X.Y.Z,

P – *parallel* terms - terms without sequential composition, e.g. $X\|Y\|Z$,

G – *general* terms - terms with arbitrarily nested sequential and parallel compositions, e.g. $(X.(Y\|Z))\|W$.

Definition 2.3. *Let* α, β *be classes of process terms* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$. *An* (α, β)-PRS (process rewrite system) Δ *is a finite set of* rewrite rules *of the form* $t_1 \overset{a}{\hookrightarrow} t_2$, *where* $t_1 \in \alpha \setminus \{\varepsilon\}$, $t_2 \in \beta$ *are process terms and* $a \in Act$ *is an action.*

Given a PRS Δ, *let* $Const(\Delta)$ *and* $Act(\Delta)$ *be the respective (finite) sets of all constants and all actions which occur in the rewrite rules of* Δ.

An (α, β)-PRS Δ determines a labelled transition system where states are process terms $t \in \beta$ over *Const*(Δ). The transition relation $\longrightarrow$ is the least relation satisfying the following inference rules (recall that '$\|$' is commutative):

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

The formalism of process rewrite systems can be extended to include a finite-state control unit in the following way.

Definition 2.4. *Let* $M = \{m, n, \ldots\}$ *be a set of* control states*. Let* α, β *be classes of process terms* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$*. An* (α, β)-sePRS (state extended process rewrite system) Δ *is a finite set of* rewrite rules *of the form* $(m, t_1) \overset{a}{\hookrightarrow} (n, t_2)$*, where* $t_1 \in \alpha \setminus \{\varepsilon\}$*,* $t_2 \in \beta$*,* $m, n \in M$*, and* $a \in$ *Act.*

$M(\Delta)$ *denotes the finite set of control states which occur in* Δ*.*

An (α, β)-sePRS Δ determines a labelled transition system where states are the pairs of the form (m, t) such that $m \in M(\Delta)$ and $t \in \beta$ is a process term over *Const*(Δ). The transition relation $\longrightarrow$ is the least relation satisfying the following inference rules:

$$\frac{((m, t_1) \overset{a}{\hookrightarrow} (n, t_2)) \in \Delta}{(m, t_1) \xrightarrow{a} (n, t_2)} \qquad \frac{(m, t_1) \xrightarrow{a} (n, t_2)}{(m, t_1 \| t) \xrightarrow{a} (n, t_2 \| t)} \qquad \frac{(m, t_1) \xrightarrow{a} (n, t_2)}{(m, t_1.t) \xrightarrow{a} (n, t_2.t)}$$

To shorten our notation we write mt in lieu of (m, t).

Definition 2.5. *An* (α, β)*-sePRS* Δ *is called a* process rewrite system with weak finite-state control unit *or just a* weakly extended process rewrite system*, written* (α, β)*-wPRS, if there exists a partial order* $\leq$ *on* $M(\Delta)$ *such that every rule* $(m, t_1) \overset{a}{\hookrightarrow} (n, t_2)$ *of* Δ *satisfies* $m \leq n$.

Finally, we recall the extension of *process rewrite systems with finite constraint systems* introduced in [24]. This extension has been directly motivated by constraint systems used in *concurrent constraint programming* (CCP), for example, see [19].

Definition 2.6. *A* constraint system *is a bounded lattice* $(C, \geq, \wedge, tt, ff)$*, where* C *is the set of* constraints*,* $\geq$ *(called* entailment*) is an ordering on this set,* $\wedge$ *is the least upper bound operation, and tt (*true*), ff (*false*) are the least and the greatest elements of* C *respectively (*$ff \geq tt$ *and* $tt \neq ff$*).*

Example 2.7. *Example of a constraint system.*

$$\begin{tikzcd}
 & \mathit{ff} & \\
\mathrm{m} \arrow[ur, no head] & & \mathrm{n} \arrow[ul, no head] \\
 & \mathit{tt} \arrow[ul, no head] \arrow[ur, no head] &
\end{tikzcd}$$

Definition 2.8. *Let* $\mathcal{C} = (C, \geq, \wedge, \mathit{tt}, \mathit{ff})$ *be a finite constraint system describing the* store*; the elements of* C *represent* values of the store. *Let* α, β *be classes of process terms* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$. *An* (α, β)-fcPRS (PRS with finite constraint system) Δ *is a finite set of* rewrite rules *of the form* $(m, t_1) \overset{a}{\hookrightarrow} (n, t_2)$*, where* $t_1 \in \alpha$*,* $t_1 \neq \varepsilon$*,* $t_2 \in \beta$ *are process terms,* $a \in \mathit{Act}$*, and* $m, n \in C$ *are constraints.*

$C(\Delta)$ *denotes the finite set of constraints which occur in* Δ*.*

An (α, β)-fcPRS Δ determines a labelled transition system where states are the pairs of the form (m, t) such that $m \in C(\Delta) \smallsetminus \{\mathit{ff}\}$ and $t \in \beta$ is a process term over *Const*(Δ). The transition relation $\longrightarrow$ is the least relation satisfying the following inference rules:

$$\frac{((m, t_1) \overset{a}{\hookrightarrow} (n, t_2)) \in \Delta}{(o, t_1) \xrightarrow{a} (o \wedge n, t_2)} \quad \text{if } o \geq m \text{ and } o \wedge n \neq \mathit{ff},$$

$$\frac{(o, t_1) \xrightarrow{a} (p, t_2)}{(o, t_1 \| t) \xrightarrow{a} (p, t_2 \| t)}, \qquad \frac{(o, t_1) \xrightarrow{a} (p, t_2)}{(o, t_1.t) \xrightarrow{a} (p, t_2.t)}.$$

To shorten our notation we write mt in lieu of (m, t).

The two side conditions of the first inference rule are very close to principles used in CCP. The first one ($o \geq m$) ensures the rule $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$ can be used only if the current value of the store o *entails* m (it is similar to *ask(m)* in CCP). The second condition ($o \wedge n \neq \mathit{ff}$) guarantees that the store stays *consistent* after the application of the rule (analogous to the consistency requirement when processing *tell(n)* in CCP).

