Available online at www.sciencedirect.com

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 96 (2004) 153–172

www.elsevier.com/locate/entcs

Weak Bisimilarity and Regularity of Context-Free Processes is *EXPTIME*-hard

Richard Mayr

Department of Computer Science
Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee 51, D-79110 Freiburg, Germany.
E-mail: `mayrri@informatik.uni-freiburg.de`

Abstract

We show that checking weak bisimulation equivalence of two context-free processes (also called BPA-processes) is *EXPTIME*-hard, even under the condition that the processes are normed. Furthermore, checking weak regularity (finiteness up to weak bisimilarity) for context-free processes is *EXPTIME*-hard as well. Adding a finite control of the minimal non-trivial size of 2 to the BPA process already makes weak bisimilarity undecidable.

Keywords: Context-free processes, BPA, pushdown automata, bisimulation

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [12]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [2,16] for surveys). Here we consider the complexity of checking weak and strong bisimulation equivalence for several subclasses of pushdown automata (PDA). BPA (Basic Process Algebra; also called context-free processes) are equivalent to the subclass of pushdown automata where the finite control has size 1. The subclasses of *normed* PDA and BPA (denoted as nPDA and nBPA) satisfy the additional condition that from every reachable configuration it is possible to empty the stack. A normed PDA/BPA is called *totally normed* if the stack can always be emptied, but not by internal τ-actions alone.

doi:10.1016/j.entcs.2004.04.026

The state of the art: Strong bisimilarity was shown to be decidable for normed PDA by Stirling [21], and later for general PDA by Sénizergues [14]. However, no upper complexity bound for this problem is known so far. The best known lower bound is *EXPTIME*-hardness [8], which holds even for (totally) normed PDA. Weak bisimulation equivalence for (normed) PDA is undecidable [20].

The best known algorithm for strong bisimilarity of BPA requires doubly exponential time (and space) [3], while the best known lower bound is *PSPACE*-hardness [18]. However, strong bisimilarity of normed BPA is decidable in polynomial time [6]. The decidability of weak bisimilarity for BPA is an open question. The undecidability proof for PDA does not carry over to BPA, and it is widely conjectured that this problem is decidable for BPA. It is known that weak bisimilarity is decidable for the subclass of totally normed BPA [5]. The best known lower bound for weak bisimilarity of general BPA was *PSPACE*-hardness [23], while the best lower bound for weak bisimilarity of (totally) normed BPA was only $\mathcal{NP}$-hardness [23].

Our contribution. We show that weak bisimilarity of BPA is *EXPTIME*-hard, even for normed BPA. Then we use this result to show that the problem of deciding if a given BPA is weakly regular (finite up to weak bisimilarity) is also *EXPTIME*-hard. This improves a previously known *PSPACE* lower bound [18]. Furthermore, we show that adding a finite control of the minimal size of 2 to the BPA process already makes weak bisimilarity undecidable.

2 Definitions

BPA processes correspond to pushdown automata with a finite control of size 1. They can be described by $(1, S)$-PRS in the framework of process rewrite systems (PRS) [10]. Let $Act = \{\tau\} \cup \{a, b, c, \ldots\}$ and $Const = \{\epsilon\} \cup \{X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The action τ is a special 'silent' internal action and the special constant ϵ denotes the empty term. The class of *sequential process expressions* S is defined by $E ::= X \mid E.E$, where $X \in Const$ and '.' is a binary operator of sequential composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' is associative and 'ϵ' is a unit for '.'. (In particular it follows that $\epsilon.E = E$).

A BPA is a finite set Δ of *rules* which have the form $X \xrightarrow{a} E$, where $X \in Const - \{\epsilon\}$, $E \in S$, $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite).

Each BPA Δ defines a unique labeled transition system where states are process expressions. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules:

$$\frac{(E \xrightarrow{a} F) \in \Delta}{E \xrightarrow{a} F} \qquad \frac{E \xrightarrow{a} E'}{E.F \xrightarrow{a} E'.F}$$

We extend the notation $E \xrightarrow{a} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \xrightarrow{w} F$ for some $w \in Act^*$. A BPA-process (α, Δ) is given by an expression $\alpha \in S$ and a set of rules Δ. Δ is omitted if it is clear from the context. The length of a string of symbols α is denoted by $|\alpha|$.

Definition 2.1 The norm $\|\alpha\|$ of a BPA-process α is defined as the length of the shortest derivation sequence from α to ϵ, not counting τ-moves. A BPA Δ is **normed** if for every X in $Const(\Delta)$ we have $0 \leq \|X\| < \infty$. A BPA Δ is **totally normed** if for every X in $Const(\Delta) - \{\epsilon\}$ we have $0 < \|X\| < \infty$. We denote the classes of normed **BPA**, totally normed **BPA** and normed **PDA** processes by **nBPA**, **tnBPA** and **nPDA**, respectively.

We consider the semantic equivalences *weak bisimilarity* and *strong bisimilarity* [12] over labeled transition systems (e.g., those generated by BPA).

Definition 2.2 The *extended transition relation* '$\overset{a}{\Rightarrow}$' is defined by $E \overset{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \xrightarrow{\tau^i} E' \xrightarrow{a} E'' \xrightarrow{\tau^j} F$ for some $i, j \in \mathbb{N}_0$. A binary relation R over states in a labeled transition graph is a *weak bisimulation* iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \xrightarrow{a} E'$ then there is $F \overset{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \xrightarrow{a} F'$ then there is $E \overset{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. States E, F are *weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. (Sometimes weak bisimulation is defined with $\Rightarrow$ instead of $\rightarrow$ everywhere. However, the two definitions are equivalent.) Strong bisimulation is defined similarly with $\xrightarrow{a}$ instead of $\overset{a}{\Rightarrow}$ everywhere. E, F are *strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them.

Sometimes the following game theoretic characterization of bisimulation is more useful for reasoning about it. Bisimulation equivalence can be described by *bisimulation games* [22] between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

We consider the problem of weak bisimulation equivalence of normed context-free processes (nBPA).

nBPA $\approx$ nBPA (Weak bisimilarity of normed BPA)

Instance: A set of BPA rules Δ and two normed processes α and α'.
Question: $(\alpha, \Delta) \approx (\alpha', \Delta)$?

We show an *EXPTIME*-lower bound for this problem by a polynomial-time reduction from the *EXPTIME*-complete acceptance problem of alternating linear-bounded automata (alternating LBA).

