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.

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.

Key words: 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.

Email address: `mayrri@informatik.uni-freiburg.de` (Richard Mayr).
URL: `http://www.informatik.uni-freiburg.de/~mayrri` (Richard Mayr).

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 [19].

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 *NP*-hardness [23].

Our contribution. We show that weak bisimilarity of BPA is *EXPTIME*-hard, even for normed BPA. Then we use a slightly modified variant of 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.

Outline of the paper. In Section 2 we introduce basic concepts like BPA and bisimulation. In Section 3 we informally describe the main idea of reducing the acceptance problem of alternating linear-bounded automata to weak bisimilarity of BPA. In Section 4 we describe the reduction formally, while in Section 5 we prove its correctness and the main result. In Section 6 we modify the construction to show an *EXPTIME* lower bound for the regularity problem for BPA, and in Section 7 we prove that extending BPA with only 2 control-states already makes weak bisimilarity undecidable. The paper ends with a section that summarizes the results and related work.

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 $X \in Const \setminus \{\epsilon\}$, $E \in S$, $a \in Act$. $Const(\Delta)$ 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: $\dfrac{(E \xrightarrow{a} F) \in \Delta}{E \xrightarrow{a} F}$ $\qquad \dfrac{E \xrightarrow{a} E'}{E.F \xrightarrow{a} E'.F}$

We extend the notation $E \xrightarrow{a} F$ to elements of $Act(\Delta)^*$ in a standard way. Moreover, we say that F is *reachable* from E if $E \xrightarrow{w} F$ for some $w \in Act(\Delta)^*$. 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 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 $Const(\Delta) \setminus \{\epsilon\}$ we have $0 < \|X\| < \infty$. A PDA is* normed *if for every reachable configuration $p\alpha$, where p is a control-state and α a sequence of stack symbols, there exists some control-state p' s.t. $p\alpha \rightarrow^* p'$. 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 *The* extended transition relation *'$\stackrel{a}{\Rightarrow}$' is defined by $E \stackrel{a}{\Rightarrow} F$ 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 system is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \xrightarrow{a} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \xrightarrow{a} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ 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 $\stackrel{a}{\Rightarrow}$ 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 normed BPA system Δ and two 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 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 (without loss of generality) 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 $|\hat{\gamma}(s, A)| = 2$. 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). This directly implies that $\gamma(s, A) = \{s_1, s_2\}$ 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).*
- *For all $s \in S$ and $A \in \Sigma$ such that $\pi(s) = acc$ or $\pi(s) = rej$ we have*

that $\hat{\gamma}(s,A)=\emptyset$, 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 4 *One can assume without restriction that the computation-tree of an alternating LBA has finite depth. More precisely, 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 $|\Sigma|^{|w|}|S|(|w|+2)$ different configurations (tape contents, control-states and positions of the head). For any alternating LBA M one could construct an equivalent one with this 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 a special action 'f' in process α and thus enter the next phase. The 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 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 $Const(\Delta)$ of the BPA is defined as

$$Const(\Delta) := T \cup \Sigma \cup S \cup \{B, C, G, F\}$$

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 at 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. $Act(\Delta) := \Sigma \cup S \times \Sigma \cup S \cup \{a, b, c, f, \tau\}$.

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.	$(s,i) \xrightarrow{X} (s,i+1)X$	$\forall s \in S,\ X \in \Sigma,\ 0 \le i \le n-2;$
2.	$(s,i) \xrightarrow{(s,Y)} ((s,Y),i+1)sY$	$\forall s \in S,\ Y \in \Sigma,\ 0 \le i \le n-1;$
3.	$((s,Y),i) \xrightarrow{X} ((s,Y),i+1)X$	$\forall s \in S,\ X,Y \in \Sigma,\ 0 \le i \le n-1;$
4.	$(s',i) \xrightarrow{X} (s',i+1)X$	$\forall s \in S,\ X \in \Sigma,\ 0 \le i \le n-2;$
5.	$(s',i) \xrightarrow{(s,Y)} ((s',Y),i+1)sY$	$\forall s \in S,\ Y \in \Sigma,\ 0 \le i \le n-1;$
6.	$((s',Y),i) \xrightarrow{X} ((s',Y),i+1)X$	$\forall s \in S,\ X,Y \in \Sigma,\ 0 \le i \le n-1;$
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 $(err_1, err_2)_i$, (with $err_1, err_2 \in (\Sigma \cup S)^3$) 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.