An important observation is that the value of a store can move in a lattice only upwards as $o \wedge n$ always entails o. Intuitively, partial information can only be added to the store, but never retracted (the store is *monotonic*).

Please note that an execution of a transition which starts in a state with o on the store and which is generated by a rule $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$ implies that for every subsequent value of the store p the conditions $p \geq m$ and $p \wedge n \neq \mathit{ff}$ are satisfied (and thus the use of the rule cannot be forbidden by a value of the store in the future). The first condition $p \geq m$ comes from the monotonic behaviour of the store. The second condition comes from two following facts: the constraint n of the rule can only change the store in the first application of the rule; and $p \wedge n = p$ holds for any subsequent state p of the store.

Definition 2.9. *An (α, β)-fcPRS Δ is* normed in a state m_0t_0 *of Δ iff for all states mt satisfying $m_0t_0 \longrightarrow^* mt$ it holds that $mt \longrightarrow^* o\varepsilon$ for some $o \in C(\Delta)$.*

Some classes of (α, β)-PRS correspond to widely known models as *finite-state systems* (FS), *basic process algebras* (BPA), *basic parallel processes* (BPP), *process algebras* (PA), *pushdown processes* (PDA, see [4] for justification), and *Petri nets* (PN). The other (α, β)-PRS classes were introduced and named as PAD, PAN, and PRS by Mayr [13]. The correspondence between (α, β)-PRS classes and the acronyms is given in Figure 1. Instead of (α, β)-sePRS, (α, β)-wPRS, and (α, β)-fcPRS we use the prefixes 'se-', 'w-', and 'fc-' in connection with the acronym for the corresponding (α, β)-PRS class. For example, we use wBPA and wBPP rather than $(1, S)$-wPRS and $(1, P)$-wPRS, respectively. Finally, we note that seBPP are also known as *multiset automata* (MSA) or *parallel pushdown processes* (PPDA).

Figure 1 depicts relations between the expressive power of the considered classes. The expressive power of a class is measured by the set of labelled transition systems that are definable (up to strong bisimulation equivalence) by the class. A solid line between two classes means that the upper class is strictly more expressive than the lower one. A dotted line means that the upper class is at least as expressive as the lower class (and the strictness is just our conjecture). Details can be found in [11, 10].

3 Undecidability of Weak Bisimilarity

In this section, we show that weak bisimilarity is undecidable for the classes wBPA and wBPP. More precisely, we study the following problems for extended (α, β)-PRS classes.

Problem:	*Weak bisimilarity problem for an extended (α, β)-PRS class*
Instance:	An extended (α, β)-PRS system Δ and two of its states $mt, m't'$
Question:	Are the two states weakly bisimilar?

3.1 wBPA

In [14] Mayr studied the question of how many control states are needed in PDA to make weak bisimilarity undecidable.

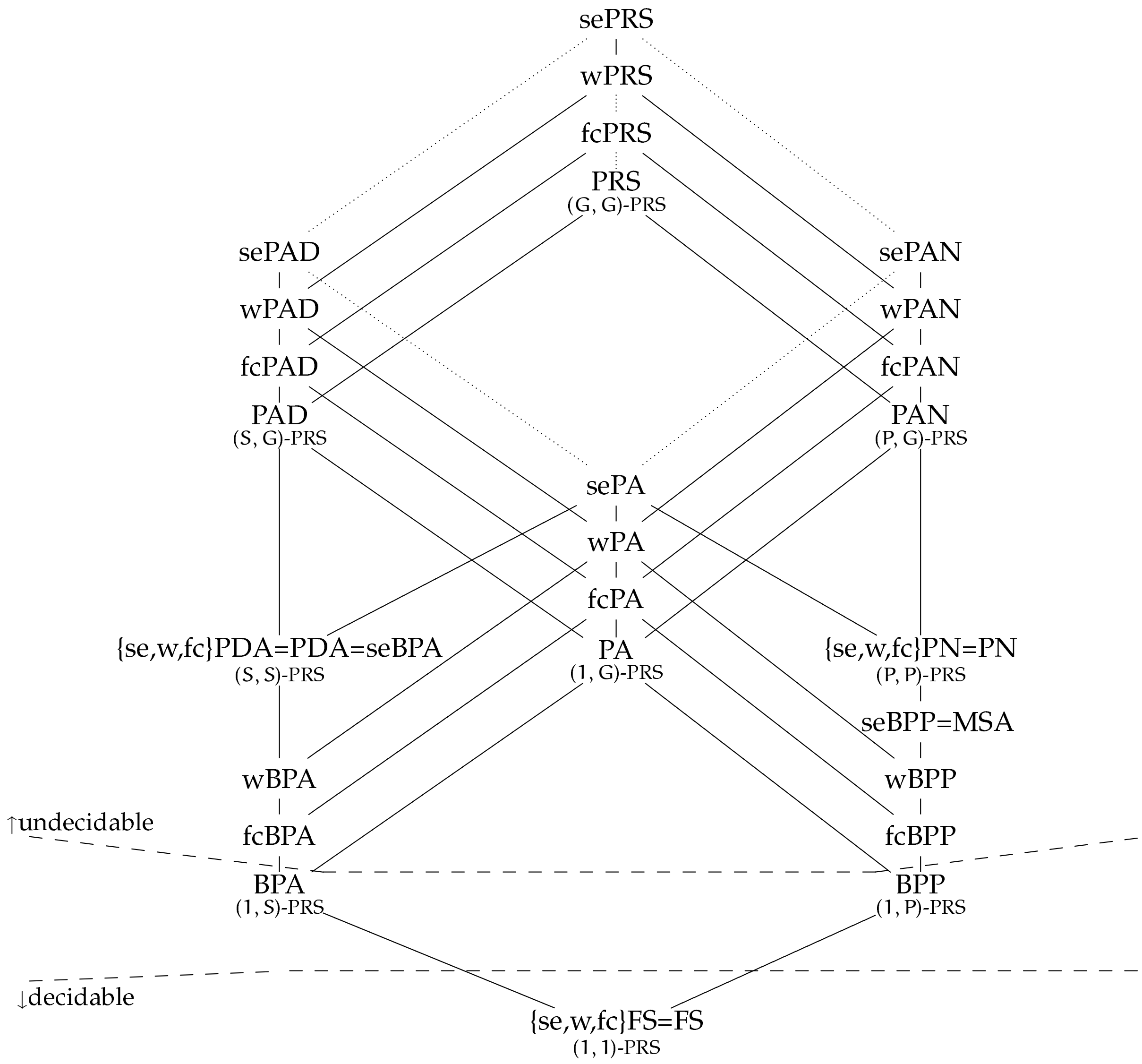


Figure 1: The hierarchy with (un)decidability boundaries of weak bisimilarity.