Definition 2.3 An *alternating LBA* (alternating linear-bounded automaton) [4] is a tuple $\mathcal{M} = (S, \Sigma, \hat{\gamma}, \gamma, s_0, \vdash, \dashv, \pi)$ S is a finite set of control-states, s_0 is the initial control-state, Σ is the set of tape symbols, $\vdash, \dashv \in \Sigma$ are the left-end and right-end markers, respectively ($\vdash$ and $\dashv$ cannot be overwritten or moved), and $\pi : S \to \{\forall, \exists, acc, rej\}$ is a function which partitions the control states of S into *universal*, *existential*, *accepting*, and *rejecting*, respectively. The computation step function $\hat{\gamma} : S \times \Sigma \to 2^{S\times\Sigma\times\{-1,0,1\}}$ takes as input a control-state and the symbol under the head and returns a set of possible steps containing each: a new control state, the symbol to be written and the instruction where to move the head (left, same place, right). For a more convenient notation we define the function γ as the restriction of $\hat{\gamma}$ to the (successor) control-states, i.e., $\gamma : S \times \Sigma \to 2^S$ with $\gamma(s, A) := \{s' \in S \mid \exists B, x. (s', B, x) \in \hat{\gamma}(s, A)\}$.

We assume (w.l.o.g.) that γ has the following properties:

- for all $s \in S$ and $A \in \Sigma$ such that $\pi(s) = \forall$ or $\pi(s) = \exists$ we have that $|\gamma(s, A)| = 2$ (i.e., $\gamma(s, A) = \{s_1, s_2\}$ for some $s_1, s_2 \in S$). The first element of $\gamma(s, A)$ is denoted by *first*(s, A), and the second one by *second*(s, A). It means that each configuration of $\mathcal{M}$ where the control state is universal or existential has exactly two immediate successors (configurations reachable in one computation step).
- for all $s \in S$ and $A \in \Sigma$ such that $\pi(s) = acc$ or $\pi(s) = rej$ we have that $\gamma(s, A) = \emptyset$, i.e., each configuration of $\mathcal{M}$ where the control state is accepting or rejecting is 'terminated' (without any successors).

A *computation tree* for $\mathcal{M}$ on a word $w \in \Sigma^*$ is a finite tree T satisfying the following: the root of T is (labeled by) the initial configuration $s_0{\vdash}w{\dashv}$ of $\mathcal{M}$, and if N is a node of $\mathcal{M}$ labeled by a configuration usv where $u, v \in \Sigma^*$ and $s \in S$, then the following holds:

- if s is accepting or rejecting, then T is a leaf;

- if s is existential, then T has one successor whose label is one of the two configurations reachable from usv in one step (here, the notion of a computation step is defined in the same way as for 'ordinary' Turing machines);
- if s is universal, then T has two successors labeled by the two configurations reachable from usv in one step.

$\mathcal{M}$ *accepts* w iff there is a computation tree T such that all leaves of T are accepting configurations. The acceptance problem for alternating LBA is known to be *EXPTIME*-complete [4].

Remark 2.4 We assume that all branches of a computation tree of an alternating LBA M on an input word w have a finite length, which is at most exponential in $|w|$. This is no restriction, since there are at most $|\Sigma|^{|w|}|S||w|$ different configurations. For any alternating LBA M one could construct an equivalent one with the required property by counting the number of computation steps on the tape (using only linear space).

3 The Idea

We prove *EXPTIME*-hardness of the **nBPA** $\approx$ **nBPA** problem by a reduction of the acceptance problem of alternating LBA to it. First, we describe the general ideas of our construction in an informal way.

For an alternating LBA $M = (S, \Sigma, \hat{\gamma}, \gamma, s_0, \vdash, \dashv, \pi)$ with input word w (of length n) we construct in polynomial time a normed BPA Δ and two processes α and α' s.t. M accepts w iff $\alpha \not\approx \alpha'$. Thus we reduce the *EXPTIME*-complete problem of alternating LBA acceptance [4] to weak non-bisimilarity of BPA processes. This shows *EXPTIME*-hardness of weak bisimilarity of BPA processes, since the class *EXPTIME* is closed under complement.

We represent an LBA configuration as a sequence usv where $u \in \Sigma^*$ is the content of the tape to the left of the head, s is the control-state, and $v \in \Sigma^+$ is the content of the tape under the head and to the right of it. The construction ensures that the bisimulation game on α, α' proceeds in three phases.

Phase 1: First, a sequence of LBA configurations is pushed onto the stack in both processes. The attacker determines the tape symbols (from Σ) of these configurations and the successor control-states (from S) if the current control-state is existential. The defender determines the successor control-states (from S) if the current control-state is universal. (This is achieved by an application of the so-called 'existential quantification technique' (also called 'defender power technique') which is due to Jančar [7]; a more explicit formulation is due to Srba [17].) This construction does not guarantee that this sequence of configurations really is a correct branch of a computation tree of the LBA. However, the rest of the construction ensures that the attacker can win if and only if the sequence of configurations is a correct branch of a computation tree that ends in an accepting LBA configuration.

Phase 2: At an accepting LBA configuration the attacker can do the special action 'f' in process α and thus enter the next phase. Action 'f' is not immediately enabled in the process α'. Thus, the defender can only reply by popping the whole content of the stack of process α' (by τ-actions) and rebuilding the stack content (again by τ-actions). However, only incorrect branches of a computation tree of the LBA can be generated in this way. This long defender-move ends with the visible action 'f'. This means that if the stack content of process α (on the attacker's side) is not a correct accepting branch of a computation tree of the LBA then the defender can make the two processes syntactically equal (and thus weakly bisimilar). Otherwise the two resulting processes will not be syntactically equal (and also not weakly bisimilar as the next phase will show).

Phase 3: In the last phase the content of the stack will be popped by characteristic actions for each symbol. So, in this phase, the current processes will be weakly bisimilar if and only if they are syntactically equivalent.

Thus, the only way to win for the attacker is to push a sequence of LBA configurations onto the stack that represents a correct accepting branch of a computation tree of the LBA. If the attacker can do this despite the defender's interference in choosing the successor control-states at the universal control-states, then the attacker will win. In any other case the game will either continue forever, or the defender can make the two processes syntactically equal and so the defender will win. Thus, the attacker can win iff M accepts w.