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

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 represent 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 *The following properties are equivalent.*

(1) M accepts w.
(2) 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 with 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 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 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. $\square$

Lemma 6 *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 we have $|\gamma'| > 0$ and thus $|\gamma| > 0$. Therefore $\gamma = Z\delta$ for some $Z \in \Sigma \cup S$ and $\delta \in (\Sigma \cup S)^*$ with $|\delta| = |\gamma| - 1$. The attacker plays $\gamma B = Z\delta B \xrightarrow{Z} \delta B$ 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 (i.e., $\gamma' = Z\delta'$ for some $\delta' \in (\Sigma \cup S)^*$) then the defender can play

$\gamma' B = Z\delta' B \xrightarrow{Z} \delta' B$ 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 $\gamma' B \stackrel{Z}{\Rightarrow} \delta' B$ (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 7 *$B \xrightarrow{\tau^*} \xrightarrow{f} C\gamma B$ (and $GB \xrightarrow{\tau^*} \xrightarrow{f} C\gamma B$) 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 and 27–32, respectively. □

Lemma 8 *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.

$\Rightarrow$ 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 7, 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 $\beta \approx \beta'$.

$\Leftarrow$ 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 $\beta' = ((s', Y), n)\gamma B \xrightarrow{\tau^*} B \stackrel{f}{\Rightarrow} C\gamma' B$. However, by Lemma 7, 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 $C\gamma' B \stackrel{c}{\Rightarrow} \delta B$ for some δ s.t. $\delta \stackrel{h}{\not\Rightarrow}$ for all $h \neq f$ (i.e., f is the only possible next visible action of δ) and thus the attacker can win in next move. Therefore the defender can only reply $C\gamma' B \stackrel{c}{\Rightarrow} \gamma'' B$, where γ'' is a suffix of γ' (by rules 33 and 25). Thus $\gamma \neq \gamma''$ and, by Lemma 6, $\gamma B \not\approx \gamma'' B$ and so the attacker wins.

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

Lemma 9 *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 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 8 $\beta \not\approx \beta'$ 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 the defender has a strategy by which he can 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 γ does not represent a correct branch of an accepting computation tree of M and thus by Lemma 8 $\beta \approx \beta'$. So the defender always wins and $\alpha \approx \alpha'$. $\square$

Lemma 10 *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 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 4. Thus all constants in T have finite norm, because of rules 1–24, 33 and 25. Therefore all constants in $Const(\Delta)$ have finite norm. $\square$

Note that our system Δ is normed, but not totally normed (see Def. 1), because some constants have norm 0. (Weak bisimilarity of totally normed BPA is known to be decidable [5] and *NP*-hard [23]. See also the table of results in Section 8.)

Theorem 11 *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 10 the BPA is normed and by Lemma 9 $\alpha \not\approx \alpha'$ if and only if M accepts w. The result follows, since the class *EXPTIME* is closed under complement. $\square$

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.

<u>WEAK REGULARITY OF BPA</u>
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) as stack content 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 (see Remark 13 why this is necessary).

Let Δ be a set of BPA rules 1–35 defined in Section 4. We now construct a modified set of rules Δ' as follows.

- We replace the old rule 22:

$$((s, Y), n) \xrightarrow{f} C$$

by the new rule 22:

$$((s, Y), n) \xrightarrow{f} \epsilon$$

- We replace the old rule 25:

$$Z \xrightarrow{\tau} \epsilon \qquad \text{(for all } Z \in \Sigma \cup S)$$

by the new rule 25:

$$Z \xrightarrow{\tau} GB \qquad \text{(for all } Z \in \Sigma \cup S)$$

- We replace the old rule 32:

$$F \xrightarrow{f} C$$

by the new rule 32:

$$F \xrightarrow{f} \epsilon$$

- 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 1–36. The rest of this section always refers to this set of rules Δ'.

Remark 12 *Since the constant B is unnormed w.r.t. the set of rules Δ' (unlike for Δ) it can never be removed. Thus we obtain that $\forall \alpha, \beta.\, \alpha B \beta \approx \alpha B$.*

Remark 13 Intuitively, the main difference between Δ and Δ' is the following. In Δ' all the infinitely many different reachable configurations that store wrong computations of the LBA must be weakly bisimilar to each other. This is achieved by making it possible for the defender to generate any wrong computation sequence (of the LBA) on the stack *at any time in phase 3*, not just in phase 2 (by using the new rule 25 and 27–31). However, this extra generation phase must not end with a special action, thus the new rule 36. In Δ the whole stack is first popped (by the old rule 25) before the new content is generated. In Δ' instead the whole previous content is made insignificant by pushing a new bottom symbol B onto the stack (which cannot be removed). Thus Δ' is no longer normed.