Theorem 3.1 ([14], Theorem 29). *Weak bisimilarity is undecidable for pushdown automata with only 2 control states.*

The proof is done by a reduction of Post's correspondence problem to the weak bisimilarity problem for PDA. The constructed PDA has only two control states, p and q. Quick inspection of the construction shows that the resulting pushdown automata are in fact wBPA systems as there is no transition rule changing q to p and each rule has only one process constant on the left hand side. Hence Mayr's theorem can be reformulated as follows.

Theorem 3.2. *Weak bisimilarity is undecidable for wBPA systems with only 2 control states.*

3.2 wBPP

We show that the non-halting problem for Minsky 2-counter machines can be reduced to the weak bisimilarity problem for wBPP. First, let us recall the notions of Minsky 2-counter machine and the non-halting problem.

A *Minsky 2-counter machine*, or a *machine* for short, is a finite sequence

$$N = l_1\colon i_1,\ l_2\colon i_2,\ \ldots,\ l_{n-1}\colon i_{n-1},\ l_n\colon \texttt{halt}$$

where $n \geq 1$, $l_1, l_2, \ldots, l_n$ are *labels*, and each i_j is an instruction for

- *increment*: $\texttt{c}_k\texttt{:=}\texttt{c}_k\texttt{+1;}\ \texttt{goto}\ l_r$, or
- *test-and-decrement*: $\texttt{if}\ \texttt{c}_k\texttt{>0}\ \texttt{then}\ \texttt{c}_k\texttt{:=}\texttt{c}_k\texttt{-1;}\ \texttt{goto}\ l_r\ \texttt{else}\ \texttt{goto}\ l_s$

where $k \in \{1, 2\}$ and $1 \leq r, s \leq n$.

The semantics of a machine N is given by a labelled transition system the states of which are *configurations* of the form (l_j, v_1, v_2) where l_j is a label of an instruction to be executed and v_1, v_2 are nonnegative integers representing current values of counters $\texttt{c}_1$ and $\texttt{c}_2$, respectively. The transition relation is the smallest relation satisfying the following conditions: if i_j is an instruction of the form

- $\texttt{c}_1\texttt{:=}\texttt{c}_1\texttt{+1;}\ \texttt{goto}\ l_r$, then $(l_j, v_1, v_2) \xrightarrow{inc} (l_r, v_1 + 1, v_2)$ for all $v_1, v_2 \geq 0$;
- $\texttt{if}\ \texttt{c}_1\texttt{>0}\ \texttt{then}\ \texttt{c}_1\texttt{:=}\texttt{c}_1\texttt{-1;}\ \texttt{goto}\ l_r\ \texttt{else}\ \texttt{goto}\ l_s$, then $(l_j, v_1 + 1, v_2) \xrightarrow{dec} (l_r, v_1, v_2)$ and $(l_j, 0, v_2) \xrightarrow{zero} (l_s, 0, v_2)$ for all $v_1, v_2 \geq 0$;

and the analogous condition for instructions manipulating $\texttt{c}_2$. We say that the (computation of) machine N *halts* if there are numbers $v_1, v_2 \geq 0$ such that $(l_1, 0, 0) \longrightarrow^* (l_n, v_1, v_2)$. Let us note that the system is deterministic, i.e. for every configuration there is at most one transition leading from the configuration.

The *non-halting problem* is to decide whether a given machine N does not halt. The problem is undecidable [16].

Let us fix a machine $N = l_1 : i_1,\ l_2 : i_2,\ \ldots,\ l_{n-1} : i_{n-1},\ l_n : \texttt{halt}$. We construct a wBPP system Δ such that its states $\mathit{sim}L_1$ and $\mathit{sim}L'_1$ are weakly bisimilar if and only if N does not halt. Roughly speaking, we create a set of wBPP rules allowing us to simulate the computation of N by two separate sets of terms. If the instruction `halt` is reached in the computation of N, the corresponding term from one set can perform the action *halt*, while the corresponding term from the other set can perform the action *halt'*. Therefore, the starting terms are weakly bisimilar if and only if the machine does not halt.

The wBPP system Δ we are going to construct uses five control states, namely $\mathit{sim}, \mathit{check}_1, \mathit{check}'_1, \mathit{check}_2, \mathit{check}'_2$. We associate each label l_j and each counter c_k with process constants L_j, L'_j and X_k, Y_k respectively. A parallel composition of x copies of X_k and y copies of Y_k, written $X_k^x \| Y_k^y$, represents the fact that the counter c_k has the value $x - y$. Hence, terms $\mathit{sim}L_j \| X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$ and $\mathit{sim}L'_j \| X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$ are associated with a configuration $(l_j, x_1 - y_1, x_2 - y_2)$ of the machine N. Some rules contain auxiliary process constants. In what follows, β stands for a term of the form $\beta = X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$. The control states $\mathit{check}_k, \mathit{check}'_k$ for $k \in \{1, 2\}$ are intended for testing emptiness of the counter c_k. The only rules applicable in these control states are:

$$\mathit{check}_1 X_1 \overset{\mathit{chk}_1}{\hookrightarrow} \mathit{check}_1 \varepsilon \qquad \mathit{check}_2 X_2 \overset{\mathit{chk}_2}{\hookrightarrow} \mathit{check}_2 \varepsilon$$
$$\mathit{check}'_1 Y_1 \overset{\mathit{chk}_1}{\hookrightarrow} \mathit{check}'_1 \varepsilon \qquad \mathit{check}'_2 Y_2 \overset{\mathit{chk}_2}{\hookrightarrow} \mathit{check}'_2 \varepsilon$$

One can readily confirm that $\mathit{check}_k \beta \approx \mathit{check}'_k \beta$ if and only if the value of c_k represented by β equals zero.