4 The Construction

For an alternating LBA $M = (S, \Sigma, \hat{\gamma}, \gamma, s_0, \vdash, \dashv, \pi)$ with input word w (of length n), we construct (in polynomial time) a normed BPA Δ with $\Gamma := Const(\Delta)$ and $Act := Act(\Delta)$ and processes α, α' s.t. $\mathcal{M}$ accepts w iff $\alpha \not\approx \alpha'$. We represent an LBA configuration as a sequence usv where $u \in \Sigma^*$ is the content of the tape to the left of the head, s is the control-state, and $v \in \Sigma^+$ is the content of the tape under the head and to the right of it.

The set of constants Γ of the BPA is defined as

$$\Gamma := T \cup \Sigma \cup S \cup \{B, C, G, F\}$$

where T (the top symbols) is a set of additional constants that occur only at the top of the stack in the processes α and α'. T is defined as follows: Let $S' :=$

$\{s' \mid s \in S\}$. $T := T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5 \cup T_6 \cup T_7$, where $T_1 := S \times \{0, \ldots, n-1\}$ (used in process α for pushing new LBA configurations. LBA control-state and number of symbols pushed already), $T_2 := (S \times \Sigma) \times \{1, \ldots, n\}$ (used in process α for pushing new LBA configurations. LBA control-state, tape symbol under the head and number of symbols pushed already), $T_3 := S' \times \{0, \ldots, n-1\}$ (like T_1, but for process α'), $T_4 := (S' \times \Sigma) \times \{1, \ldots, n\}$ (like T_2, but for process α'), $T_5 := \{(\tilde{s}, 0) \mid s \in S\}$ (used in choosing the successor control-state), $T_6 := (S \times S) \times \{0\}$ (used to store two possible successor control-states), $T_7 := \{(E, j, i) \mid 1 \leq j \leq |ERROR|, 0 \leq i \leq n-3\}$ (used by the defender player in phase 2 to push a sequence of LBA configurations with a least one error onto the stack). (The set $ERROR$ will be defined later.)

For every state $s \in S$, the states s' and $\tilde{s}$ are seen as being associated to s. $Act := \Sigma \cup S \times \Sigma \cup S \cup \{a, b, c, f, \tau\}$.

The initial configurations α, α' are defined as follows: $\alpha := ((s_0, A), n)s_0wB$ and $\alpha' := ((s_0', A), n)s_0wB$ where $A \in \Sigma$ is the first symbol of w (i.e., $w = Aw'$ for some w'). This means that we start with the first configuration s_0w already on the stack. B is the bottom-symbol. The first symbol $((s_0, A), n)$ encodes the facts that the LBA control-state is s_0 and A is the symbol under the head, and the number n means that the complete configuration (of length n) is present on the stack.

The set of transitions Δ is defined in several steps. First we define the transition rules for phase 1: Intuitively, the rules 1–6 push a new configuration onto the stack. With the rules 7–8 the attacker chooses the next control-state if the current one is existential. With the rules 9–19 the defender chooses the next control-state if the current one is universal.

The weak bisimulation game starts with the processes α, α' where the first configuration is already on the stack. So the first thing that happens is the choice of the next control-state by rules 7–8 or 9–19.

1.	$(s,i) \xrightarrow{X} (s,i+1)X$	for all $s \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
2.	$(s,i) \xrightarrow{(s,Y)} ((s,Y),i+1)sY$	for all $s \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;
3.	$((s,Y),i) \xrightarrow{X} ((s,Y),i+1)X$	for all $s \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
4.	$(s',i) \xrightarrow{X} (s',i+1)X$	for all $s \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
5.	$(s',i) \xrightarrow{(s,Y)} ((s',Y),i+1)sY$	for all $s \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;
6.	$((s',Y),i) \xrightarrow{X} ((s',Y),i+1)X$	for all $s \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
7.	$((s,Y),n) \xrightarrow{s_1} (s_1,0)$	if $\pi(s) = \exists$ and $s_1 \in \gamma(s,Y)$;
8.	$((s',Y),n) \xrightarrow{s_1} (s'_1,0)$	if $\pi(s) = \exists$ and $s_1 \in \gamma(s,Y)$;
9.	$((s,Y),n) \xrightarrow{a} (\tilde{s}_1,0)$	if $\pi(s) = \forall$ and $s_1 = \mathit{first}(s,Y)$;
10.	$((s,Y),n) \xrightarrow{a} (\tilde{s}_2,0)$	if $\pi(s) = \forall$ and $s_2 = \mathit{second}(s,Y)$;
11.	$((s,Y),n) \xrightarrow{a} ((s_1,s_2),0)$	if $\pi(s) = \forall$, $s_1 = \mathit{first}(s,Y)$ and $s_2 = \mathit{second}(s,Y)$;
12.	$((s',Y),n) \xrightarrow{a} (\tilde{s}_1,0)$	if $\pi(s) = \forall$ and $s_1 = \mathit{first}(s,Y)$;
13.	$((s',Y),n) \xrightarrow{a} (\tilde{s}_2,0)$	if $\pi(s) = \forall$ and $s_2 = \mathit{second}(s,Y)$;
14.	$((s_1,s_2),0) \xrightarrow{s_1} (s_1,0)$	
15.	$((s_1,s_2),0) \xrightarrow{s_2} (s_2,0)$	
16.	$(\tilde{s}_1,0) \xrightarrow{s_1} (s'_1,0)$	
17.	$(\tilde{s}_1,0) \xrightarrow{s_2} (s_2,0)$	
18.	$(\tilde{s}_2,0) \xrightarrow{s_1} (s_1,0)$	
19.	$(\tilde{s}_2,0) \xrightarrow{s_2} (s'_2,0)$	

Now we define the transition rules for phase 2.

In this phase we need to make it possible to generate all incorrect branches of a computation tree of the LBA. This is possible, since any incorrect branch of a computation tree can be characterized as containing at least one error in a computation step, which can be detected locally, i.e., by comparing 3 symbols of an LBA configuration with the 3 symbols at the same tape location at the previous LBA configuration. We now define the set *ERROR* that describes all these possible errors in computations. It consists of elements of the form $(\mathit{err}_1, \mathit{err}_2)_i$, (with $\mathit{err}_1, \mathit{err}_2 \in (\Sigma \cup S)^3$) that are pairs of strings of symbols that occur at the same position of successive LBA configurations.