Why can one not just pop the whole stack as in Δ ? The problem is that in Δ' (unlike in Δ) the generation phase does not necessarily end with a visible action (because of rule 36). So if popping the stack by τ-actions was allowed (as in the old rule 25) then the defender could generate not only incorrect computation sequences (of the LBA) on the stack, but also correct ones. How could this be done ? First the defender would generate an incorrect computation sequence (of the LBA) on the stack (by τ-actions) and then he could pop the part of it that contains the error (again by τ-actions), leaving only the correct part. (This behavior is prevented in Δ by the explicit visible generation-phase end-actions in rules 32 and 33., but possible in Δ' because of rule 36.) Since generating correct computations of the LBA would fundamentally spoil the construction, it must be prevented. Thus the modified unnormed Δ' where the stack cannot be popped by τ-actions.

So our construction to show an *EXPTIME* lower bound for the weak regularity problem for BPA does not apply to the subclass of *normed* BPA. So far the best known lower bound for normed BPA is only *NP*-hardness (see Section 8).

We will now show 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 6 for the new system Δ' as follows.

Lemma 14 *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.

(1) 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. Thus, altogether the defender's move is $\gamma' B \overset{Z}{\Rightarrow} \delta B\delta' B$. 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 $\gamma B \approx \gamma' B$.

(2) 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 $|\gamma|$ (the length of γ). 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 $\gamma' B \overset{b}{\not\Rightarrow}$. 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 $\gamma B = Z\delta B \overset{b}{\not\Rightarrow}$. 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 partial 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 $\gamma B \not\approx \gamma' B$. □

Now we show that in Δ' our processes are finite up-to weak bisimilarity.

Lemma 15 *Let the processes α, α' be defined as in Section 4. Then both (α, Δ') and (α', Δ') are finite up-to weak bisimilarity.*

PROOF. We show that (α, Δ') is finite up-to weak bisimilarity. The proof for (α', Δ') is analogous. By Remark 4, all correct branches of computation trees (of the alternating LBA) have exponentially bounded length. Thus, there exists a number k s.t. all configurations longer than k must contain (a part of) an incorrect branch of the computation tree. Therefore, by Lemma 14, if $\gamma, \gamma' \in (\Sigma \cup S)^*$ and $|\gamma| \geq k$ and $|\gamma'| \geq k$ then $\gamma B \approx \gamma' B$. The reachable configurations of process (α, Δ') can only have the following forms: (We ignore the suffixes of the configurations after the first symbol B, since they are irrelevant w.r.t. weak bisimilarity.)

- $t\gamma B$ with $t \in T$ and $\gamma \in (\Sigma \cup S)^*$.
  If $\gamma, \gamma' \in (\Sigma \cup S)^*$ and $|\gamma| \geq k$ and $|\gamma'| \geq k$ then $t\gamma B \approx t\gamma' B$. The winning strategy for the defender is as follows. Just imitate all moves of the attacker until the top stack symbol is removed. Then the resulting pair of processes will be $\gamma''\gamma B$ and $\gamma''\gamma' B$ for some $\gamma'' \in (\Sigma \cup S)^*$. We have $\gamma''\gamma B \approx \gamma''\gamma' B$ as shown above, because $|\gamma''\gamma| \geq k$ and $|\gamma''\gamma'| \geq k$. Therefore, there exist at most $|T| * |\Gamma|^{k-1} + |T|$ configurations of this form which are different w.r.t. weak bisimilarity.
- γB with $\gamma \in (\Sigma \cup S)^*$.
  By the same argument as above, there are at most $|\Gamma|^{k-1}+1$ configurations of this form which are different w.r.t. weak bisimilarity.
- $x\gamma B$ with $x \in T_7 \cup \{B, G, F\}$.
  If $\gamma, \gamma' \in (\Sigma \cup S)^*$ and $|\gamma| \geq k$ and $|\gamma'| \geq k$ then $x\gamma B \approx x\gamma' B$. The winning strategy for the defender is as follows. Just imitate all moves of the attacker until the top stack symbol is removed. Then the resulting pair of processes will be $\gamma''\gamma B$ and $\gamma''\gamma' B$ for some $\gamma'' \in (\Sigma \cup S)^*$. We have $\gamma''\gamma B \approx \gamma''\gamma' B$ as shown above. Therefore, there exist at most $(|T_7| + 3) * |\Gamma|^{k-1} + |T_7| + 3$ configurations of this form which are different w.r.t. weak bisimilarity.

Thus, there are only finitely many reachable configurations which are different w.r.t. weak bisimilarity. Therefore, the processes (α, Δ') (and analogously also process (α', Δ')) is finite up-to weak bisimilarity, i.e., weakly regular. $\square$

Now we adapt the lemmas and proofs from Section 5 to the new system Δ'.

Remark 16 *Lemma 5 carries over directly to Δ'. This is because it refers only to phase 1 of the weak bisimulation game, and none of the modified/added rules can be used in this phase.*

A modified variant of Lemma 7 holds for Δ'.