In what follows we construct a set of wBPP rules for each instruction of the machine N. At the same time we argue that the only chance for the attacker to win is to simulate the machine without cheating as every cheating can be punished by the defender's victory. This attacker's strategy is winning if and only if the machine halts.

Halt: l_n: `halt`

Halt instruction is translated into the following two rules:

$$\mathit{sim}L_n \overset{\mathit{halt}}{\hookrightarrow} \mathit{sim}\varepsilon \qquad \mathit{sim}L'_n \overset{\mathit{halt}'}{\hookrightarrow} \mathit{sim}\varepsilon$$

Clearly, *the states* $\mathit{sim}L_n \| \beta$ *and* $\mathit{sim}L'_n \| \beta$ *are not weakly bisimilar.*

Increment: l_j: `c`$_k$`:=c`$_k$`+1; goto l`$_r$

For each such an instruction of the machine N we add the following rules to Δ:

$$\mathit{sim}L_j \overset{\mathit{inc}}{\hookrightarrow} \mathit{sim}L_r \| X_k \qquad \mathit{sim}L'_j \overset{\mathit{inc}}{\hookrightarrow} \mathit{sim}L'_r \| X_k$$

Obviously, *every round of the bisimulation game starting at states* $\mathit{sim}L_j \| \beta$ *and* $\mathit{sim}L'_j \| \beta$ *ends up in states* $\mathit{sim}L_r \| X_k \| \beta$ *and* $\mathit{sim}L'_r \| X_k \| \beta$.

Test-and-decrement: l_j: `if c`$_k$`>0 then c`$_k$`:=c`$_k$`-1; goto l`$_r$ `else goto l`$_s$

For any such instruction of the machine N we add two sets of rules to Δ, one for the $c_k > 0$ case and the other for the $c_k = 0$ case. The wBPP formalism has no power to rewrite a process constant corresponding to a label l_j and to check whether $c_k > 0$ at the same time. Therefore, in the bisimulation game it is the attacker who has to decide whether $c_k > 0$ holds or not, i.e. whether he will play an action *dec* or an action *zero*. We show that whenever the attacker tries to cheat, the defender can win the game.

At this point our construction of wBPP rules uses a variant of the technique called *defender's choice* [9]. In a round starting at the pair of states s_1, s_2, the attacker is forced to choose one specific transition (indicated by a wavy arrow henceforth). If he chooses a different transition, say $s_k \xrightarrow{a} s$ where $k \in \{1,2\}$, then there exists a transition $s_{3-k} \xrightarrow{a} s$ that enables the defender to reach the same state and win the play. The name of this technique refers to the fact that after the attacker chooses the specific transition, the defender can choose an arbitrary transition with the same label. These transitions are indicated by solid arrows. The dotted arrows stands for auxiliary transitions which compel the attacker to play the specific transition.

First, we discuss the following rules designed for the $c_k > 0$ case:

$$\begin{array}{lll}
\mathit{sim}L_j \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}A_{k,r} & \mathit{sim}A_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{check}_k\varepsilon & \mathit{sim}B_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}L_r\|Y_k \\
\mathit{sim}L_j \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}B_{k,r} & \mathit{sim}A_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}L'_r\|Y_k & \mathit{sim}B_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}L'_r\|Y_k \\
\mathit{sim}L'_j \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}A_{k,r} & \mathit{sim}A_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{check}'_k\varepsilon & \mathit{sim}B_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{check}'_k\varepsilon \\
\mathit{sim}L'_j \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}B_{k,r} & & \mathit{sim}C_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}L'_r\|Y_k \\
\mathit{sim}L'_j \overset{\mathit{dec}}{\hookrightarrow} \mathit{sim}C_{k,r} & & \mathit{sim}C_{k,r} \overset{\mathit{dec}}{\hookrightarrow} \mathit{check}'_k\varepsilon
\end{array}$$

The situation can be depicted as follows.

$$\begin{tikzcd}
 & \mathit{sim}L_j\|\beta \arrow[dl, "\mathit{dec}"'] \arrow[d, "\mathit{dec}"] & \mathit{sim}L'_j\|\beta \arrow[dll, dotted, "\mathit{dec}"'] \arrow[dl, dotted, "\mathit{dec}"] \arrow[dr, squiggly, "\mathit{dec}"] & \\
\mathit{sim}A_{k,r}\|\beta \arrow[d, squiggly, "\mathit{dec}"'] \arrow[drr, dotted, "\mathit{dec}"] \arrow[drrr, dotted, "\mathit{dec}"] & \mathit{sim}B_{k,r}\|\beta \arrow[d, squiggly, "\mathit{dec}"] \arrow[dr, dotted, "\mathit{dec}"] \arrow[drr, dotted, "\mathit{dec}"] & & \mathit{sim}C_{k,r}\|\beta \arrow[dl, "\mathit{dec}"'] \arrow[d, "\mathit{dec}"] \\
\mathit{check}_k\beta & \mathit{sim}L_r\|Y_k\|\beta & \mathit{sim}L'_r\|Y_k\|\beta & \mathit{check}'_k\beta
\end{tikzcd}$$