The set $ERROR$ contains exactly those pairs that cannot occur in correct computations of the LBA M. Thus $ERROR$ depends on M. To define this formally, we first construct the set $CORR$ of all correct pairs of this form, i.e., the complement of $ERROR$.

$$CORR := \{(\alpha(i)\alpha(i{+}1)\alpha(i{+}2), \beta(i)\beta(i{+}1)\beta(i{+}2)) \mid \alpha \rightarrow_M \beta, 0 \leq i \leq n-2\}$$

where α, β are configurations of M (of length $n+1$, since the control-state is stored as well), $\alpha(i)/\beta(i)$ is the symbol at position i of α/β and $\alpha \rightarrow_M \beta$ means that β is a successor-configuration of α, according to the computation of M.

$$\begin{aligned} ERROR &:= \{(err_1, err_2)_1, \ldots, (err_1, err_2)_m\} \\ &:= ((\Sigma \cup S)^3 \times (\Sigma \cup S)^3) - CORR \end{aligned}$$

Thus, $|ERROR| = m \leq (|\Sigma| + |S|)^6$.

20. $((s, Y), n) \xrightarrow{\tau} \epsilon$ if $\pi(s) = rej$

21. $((s', Y), n) \xrightarrow{\tau} \epsilon$ if $\pi(s) = rej$

22. $((s, Y), n) \xrightarrow{f} C$ if $\pi(s) = acc$

23. $((s, Y), n) \xrightarrow{\tau} \epsilon$ if $\pi(s) = acc$

24. $((s', Y), n) \xrightarrow{\tau} \epsilon$ if $\pi(s) = acc$

25. $Z \xrightarrow{\tau} \epsilon$ for all $Z \in \Sigma \cup S$

26. $B \xrightarrow{\tau} GB$

27. $G \xrightarrow{\tau} GZ$ for all $Z \in \Sigma \cup S$

28. $G \xrightarrow{\tau} (E, j, 0)err_1$ for all $(err_1, err_2)_j \in ERROR$

29. $(E, j, i) \xrightarrow{\tau} (E, j, i+1)Z$ for all $Z \in \Sigma \cup S$, $0 \leq i \leq n-3$

30. $(E, j, n-2) \xrightarrow{\tau} F err_2$ for all $(err_1, err_2)_j \in ERROR$

31. $F \xrightarrow{\tau} FZ$ for all $Z \in \Sigma \cup S$

32. $F \xrightarrow{f} C$

Note that a series of τ-actions by rules 26–31 can push arbitrarily long sequences of symbols onto the stack. However, by definition of the set $ERROR$, none of these sequences represents a correct branch of a computation tree of the LBA M. Furthermore, this sequence of τ-actions must always end with the visible action 'f' (by rule 32). Finally, we define the transitions for phase 3.

33. $C \xrightarrow{c} \epsilon$

34. $Z \xrightarrow{Z} \epsilon$ for all $Z \in \Sigma \cup S$

35. $B \xrightarrow{b} \epsilon$

5 The Proof

We assume now that the alternating LBA M, the input word w, the BPA Δ and the processes α and α' are defined as in Section 4.

Lemma 5.1 *The following properties are equivalent.*

(i) *M accepts w.*

(ii) *Starting with processes α, α', the attacker has a strategy in the weak bisimulation game to enforce (whatever the defender does) that the game reaches a configuration of processes $\beta = ((s, Y), n)\gamma B$, $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$ and γ is a sequence of LBA configurations that describes a correct accepting branch of a computation tree of M.*

Proof. In the process α, by the rules 1–3, LBA configurations of length $n+1$ are pushed onto the stack (n tape symbols plus one symbol for the control-state). The control-state of the simulated LBA is stored in the top symbol of the stack. After the control-state has been pushed (by rule 2), it is stored in the top symbol of the stack together which the tape symbol under the head of the (simulated) LBA. Exactly the same is done in the process α' by rules 4–6. (The defender is forced to copy the attacker's moves. In this phase the only difference between α and α' is in the top symbol of the stack.) The attacker determines which tape symbols are pushed onto the stack.

After a configuration has been pushed onto the stack, the successor control-state is determined. If the current control-state is existential ($\pi(s) = \exists$) then the attacker determines the successor control-state by rule 7 (or 8). The defender must imitate this move by rule 8 (or 7 resp.). If the current control-state is universal ($\pi(s) = \forall$) then the attacker must apply rule 11 in process α. In any other case (if the attacker uses rule 9 or 10 in α or rule 12 or 13 in α') the defender can make the two processes syntactically equal (by rule 9,10,12 or 13) in the same round and wins. The defender then chooses the successor control-state by applying either rule 12 or rule 13 in α'. We now assume that the defender chose rule 12, i.e., control-state s_1. The other case is symmetric. Then the top symbols of α and α' are $((s_1, s_2), 0)$ and $(\tilde{s}_1, 0)$, respectively. The attacker must now play a rule with action s_1 (either 14 in α or 16 in α') which yields the new top symbols $(s_1, 0)$ and $(s_1', 0)$ of α and α', respectively. If the attacker chooses rule 15 or 17 then the two processes become syntactically equal and the defender wins. The effect of this construction is, that at universal control-states s it is the defender who chooses between the two possible successor control-states s_1 and s_2.

$\Rightarrow$ If M accepts w then, by Definition 2.3, there is a computation tree of M on w where all branches end with an accepting leaf. By the construction above the attacker selects the tree and the defender selects the branch. By pushing only correct successor configurations of the LBA onto the stack, the attacker creates a stack content γ that corresponds to a branch in the accepting tree. The defender can only copy his moves in the other process. Thus a configuration of processes $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$ is reached, where $\pi(s) = acc$ and γ is a sequence of LBA configurations that describes a correct accepting computation of M.

$\Leftarrow$ If M does not accept w then, by Definition 2.3, every computation tree has at least one branch that does not accept. By the construction above the defender can select the successor control-states at the universal nodes in such a way that a non-accepting branch of the computation-tree is chosen. Therefore, if the attacker chooses to push the right tape symbols onto the stack (s.t. a correct branch of a computation tree of M is pushed onto the stack) this will end in a rejecting configuration. However, the attacker might also introduce errors into the computation by pushing wrong LBA configurations onto the stack and thus reach a pair of configurations $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$ and γ is **not** a correct branch of a computation tree of the LBA. In either case it is impossible for the attacker to enforce a configuration of processes $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$ and γ is a sequence of LBA configurations that describes a correct accepting branch of a computation tree of M.

□

Lemma 5.2 *Let $\gamma, \gamma' \in (\Sigma \cup S)^*$. We have $\gamma B \approx \gamma' B$ if and only if $\gamma = \gamma'$*

Proof. The 'if' direction is trivial. For the 'only if' direction assume that $\gamma \neq \gamma'$ and (without restriction) $|\gamma| \geq |\gamma'|$. We prove $\gamma B \not\approx \gamma' B$ by induction on $|\gamma'|$.

In the base case $|\gamma'| = 0$ and $|\gamma| > 0$, since $\gamma \neq \gamma'$. Thus $\gamma B \xrightarrow{Z} \delta B$ (by rule 34) for some $Z \in \Sigma \cup S$, but $\gamma' B = B \stackrel{Z}{\not\rightarrow}$. So the attacker wins and $\gamma B \not\approx \gamma' B$.

For the induction step the attacker plays $\gamma B \xrightarrow{Z} \delta B$ by rule 34 (i.e., he removes the top symbol Z of γ). The defender can respond in two ways. First, if the top symbol of γ' is the same Z as that of γ then the defender can play $\gamma' B \xrightarrow{Z} \delta' B$ by rule 34. It follows that $\delta \neq \delta'$ and $|\delta'| = |\gamma'| - 1$. By induction hypothesis $\delta B \not\approx \delta' B$. Second, if the symbol Z occurs at some other position in γ' then the defender can play $\gamma' B \overset{Z}{\Rightarrow} \delta' B$ (by rules 25. and 34.). However, in this case $|\delta'| < |\gamma'| - 1$ while $|\delta| = |\gamma| - 1$. Therefore $|\delta'| < |\delta|$ and thus $\delta' \neq \delta$. By induction hypothesis again $\delta B \not\approx \delta' B$. (Using the rules 26–32 to generate new symbols on the stack is not an option for the defender, since this always ends with the visible action 'f'.) Thus, the attacker has a winning strategy and $\gamma B \not\approx \gamma' B$. □

Lemma 5.3 *$B \xrightarrow{\tau^*}\xrightarrow{f} C\gamma B$ if and only if $\gamma \in (\Sigma \cup S)^*$ is not a correct path in a computation tree of M.*

Proof. Directly from the definition of the set *ERROR* and the rules 26–32.□

Lemma 5.4 *Let $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$ be processes, where $\pi(s) = acc$. We have $\beta \not\approx \beta'$ if and only if γ is a correct branch of a computation tree of the LBA M.*

Proof.

⇒ We assume that γ is not a correct branch of a computation tree of M and show that $\beta \approx \beta'$ under this condition. The defender has the following winning strategy. If the attacker plays $\beta = ((s, Y), n)\gamma B \xrightarrow{\tau} \gamma B$ or $\beta' = ((s', Y), n)\gamma B \xrightarrow{\tau} \gamma B$ then the defender can make the two processes syntactically equal in the same round of the game and wins. If the attacker plays $\beta = ((s, Y), n)\gamma B \xrightarrow{f} C\gamma B$ then the defender can reply by $\beta' = ((s', Y), n)\gamma B \xrightarrow{\tau^*} B \xrightarrow{\tau^*}\xrightarrow{f} C\gamma B$ by Lemma 5.3, because γ is not a correct branch of a computation tree of M. Again the two processes are syntactically equal and the defender wins. Thus $\beta \approx \beta'$.

⇐ If γ is a correct branch of a computation tree of the LBA M then the attacker has the following winning strategy. The attacker plays the move $\beta = ((s, Y), n)\gamma B \xrightarrow{f} C\gamma B$. The defender can only reply by popping the whole stack (by rules 24 and 25), pushing a new sequence of symbols onto the stack (by rules 26–31) and finally doing action 'f' (by rule 32). So the defender can only play $\beta' = ((s', Y), n)\gamma B \xrightarrow{\tau^*} B \overset{f}{\Rightarrow} C\gamma' B$. However, by Lemma 5.3, γ' does not represent a correct computation of M and thus $\gamma' \neq \gamma$.

We assume w.l.o.g. that $|\gamma| \geq |\gamma'|$. (The other case is symmetric.) Then the attacker plays $C\gamma B \xrightarrow{c} \gamma B$. If the defender uses the rule 26 in his reply then the result will be $C\gamma' B \overset{c}{\Rightarrow} \delta B$ for some δ s.t. $\delta \overset{h}{\not\Rightarrow}$ for all $h \neq f$ and thus the attacker can win in next move. Therefore the defender can only reply $C\gamma' B \overset{c}{\Rightarrow} \gamma'' B$, where γ'' is a suffix of γ' (by rules 33 and 25). Thus $\gamma \neq \gamma''$ and, by Lemma 5.2, $\gamma B \not\approx \gamma'' B$ and so the attacker wins.

Therefore the attacker has a winning strategy and $\beta \not\approx \beta'$.

□

Lemma 5.5 *Let α, α' be the processes from Section 4. $\alpha \not\approx \alpha'$ if and only if M accepts w.*

Proof.

$\Leftarrow$ If M accepts w then the attacker has the following winning strategy. By Lemma 5.1 the attacker has a strategy in the weak bisimulation game to reach a configuration of processes $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$ and γ is a sequence of LBA configurations that describes a correct branch of a computation tree of M. (It must be a correct accepting computation branch on w, since it starts with w and $\pi(s) = acc$). By Lemma 5.4 $\beta \not\approx \beta'$ and the attacker wins. Thus $\alpha \not\approx \alpha'$.

$\Rightarrow$ If M does not accept w then the defender has a winning strategy. By Lemma 5.1 the defender has a strategy to **avoid** a configuration $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$ and γ is a sequence of LBA configurations that describes a correct branch of a computation tree of M. The defender plays according to this strategy until one of the following two situations occurs: If the top symbols correspond to a rejecting state then only rules 20 and 21 are applicable, the processes become syntactically equal and the defender wins. If the top symbols correspond to accepting states then the two current processes must have the form $\beta = ((s, Y), n)\gamma B$ and $\beta' = ((s', Y), n)\gamma B$, where $\pi(s) = acc$. By Lemma 5.1 γ does not represent a correct branch of an accepting computation tree of M and thus by Lemma 5.4 $\beta \approx \beta'$. So the defender always wins and $\alpha \approx \alpha'$.