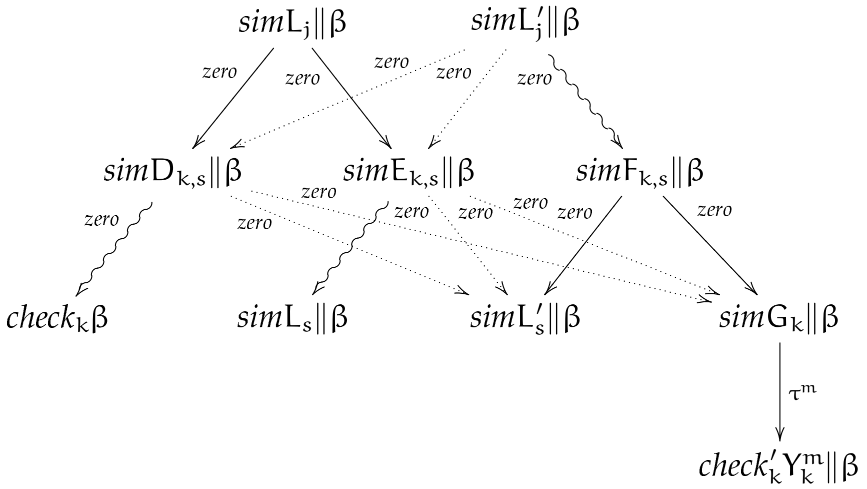
Let us assume that in a round starting at states $\mathit{sim}L_j\|\beta, \mathit{sim}L'_j\|\beta$ the attacker decides to perform the action *dec*. Due to the principle of defender's choice employed here, the attacker has to play the transition $\mathit{sim}L'_j\|\beta \xrightarrow{\mathit{dec}} \mathit{sim}C_{k,r}\|\beta$, while the defender can choose between the transitions leading from $\mathit{sim}L_j\|\beta$ either to $\mathit{sim}A_{k,r}\|\beta$ or to

$sim\mathrm{B}_{k,r}\|\beta$. Thus, the round will finish either in states $sim\mathrm{A}_{k,r}\|\beta, sim\mathrm{C}_{k,r}\|\beta$ or in states $sim\mathrm{B}_{k,r}\|\beta, sim\mathrm{C}_{k,r}\|\beta$. Using the defender's choice again, one can easily see that the next round ends up in $check_k\beta$ or $sim\mathrm{L}_r\|\mathrm{Y}_k\|\beta$, and $sim\mathrm{L}'_r\|\mathrm{Y}_k\|\beta$ or $check'_k\beta$. The exact combination is chosen by the defender. The defender will not choose any pair of states where one control state is *sim* and the other is not as such states are clearly not weakly bisimilar. Hence, the two considered rounds of the bisimulation game end up in a pair of states either $sim\mathrm{L}_r\|\mathrm{Y}_k\|\beta, sim\mathrm{L}'_r\|\mathrm{Y}_k\|\beta$ or $check_k\beta, check'_k\beta$. The latter pair is weakly bisimilar iff the value of c_k represented by β is zero, i.e. iff the attacker cheats when he decides to play an action *dec*. This means that *if the attacker cheats, the defender wins. If the attacker plays the action dec correctly, the only chance for either player to force a win is to finish these two rounds in states* $sim\mathrm{L}_r\|\mathrm{Y}_k\|\beta, sim\mathrm{L}'_r\|\mathrm{Y}_k\|\beta$ *corresponding to the correct simulation of an test-and-decrement instruction with a label* l_j.

Now, we focus on the following rules designed for the $c_k = 0$ case:

$$\begin{array}{lll}
sim\mathrm{L}_j \overset{zero}{\hookrightarrow} sim\mathrm{D}_{k,s} & sim\mathrm{D}_{k,s} \overset{zero}{\hookrightarrow} check_k\varepsilon & sim\mathrm{E}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{L}_s \\
sim\mathrm{L}_j \overset{zero}{\hookrightarrow} sim\mathrm{E}_{k,s} & sim\mathrm{D}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{L}'_s & sim\mathrm{E}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{L}'_s \\
sim\mathrm{L}'_j \overset{zero}{\hookrightarrow} sim\mathrm{D}_{k,s} & sim\mathrm{D}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{G}_k & sim\mathrm{E}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{G}_k \\
sim\mathrm{L}'_j \overset{zero}{\hookrightarrow} sim\mathrm{E}_{k,s} & sim\mathrm{F}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{L}'_s & sim\mathrm{G}_k \overset{\tau}{\hookrightarrow} sim\mathrm{G}_k\|\mathrm{Y}_k \\
sim\mathrm{L}'_j \overset{zero}{\hookrightarrow} sim\mathrm{F}_{k,s} & sim\mathrm{F}_{k,s} \overset{zero}{\hookrightarrow} sim\mathrm{G}_k & sim\mathrm{G}_k \overset{\tau}{\hookrightarrow} check'_k\mathrm{Y}_k
\end{array}$$

The situation can be depicted as follows.



$$\begin{tikzcd}
 & & sim\mathrm{L}_j\|\beta \arrow[dl, "zero"'] \arrow[dr, "zero"] & & sim\mathrm{L}'_j\|\beta \arrow[dlll, dotted, "zero"'] \arrow[dl, dotted, "zero"'] \arrow[dr, rightsquigarrow, "zero"] & & \\
 & sim\mathrm{D}_{k,s}\|\beta \arrow[dl, rightsquigarrow, "zero"'] \arrow[drrr, dotted, "zero"'] \arrow[drrrrr, dotted, "zero"] & & sim\mathrm{E}_{k,s}\|\beta \arrow[dl, rightsquigarrow, "zero"] \arrow[dr, dotted, "zero"] \arrow[drrr, dotted, "zero"] & & sim\mathrm{F}_{k,s}\|\beta \arrow[dl, "zero"'] \arrow[dr, "zero"] & \\
check_k\beta & & sim\mathrm{L}_s\|\beta & & sim\mathrm{L}'_s\|\beta & & sim\mathrm{G}_k\|\beta \arrow[d, "\tau^m"] \\
 & & & & & & check'_k\mathrm{Y}_k^m\|\beta
\end{tikzcd}$$

Let us assume that the attacker decides to play the action *zero*. The defender's choice technique allows the defender to control the two rounds of the bisimulation game starting at states $sim\mathrm{L}_j\|\beta$ and $sim\mathrm{L}'_j\|\beta$. The two rounds end up in a pair of states $sim\mathrm{L}_s\|\beta, sim\mathrm{L}'_s\|\beta$ or in a pair of the form $check_k\beta, check'_k\mathrm{Y}_k^m\|\beta$ where $m \geq 1$; all the other choices of the defender lead to his loss. As in the previous case, the latter possibility is designed to punish any possible attacker's cheating. The attacker is cheating if he plays the action *zero* and the value of c_k represented by β, say v_k, is positive[1]. In such a case, the defender can control the two rounds to end up in states $\mathit{check}_k\beta, \mathit{check}'_k Y_k^{v_k} \| \beta$ which are weakly bisimilar. If the attacker plays correctly, i.e. the value of c_k represented by β is zero, then the defender has to control the two discussed rounds to end up in states $\mathit{sim}L_s \| \beta, \mathit{sim}L'_s \| \beta$ as the states $\mathit{check}_k\beta, \mathit{check}'_k Y_k^m \| \beta$ are not weakly bisimilar for any $m \geq 1$. To sum up, *the attacker's cheating can be punished by defender's victory. If the attacker plays correctly, the only chance for both players to win is to end up after the two rounds in states* $\mathit{sim}L_s \| \beta, \mathit{sim}L'_s \| \beta$ *corresponding to the correct simulation of the considered instruction.*