□

Lemma 5.6 *Let Δ be the BPA from Section 4, depending on the alternating LBA M. It is normed for every M.*

Proof. All constants $Z \in \Sigma \cup S$ have norm 0, because of rule 25. Constants B and C have norm 1, because of rules 35 and 33. $\|F\| = \|C\| + 1 = 2$, by rule 32. $\|G\| = 2$, because of rules 28–30, 32, 33 and 25. All constants $t \in T$ have finite norm, because by Remark 2.4 all correct branches of a computation tree of M have finite length. It is possible to push a correct branch of a computation tree of M onto the stack. This sequence will end in an accepting or a rejecting state after an at most exponential number of steps by Remark 2.4. Thus all constants in T have finite norm, because of rules 1–24, 33 and 25. Therefore all constants in Γ have finite norm. □

Note that our system Δ is normed, but not totally normed (see Def. 2.1), because some constants have norm 0.

Theorem 5.7 *Weak bisimulation equivalence for normed BPA is EXPTIME-hard.*

Proof. By reduction of the acceptance problem for alternating LBA. For any alternating LBA M with input word w we construct the BPA Δ and processes α, α' as in Section 4. By Lemma 5.6 the BPA is normed and by Lemma 5.5 $\alpha \not\approx \alpha'$ if and only if M accepts w. □

6 The Regularity Problem

The regularity problem is to check if a given infinite-state process is regular (i.e., finite) up-to a given notion of semantic equivalence.

Weak regularity of BPA

Instance: A BPA process (P, Δ).
Question: Does there exist a finite-state system F s.t. $(P, \Delta) \approx F$?

We show that weak regularity of BPA is *EXPTIME*-hard by a modification of the construction of Section 4. The problem with the construction from Section 4 is, that it is possible to generate infinitely many different wrong branches of a computation tree (of the LBA) on the stack, which are all pairwise non-weakly-bisimilar. Thus our processes α, α' are always infinite (i.e., non-regular) up-to weak bisimilarity. Here we modify our construction to make the processes regular up-to weak bisimilarity. (The idea is to make all the processes containing wrong computation sequences (of the LBA) weakly bisimilar to each other.) Then we use a standard reduction (from [15]) from weak bisimilarity of weakly regular processes to the weak regularity problem. However, unlike in Section 4, our modified processes are not normed.

The modification of the construction is done as follows:

- We replace the old rule 25: $Z \xrightarrow{\tau} \epsilon$ (for all $Z \in \Sigma \cup S$) by the new rule 25: $Z \xrightarrow{\tau} GB$ (for all $Z \in \Sigma \cup S$).
- We replace the old rule 35: $B \xrightarrow{b} \epsilon$ by the new rule 35: $B \xrightarrow{b} B$. (Thus the new system is no longer normed, since $\|B\| = \infty$.)
- We add the new rule number 36: $F \xrightarrow{\tau} \epsilon$.

Let Δ' be this new modified set of rules. We obtain that $\gamma B \approx \gamma' B$ for all branches of a computation tree γ, γ' (of the alternating LBA) that contain at least one error, because all sequences with errors can be generated by τ-actions. Formally, we show a modified version of Lemma 5.2 for the new system Δ' as follows:

Lemma 6.1 *Let $\gamma, \gamma' \in (\Sigma \cup S)^*$. If γ and γ' are incorrect branches of a* *computation tree (of the alternating LBA) then $\gamma B \approx \gamma' B$. If γ is a correct branch of a computation tree and $\gamma' \neq \gamma$ then $\gamma B \not\approx \gamma' B$.*

Proof.

(i) Let $\gamma, \gamma' \in (\Sigma \cup S)^*$ be incorrect branches of computation trees (of the alternating LBA). Then the defender has the following winning strategy in the weak bisimulation game starting at $\gamma B, \gamma' B$. Let $Z, Z' \in \Sigma \cup S$ be the first symbols in γ, γ' respectively, i.e., $\gamma = Z\delta$ and $\gamma' = Z'\delta'$. (Z, Z' exist, since the empty sequence is a correct branch of a computation tree, because it does not contain any wrong steps.) There are only two possibilities for the attacker's move:

(a) If the attacker plays $\gamma B = Z\delta B \xrightarrow{\tau} GB\delta B$ by the new rule 25 then the defender responds by $\gamma' B = Z'\delta' B \xrightarrow{\tau} GB\delta' B$ (by rule 25), and vice versa. Since $\|B\| = \infty$ we have $GB\delta B \approx GB \approx GB\delta' B$ and thus the defender can win.

(b) If the attacker plays $\gamma B = Z\delta B \xrightarrow{Z} \delta B$ (by rule 34) then the defender responds by a long move: First $\gamma' B = Z'\delta' B \xrightarrow{\tau} GB\delta' B$ by rule 25. Then $GB\delta' B \xrightarrow{\tau^*} F\gamma B\delta' B$ by rules 27–31 (this is possible, because γ is an incorrect branch of a computation tree of the alternating LBA). Then $F\gamma B\delta' B \xrightarrow{\tau} \gamma B\delta' B = Z\delta B\delta' B$ (by the new rule 36). Finally, $Z\delta B\delta' B \xrightarrow{Z} \delta B\delta' B$ by rule 34. Since $\|B\| = \infty$ we have $\delta B \approx \delta B\delta' B$ and thus the defender can win.

The other case where the attacker plays $\gamma' B = Z'\delta' B \xrightarrow{Z'} \delta' B$ is symmetric, because γ' is also an incorrect branch of a computation tree of the alternating LBA.

Thus the defender has a winning strategy and so $\gamma B \approx \gamma' B$.

(ii) Now we assume that γ is a correct computation sequence and $\gamma' \neq \gamma$. We show that $\gamma B \not\approx \gamma' B$ by induction on $|\gamma|$. We describe a winning strategy for the attacker in the weak bisimulation game.

In the base case $|\gamma| = 0$ and $|\gamma'| > 0$, since $\gamma \neq \gamma'$. Thus $\gamma B = B \xrightarrow{b} B$ (by rule 35), but $\gamma' B \stackrel{b}{\not\rightarrow}$. So the attacker wins and $\gamma B \not\approx \gamma' B$.

For the induction step $|\gamma| \geq 1$ and thus $\gamma = Z\delta$ for some $Z \in (\Sigma \cup S)$ and $\delta \in (\Sigma \cup S)^*$. In the special case that $|\gamma'| = 0$ we have $\gamma' B = B \xrightarrow{b} B$ (by rule 35), but $\gamma B = Z\delta B \stackrel{b}{\not\rightarrow}$. So the attacker wins and $\gamma B \not\approx \gamma' B$. So we now assume $|\gamma'| \geq 1$ and thus $\gamma' = Z'\delta'$ for some $Z' \in (\Sigma \cup S)$ and $\delta' \in (\Sigma \cup S)^*$. The attacker plays $\gamma B = Z\delta B \xrightarrow{Z} \delta B$ (by rule 34). The defender can respond in two ways:

(a) If $Z = Z'$ then the defender can do $\gamma' B = Z'\delta' B = Z\delta' B \xrightarrow{Z} \delta' B$ (by rule 34). Then δ is still a correct branch of a computation tree of the LBA and $\delta' \neq \delta$. Furthermore, $|\delta| < |\gamma|$. Thus, by induction hypothesis $\delta B \not\approx \delta' B$ and so the attacker can win.

(b) In any case (if $Z = Z'$ or not) the defender can do $\gamma' B = Z'\delta' B \overset{Z}{\Rightarrow} \delta'' B \delta' B$ by rules 27–31, 36 and 34, where δ'' is an incorrect branch of a computation tree of the LBA and $|\delta''| > 0$. We get $\delta'' B \delta' B \approx \delta'' B$. Again we have that δ is a correct branch of a computation tree, and thus $\delta'' \neq \delta$ and furthermore $|\delta| < |\gamma|$. By induction hypothesis $\delta B \not\approx \delta'' B$ and so the attacker can win.

Thus the attacker has a winning strategy and $\gamma B \not\approx \gamma' B$.

□

Theorem 6.2 *Weak regularity of BPA is EXPTIME-hard.*

Proof. We use the modified set of rules Δ' as described above. By Lemma 6.1 all the infinitely many incorrect branches of computation trees are weakly bisimilar to each other. Since, by Remark 2.4, all correct branches of computation trees have exponentially bounded length, it follows that the processes α, α' (defined as in Section 4) in the new system Δ' are finite up to weak bisimilarity, i.e., weakly regular. However, they still satisfy the property $\alpha \not\approx \alpha'$ iff M accepts w, because

- Lemma 5.1 carries over directly to Δ'.
- Lemma 5.2 is replaced by Lemma 6.1 for Δ'.
- Lemma 5.3 carries over directly to Δ'.
- Lemma 5.4 carries over to Δ'. The proof must be modified slightly to use Lemma 6.1 instead of Lemma 5.2. Also the long moves of popping the whole stack (by rule 25 in Δ) till the bottom symbol B and generating a new stack content (by rules 26–31) are modified. By the new rule 25 in Δ' the stack content is not popped but cut off by the introduction of a new bottom symbol B, which cannot be removed in Δ'. (Remember that in Δ' we have $\|B\| = \infty$ and thus $\gamma B \delta \approx \gamma B$ for all γ, δ.)
- Lemma 5.5 carries over directly to Δ'.

Note that Lemma 5.6 does **not** carry over to Δ', since $\|B\| = \infty$.

So we obtain that $(\alpha, \Delta'), (\alpha', \Delta')$ are weakly regular and $(\alpha, \Delta') \not\approx (\alpha', \Delta')$ if and only if M accepts w.

Then we can apply the general polynomial time reduction from weak bisimilarity of weakly regular BPA (which we have just shown to be *EXPTIME*-hard) to the weak regularity problem for BPA from [15] and obtain our result.□

7 Undecidability for Two Control-States

BPA correspond to the subclass of pushdown automata with a finite control of size 1. Undecidability of weak bisimilarity for general (normed) pushdown automata has been shown by Srba in [20] by a reduction from Minsky 2-counter machines. The reason why this proof does not carry over to BPA is that BPA lack a global finite control and thus cannot remember control-information when decreasing the stack. (However, while increasing the stack, the top stack symbol can be used to encode a control-state.) The question arises how many control-states are needed in pushdown automata to make weak bisimilarity undecidable. It follows directly from the construction in [20] that a certain fixed number suffices, since one can apply this reduction to the fixed universal Minsky machine, but the number obtained in this way is certainly much bigger than 2.

However, weak bisimilarity is already undecidable for pushdown automata with only 2 control-states (a very weak extension of BPA). This can be shown by a straightforward adaption of the technique used in Srba's proof of undecidability of weak bisimilarity for PA-processes [19] (although PDA and PA are incomparable).

Theorem 7.1 *Weak bisimilarity is undecidable for normed pushdown automata with only 2 control-states.*

Proof. (sketch) Consider an instance of Post's correspondence problem (PCP) with two sets of words $A = \{u_1, \dots, u_n\}$ and $B = \{v_1, \dots, v_n\}$ where $u_i, v_i \in \Sigma^+$. It is undecidable whether there exist finitely many indices $i_1, \dots, i_m \in \{1, \dots, n\}$ s.t. $u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_m}$ [13]. One can effectively construct a PDA with only 2 control-states p_1, p_2 s.t. $p_1 X \approx p_1 X'$ iff the PCP instance has a solution. For every word u_i one defines a symbol U_i and transition rules s.t. $p_1 U_i \xrightarrow{i} p_1$ and (by using some intermediate stack symbols corresponding to suffixes of u_i) $p_2 U_i \xrightarrow{u_i} p_2$. Similarly for v_i and V_i. One can construct the bisimulation game in such a way that the defender first pushes (by τ-moves) an arbitrarily long sequence $U_{i_1} \dots U_{i_m}$ onto the stack of the left process and then in the next round an arbitrarily long sequence $V_{j_1} \dots V_{j_{m'}}$ onto the stack of the right process. During these operations the control-state will always be p_1, since the top stack symbol can be used to store control-information instead. After these operations the processes will have the form $p_1 Y U_{i_1} \dots U_{i_m}$ and $p_1 Y' V_{j_1} \dots V_{j_{m'}}$. Then one ads the rules $p_1 Y \xrightarrow{a} p_1$, $p_1 Y' \xrightarrow{a} p_1$, $p_1 Y \xrightarrow{b} p_2$, $p_1 Y' \xrightarrow{b} p_2$. It is easy to see that $p_1 U_{i_1} \dots U_{i_m} \approx p_1 V_{j_1} \dots V_{j_{m'}}$ iff $m = m'$ and $i_k = j_k$ for all $k \in \{1, \dots, m\}$. On the other hand $p_2 U_{i_1} \dots U_{i_m} \approx p_2 V_{j_1} \dots V_{j_{m'}}$ iff $u_{i_1} \dots u_{i_m} = v_{j_1} \dots v_{j_{m'}}$. The configuration represents a solution of the PCP iff both these conditions are satisfied. (Remember that it was the defender who chose the sequences $U_{i_1} \dots U_{i_m}$ and $V_{j_1} \dots V_{j_{m'}}$.) However, it is the attacker in the bisimulation game who decides which of the two conditions is checked, by choosing either action 'a' or action 'b' and thus either control-state p_1 or p_2. Therefore we have $p_1 X \approx p_1 X'$ iff $p_1 Y U_{i_1} \dots U_{i_m} \approx p_1 Y' V_{j_1} \dots V_{j_{m'}}$ iff the PCP has a solution. The constructed PDA is trivially normed. □