It has been argued that if each of the two players wants to win, then both players will correctly simulate the computation of the machine N. The computation is finite if and only if the machine halts. The states $\mathit{sim}L_1$ and $\mathit{sim}L'_1$ are not weakly bisimilar in this case. If the machine does not halt, the play is infinite and the defender wins. Hence, the two states are weakly bisimilar in this case. In other words, *the states* $\mathit{sim}L_1$ *and* $\mathit{sim}L'_1$ *of the constructed wBPP* Δ *are weakly bisimilar if and only if the Minsky 2-counter machine* N *does not halt.* Hence, we have proved the following theorem.

Theorem 3.3. *Weak bisimilarity is undecidable for wBPP systems.*

4 Weak Bisimilarity for More Restricted Classes

Here, we strengthen the results of the previous section. We show that weak bisimilarity remains undecidable for both fcBPP and fcBPA systems and moreover this holds even for their respective normed versions. Hence, weak bisimilarity is undecidable for normed wBPP and normed wBPA as well.

4.1 Normed fcBPP

In this subsection, we show that weak bisimilarity is undecidable for normed fcBPP systems.

[1]We do not have to consider the case when β represents a negative value of c_k as such a state is reachable in the game starting in states $\mathit{sim}L_1, \mathit{sim}L'_1$ only by unpunished cheating.

Let Δ be the wBPP system constructed in Subsection 3.2. We recall that given any fixed Minsky machine N, we have constructed a wBPP system Δ such that its states $sim\mathrm{L}_1$ and $sim\mathrm{L}'_1$ are weakly bisimilar if and only if N does not halt.

Based on Δ, we now construct a fcBPP Δ' and two of its states $sim\mathrm{L}_1\|\mathrm{D}$ and $sim\mathrm{L}'_1\|\mathrm{D}$ such that they satisfy the same condition as given in the previous paragraph and moreover Δ' is normed in both of the states $sim\mathrm{L}_1\|\mathrm{D}$ and $sim\mathrm{L}'_1\|\mathrm{D}$.

The constraint system of Δ' is defined as follows.

$$\begin{tikzcd}
 & & ff \arrow[d, dash] & & \\
 & & del \arrow[dll, dash] \arrow[dl, dash] \arrow[dr, dash] \arrow[drr, dash] & & \\
check_1 \arrow[drr, dash] & check'_1 \arrow[dr, dash] & & check_2 \arrow[dl, dash] & check'_2 \arrow[dll, dash] \\
 & & sim \arrow[d, dash] & & \\
 & & tt & &
\end{tikzcd}$$

Let $Const(\Delta') = \{\mathrm{D}\} \cup Const(\Delta)$ and $Act(\Delta') = \{norm\} \cup Act(\Delta)$, where $\mathrm{D} \notin Const(\Delta)$ is a fresh process constant and $norm \notin Act(\Delta)$ is a fresh action.

The set of rewrite rules Δ' is constructed as follows.

$$\begin{array}{lll}
(1) & pt_1 \overset{a}{\hookrightarrow} qt_2 & \text{for all } (pt_1 \overset{a}{\hookrightarrow} qt_2) \in \Delta, \\
(2) & tt\mathrm{D} \overset{norm}{\hookrightarrow} del\mathrm{D}, & \\
(3) & del\mathrm{X} \overset{\tau}{\hookrightarrow} del\varepsilon & \text{for all } \mathrm{X} \in Const(\Delta'), \\
(4) & del\mathrm{X} \overset{a}{\hookrightarrow} del\mathrm{X} & \text{for all } \mathrm{X} \in Const(\Delta') \text{ and } a \in Act(\Delta).
\end{array}$$

The process constant D enables the *norm* action changing the value of the store onto *del*. Starting in the state $sim\mathrm{L}_1\|\mathrm{D}$ or $sim\mathrm{L}'_1\|\mathrm{D}$, every reachable state includes the process constant D or the current value of the store has been already changed onto *del*. Whenever the value of the store is set to *del*, the rules of type (3) can be used to make the state normed. Hence, Δ' is normed in both of the states $sim\mathrm{L}_1\|\mathrm{D}$ and $sim\mathrm{L}'_1\|\mathrm{D}$.

The rewrite rules of the type (4) have been introduced as the result of the fact that one cannot forbid any further applications of the type (1) rules in the considered fcBPP systems.

Using the *norm* action in the game, weakly bisimilar states are received. As only the attacker can decide for the action, this reconstruction of Δ onto Δ' does not change the winning strategies discussed in Subsection 3.2. Hence the Theorem 3.3 can be strengthen as follows.

Theorem 4.1. *Weak bisimilarity is undecidable for normed fcBPP systems.*

4.2 Normed fcBPA

In this subsection, we show that the problem remains undecidable for normed fcBPA. Our proof is a slightly extended translation of the proof for PDA of [14] into fcBPA framework. We used the notation of [14] to make the proof comparable.

The proof is based on a reduction of Post's correspondence problem, which is known to be undecidable [15].

Problem: *Post's Correspondence problem (PCP)*

Instance: A non-unary alphabet Σ and two ordered sets of words $\mathcal{A} = \{u_1, \ldots, u_n\}$ and $\mathcal{B} = \{v_1, \ldots, v_n\}$ where $u_i, v_i \in \Sigma^+$.

Question: Do there exist finitely many indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$?

Given any instance of PCP we now construct a normed fcBPA Δ and two of its states $pTB, pT'B$ such that pTB and $pT'B$ are weakly bisimilar if and only if the instance of PCP has a solution.