Remark 7.2 Note that in this undecidability proof the stack of the PDA is first only increased and then only decreased. In other words, there is only one reversal between stack-increasing and stack-decreasing mode. (The proof in [20] used an arbitrary number of reversals.) Thus, weak bisimilarity is also undecidable for normed 1-reversal-bounded PDA with just two control-states.

Remark 7.3 It has been shown in [11] that weak bisimilarity is undecidable even for (normed) 1-counter machines (just one-counter instead of a stack). In fact, it is undecidable for an even weaker model: one-counter nets, i.e, Petri nets with just one unbounded place. However, one-counter nets/machines and BPA are incomparable in their expressive power.

It has also been shown in [11] that a fixed number of control-states in 1-counter machines suffices to make weak bisimilarity undecidable. The exact number of control-states required is unknown, but it is very unlikely that just 2 should suffice as for PDA.

8 Conclusion

We have shown an *EXPTIME* lower bound for the problem of checking weak bisimulation equivalence for general and normed BPA. However, our proof does not carry over to the class of totally normed BPA. The following table summarizes the known complexity results about strong and weak bisimilarity on PDA and BPA. New results are in boldface. (For an extensive list of results on the complexity of bisimulation checking for infinite-state systems see [16].)

	(n)PDA	**BPA**
$\sim$	decidable [14] *EXPTIME*-hard [8]	in 2-*EXPTIME*[3] *PSPACE*-hard [18]
$\approx$	undecidable [20]	? **EXPTIME-hard**

	nBPA	**tnBPA**
$\sim$	P-complete [6,1]	P-complete [6,1]
$\approx$	? **EXPTIME-hard**	decidable [5] $\mathcal{NP}$-hard [23]

It is interesting to compare these results with the results on checking bisimilarity with finite-state systems. For example, checking strong and weak bisimilarity between (normed) PDA and finite-state systems is *PSPACE*-complete [8] and checking weak (and strong) bisimilarity between BPA and finite-state systems is polynomial [9].

Furthermore, we have shown a new *EXPTIME* lower bound for weak regularity of BPA. For normed BPA, the best known lower bound for weak regularity is still $\mathcal{NP}$-hardness [15]. For totally normed BPA, weak regularity coincides with boundedness, which is **NLogspace**-complete [18].

References

[1] Balcazar, J., J. Gabarro and M. Santha, *Deciding bisimilarity is P-complete*, Formal Aspects of Computing **4** (1992), pp. 638–648.

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

[3] Burkart, O., D. Caucal and B. Steffen, *An elementary bisimulation decision procedure for arbitrary context-free processes*, in: *MFCS'95*, LNCS **969** (1995).

[4] Chandra, A. K., D. C. Kozen and L. J. Stockmeyer, *Alternation*, Journal of the ACM **28** (1981), pp. 114–133.

[5] Hirshfeld, Y., *Bisimulation trees and the decidability of weak bisimulations*, Electronic Notes in Theoretical Computer Science (ENTCS) **5** (1997).

[6] Hirshfeld, Y., M. Jerrum and F. Moller, *A polynomial algorithm for deciding bisimilarity of normed context-free processes*, Theoretical Computer Science **158** (1996), pp. 143–159.

[7] Jančar, P., *High undecidability of weak bisimilarity for Petri nets*, in: *Proceedings of CAAP'95*, LNCS **915** (1995), pp. 349–363.

[8] Kučera, A. and R. Mayr, *On the complexity of semantic equivalences for pushdown automata and BPA*, , **2420** (2002), pp. 433–445.

[9] Kučera, A. and R. Mayr, *Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time*, Theoretical Computer Science **270** (2002), pp. 667–700.

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

[11] Mayr, R., *Undecidability of weak bisimulation equivalence for 1-counter processes*, in: *Proc. of ICALP 2003*, LNCS **2719** (2003).

[12] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[13] Post, E. L., *A variant of a recursively unsolvable problem*, Bulletin of the American Mathematical Society **52** (1946), pp. 264–268.

[14] Sénizergues, G., *Decidability of bisimulation equivalence for equational graphs of finite out-degree*, in: *Proc. of FOCS'98* (1998).

[15] Srba, J., *Complexity of weak bisimilarity and regularity for BPA and BPP*, Electronic Notes in Theoretical Computer Science (ENTCS) **39** (2000), proceedings of the 7th International Workshop on Expressiveness in Concurrency (EXPRESS'00).

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

[17] Srba, J., *Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard*, in: *Proceedings of STACS 2002*, LNCS **2285** (2002), pp. 535–546.

[18] Srba, J., *Strong bisimilarity and regularity of basic process algebra is PSPACE-hard*, in: *Proc. of ICALP 2002*, LNCS **2380** (2002), pp. 716–727.

[19] Srba, J., *Undecidability of weak bisimilarity for PA-processes*, in: *Proceedings of 6th International Conference on Developments in Languague Theory (DLT'02)*, LNCS (2002), to appear.

[20] Srba, J., *Undecidability of weak bisimilarity for pushdown processes*, in: *Proc. of CONCUR 2002*, LNCS **2421** (2002), pp. 579–593.

[21] Stirling, C., *Decidability of bisimulation equivalence for normed pushdown processes*, Theoretical Computer Science **195** (1998), pp. 113–131.

[22] Stirling, C., *The joys of bisimulation*, in: *Proc. of MFCS'98*, LNCS **1450** (1998), pp. 142–151.

[23] Stříbrná, J., *Hardness results for weak bisimilarity of simple process algebras*, Electronic Notes in Theoretical Computer Science (ENTCS) **18** (1998).