A constraint system of Δ contains elements *tt*, *p*, *check*$_1$, *check*$_2$, *del*, and *ff* that are ordered as follows.

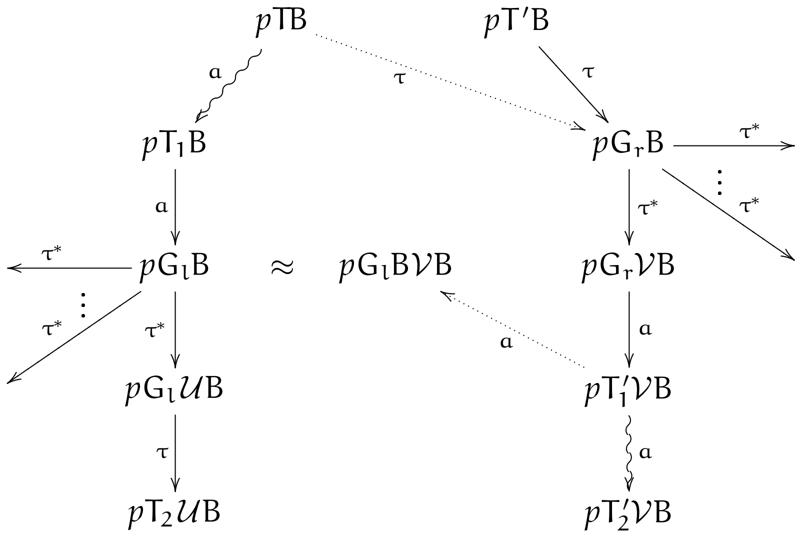
$$\begin{tikzcd}
 & ff \arrow[d, dash] & \\
 & del \arrow[dl, dash] \arrow[dr, dash] & \\
check_1 \arrow[dr, dash] & & check_2 \arrow[dl, dash] \\
 & p \arrow[d, dash] & \\
 & tt &
\end{tikzcd}$$

We use process constants $T, T', T_1, T_1', T_2, T_2', G_l, G_r, B$ and U_i, V_i for each $1 \leq i \leq n$. Actions of Δ are $a, b, c, \tau, \mathit{norm}, 1, \ldots, n$ and the letters of Σ. In what follows, $\mathcal{U}$ stands for a sequential term of process constants of $\{U_i \mid 1 \leq i \leq n\}$ and similarly $\mathcal{V}$ stands for a sequential term of process constants of $\{V_i \mid 1 \leq i \leq n\}$.

Now, we construct a set of rewrite rules Δ. The first rules are exactly the same as those of Mayr's proof and forms a defender's choice construction. The rewrite rules are as follows.

$$
\begin{aligned}
&(1) && pT \overset{a}{\hookrightarrow} pT_1 \\
&(2) && pT \overset{\tau}{\hookrightarrow} pG_r \\
&(3) && pT' \overset{\tau}{\hookrightarrow} pG_r \\
&(4) && pG_r \overset{\tau}{\hookrightarrow} pG_rV_i \quad \text{for all } i \in \{1, \dots, n\} \\
&(5) && pG_r \overset{a}{\hookrightarrow} pT_1' \\
&(6) && pT_1 \overset{a}{\hookrightarrow} pG_l \\
&(7) && pT_1' \overset{a}{\hookrightarrow} pG_lB \\
&(8) && pT_1' \overset{a}{\hookrightarrow} pT_2' \\
&(9) && pG_l \overset{\tau}{\hookrightarrow} pG_lU_i \quad \text{for all } i \in \{1, \dots, n\} \\
&(10) && pG_l \overset{\tau}{\hookrightarrow} pT_2
\end{aligned}
$$

If there is a solution of the instance of PCP, the defender can use these rules to finish the first two rounds of the bisimulation game (starting in pTB and $pT'B$) in states $pT_2\mathcal{U}B$ and $pT_2'\mathcal{V}B$, where $\mathcal{U}$ and $\mathcal{V}$ form a solution of the PCP instance. The discussed first two rounds of the bisimulation game are depicted in Figure 2. We use the same notation for arrows as in Subsection 3.2.



$$\begin{tikzcd}
 & & pTB \arrow[dl, squiggly, "a"'] \arrow[drrr, dotted, "\tau"] & & pT'B \arrow[dr, "\tau"] & & \\
 & pT_1B \arrow[d, "a"] & & & & pG_rB \arrow[r, "\tau^*"] \arrow[dr, "\tau^*"] \arrow[d, "\tau^*"] & {} \\
 {} & pG_lB \arrow[l, "\tau^*"'] \arrow[dl, "\tau^*"] \arrow[d, "\tau^*"] & \approx & pG_lB\mathcal{V}B & & pG_r\mathcal{V}B \arrow[d, "a"] & {} \\
 {} & pG_l\mathcal{U}B \arrow[d, "\tau"] & & & & pT_1'\mathcal{V}B \arrow[ull, dotted, "a"] \arrow[d, squiggly, "a"] & \\
 & pT_2\mathcal{U}B & & & & pT_2'\mathcal{V}B &
\end{tikzcd}$$

Figure 2: The first two rounds of the bisimulation game.

The following six rules form two subsequent rounds of the bisimulation game and allow attacker to decide whether to check equality of indices or equality of the letters of $\mathcal{U}$ and $\mathcal{V}$. In the first case, the attacker uses action b leading to the constraint $check_1$, while the second possibility is labelled by c and ends in the constraint $check_2$. The rewrite rules are as follows.

$$\begin{array}{llll} (11) & pT_2 \overset{a}{\hookrightarrow} pT_3 & (12) & pT'_2 \overset{a}{\hookrightarrow} pT'_3 \\ (13) & pT_3 \overset{b}{\hookrightarrow} \mathit{check}_1\varepsilon & (14) & pT'_3 \overset{b}{\hookrightarrow} \mathit{check}_1\varepsilon \\ (15) & pT_3 \overset{c}{\hookrightarrow} \mathit{check}_2\varepsilon & (16) & pT'_3 \overset{c}{\hookrightarrow} \mathit{check}_2\varepsilon \end{array}$$

Now, we list the rules that serve for the checking phases mentioned in the previous paragraph. In rules (19) and (20), we use a short notation that can be easily expressed by standard rules. The rewrite rules are as follows.

$$\begin{array}{lll} (17) & \mathit{check}_1U_i \overset{i}{\hookrightarrow} \mathit{check}_1\varepsilon & \text{for all } i \in \{1, \dots, n\} \\ (18) & \mathit{check}_1V_i \overset{i}{\hookrightarrow} \mathit{check}_1\varepsilon & \text{for all } i \in \{1, \dots, n\} \\ (19) & \mathit{check}_2U_i \overset{u_i}{\hookrightarrow} \mathit{check}_2\varepsilon & \text{for all } i \in \{1, \dots, n\} \\ (20) & \mathit{check}_2V_i \overset{v_i}{\hookrightarrow} \mathit{check}_2\varepsilon & \text{for all } i \in \{1, \dots, n\} \end{array}$$

Finally, we add rules that make the system normed. The construction of rules (21) and (22) is also discussed in Remark 30 of [14]. The rules of type (21) enables the *norm* action changing the value of the store onto *del*. In any state, whenever the value of the store is set to *del*, the rules of type (3) can be used to make the state normed. Hence, Δ is normed in all of its states. The rules of type (23) make all states composed of the *del* store and a non-empty term weakly bisimilar.

$$\begin{array}{lll} (21) & \mathit{tt}X \overset{\mathit{norm}}{\hookrightarrow} \mathit{del}X & \text{for all } X \in \mathit{Const}(\Delta) \\ (22) & \mathit{del}X \overset{\tau}{\hookrightarrow} \mathit{del}\varepsilon & \text{for all } X \in \mathit{Const}(\Delta) \\ (23) & \mathit{del}X \overset{x}{\hookrightarrow} \mathit{del}X & \text{for all } X \in \mathit{Const}(\Delta) \text{ and } x \in \mathit{Act}(\Delta) \end{array}$$

Hence, we have strengthen the Mayr's result [14], Theorem 29 (also reformulated as Theorem 3.2 of this paper) as follows.

Theorem 4.2. *Weak bisimilarity is undecidable for normed fcBPA systems.*

5 Conclusion

First, we have shown that the weak bisimilarity problem remains undecidable for weakly extended versions of BPP (wBPP) and BPA (wBPA) process classes.

We note that the result for wBPA is just our interpretation (in terms of weakly extended systems) of Mayr's proof showing that the problem is undecidable for PDA with two control states ([14], Theorem 29).

In terms of parallel systems, our technique used for wBPP is new. To mimic the computation of a Minsky 2-counter machine, one has to be able to maintain its state information – the label of a current instruction and the values of the counters c_1 and c_2. As the finite-state unit of wBPP is weak, it cannot be used to store even a part of such often changing information. Hence, contrary to the constructions in more expressive systems (PN [6] and MSA [17]) we have made the term part to manage it as follows. In a test-and-decrement instruction the process constant L_j has to be changed and moreover one of the counters has to be decreased at the same time. As two process constants cannot be rewritten by one wBPP rewrite rule, we introduce new process constants Y_1 and Y_2 to represent inverse elements to X_1 and X_2 respectively and we make a term $X_k^x \| Y_k^y$ to represent the counter c_k the value of which is $x - y$. We note that the weak state unit allows for controlling the correct order of the successive stages in the progress of a bisimulation game.

Moreover, we show that our undecidability results hold even for more restricted classes fcBPA and fcBPP and remain valid also for the normed versions of fcBPP and fcBPA. Hence, they hold for normed wBPP and normed wBPA as well.

We recall that the decidability of weak bisimilarity is an open question for BPA and BPP. We note that these problems are conjectured to be decidable (see [14] and [7] respectively) in which case our results would establish a fine undecidability border of weak bisimilarity.

Acknowledgements. We would like to thank Jiří Srba for valuable suggestions and comments.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[2] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.

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

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

[5] J. Esparza. Grammars as processes. In *Formal and Natural Computing*, volume 2300 of *LNCS*. Springer, 2002.

[6] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

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

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

[9] P. Jančar and J. Srba. Highly undecidable questions for process algebras. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science (TCS'04)*, Exploring New Frontiers of Theoretical Informatics, pages 507–520. Kluwer Academic Publishers, 2004.

[10] M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *LNCS*, pages 355–370. Springer-Verlag, 2004.

[11] M. Křetínský, V. Řehák, and J. Strejček. On extensions of process rewrite systems: Rewrite systems with weak finite-state unit. In Philippe Schnoebelen, editor, *INFINITY 2003: 5th International Workshop on Verification of Infinite-State Systems*, volume 98 of *Electronic Notes in Theoret. Computer Science*, pages 75–88. Elsevier Science Publishers, 2004.

[12] M. Křetínský, V. Řehák, and J. Strejček. Refining the Undecidability Border of Weak Bisimilarity. In *INFINITY 2005: 7th International Workshop on Verification of Infinite-State Systems*, 2005. to appear in ENTCS as Proc. of INFINITY 05.

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

[14] R. Mayr. Weak bisimilarity and regularity of context-free processes is EXPTIME-hard. *Theoretical Computer Science*, 330(3):553–575, 2005.

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

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

[17] F. Moller. Infinite results. In *Proc. of CONCUR'96*, volume 1119 of *LNCS*, pages 195–216. Springer, 1996.

[18] D. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoret. Computer Science*, 97(1–2):233–244, 1992.

[19] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of 17th POPL*, pages 232–245. ACM Press, 1990.

[20] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proceedings of 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 579–593. Springer-Verlag, 2002.

[21] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 13:567–587, 2003.

[22] 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.

[23] J. Srba. Roadmap of infinite results. In *Current Trends In Theoretical Computer Science, The Challenge of the New Century, Vol 2: Formal Models and Semantics*, pages 337–350. World Scientific Publishing Co., 2004. `http://www.brics.dk/~srba/roadmap/`.

[24] J. Strejček. Rewrite systems with constraints, EXPRESS'01. *Electronic Notes in Theoretical Computer Science*, 52, 2002.