Lemma 17 *$GB \stackrel{\tau^*}{\rightarrow}\stackrel{f}{\rightarrow} \gamma B$ for $\gamma \in (\Sigma \cup S)^*$ holds if and only if γ is not a correct path in a computation tree of M.*

PROOF. Directly from the definition of Δ', the set *ERROR* and the rules 27–32. (Note that no application of the new rule 25 is possible, because this would result in more than one B.) $\square$

We now show that Lemma 8 carries over to Δ'.

Lemma 18 *Let $\beta = ((s,Y),n)\gamma B$ and $\beta' = ((s',Y),n)\gamma B$ be processes, where $\pi(s) = acc$. We have $(\beta,\Delta') \not\approx (\beta',\Delta')$ if and only if γ is a correct branch of a computation tree of the alternating LBA M.*

PROOF.

$\Rightarrow$ 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 $\beta = ((s,Y),n)\gamma B \xrightarrow{f} \gamma B$ (by rule 22) then the defender can reply as follows. First the defender does $\beta' = ((s',Y),n)\gamma B \xrightarrow{\tau} \gamma B$ by rule 24. Let $\gamma = Z\delta$ for some symbol Z and sequence δ. (We can assume that γ is not empty, because the empty sequence would represent a correct branch of the computation-tree.) Then the defender does $\gamma B = Z\delta B \xrightarrow{\tau} GB\delta B$ by the new rule 25. Then the defender does $GB\delta B \xrightarrow{\tau^*}\xrightarrow{f} \gamma B\delta B$ by Lemma 17, because γ is not a correct branch of a computation tree of M. Since B is unnormed we have $\gamma B\delta B \approx \gamma B$. So altogether the defender's move is $\beta' \stackrel{f}{\Rightarrow} \gamma B\delta B \approx \gamma B$ and thus the defender wins. Thus $\beta \approx \beta'$.

$\Leftarrow$ 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 $\beta = ((s,Y),n)\gamma B \xrightarrow{f} \gamma B$ (by rule 22). The defender must reply by a $\stackrel{f}{\Rightarrow}$ move. The only possible one is as follows. First $\beta' = ((s',Y),n)\gamma B \xrightarrow{\tau} \gamma B$ by rule 24. Then, if $\gamma = Z\delta$ for some Z and δ, one does $\gamma B = Z\delta B \xrightarrow{\tau} GB\delta B$ by rule 25. Otherwise, if γ is empty, one uses rule 26 for $B \xrightarrow{\tau} GB$. In any case the result is weakly bisimilar to GB. Then one can only push a new sequence of symbols onto the stack (by τ-moves using rules 27–31). (It is possible to remove the F symbol by rule 36 and then apply rule 25 again, but the result is weakly bisimilar to GB again, because B is unnormed. Since this does not change anything, we consider only the last application of rule 25.) Finally the defender does action 'f' (by rule 32). So the defender can only play $\beta' = ((s',Y),n)\gamma B \stackrel{f}{\Rightarrow} \lambda \approx \gamma' B$ (λ is some configuration which is weakly bisimilar to $\gamma' B$). However, by Lemma 17, γ' does not represent a correct branch of a computation tree of M and thus $\gamma' \neq \gamma$.

So the weak bisimulation game now stands at the configurations (weakly bisimilar to) γB and $\gamma' B$ where γ is a correct branch of a computation tree of the LBA M and $\gamma' \neq \gamma$. By Lemma 14 we get $\gamma B \not\approx \gamma' B$ and thus the attacker can win. Therefore the attacker has a winning strategy and $\beta \not\approx \beta'$. $\square$

A Lemma similar to Lemma 9 also holds for Δ'.

Lemma 19 *Let α, α' be the processes from Section 4. $(\alpha, \Delta') \not\approx (\alpha', \Delta')$ if and only if M accepts w.*

PROOF.

$\Leftarrow$ If M accepts w then the attacker has the following winning strategy. By Lemma 5 (which also holds for Δ' by Remark 16) 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 18 $\beta \not\approx \beta'$ 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 (which also holds for Δ' by Remark 16) 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 γ does not represent a correct branch of an accepting computation tree of M and thus by Lemma 18 $\beta \approx \beta'$. So the defender always wins and $\alpha \approx \alpha'$. $\square$

Lemma 20 *Weak bisimilarity of weakly regular BPA is EXPTIME-hard.*

PROOF. By reduction from the *EXPTIME*-complete acceptance problem of alternating LBA. For every alternating LBA M with input word w one constructs in polynomial time the set of BPA rules Δ' and the processes α, α' from Section 4. By Lemma 15 the processes (α, Δ') and (α', Δ') are finite up-to weak bisimilarity, i.e., weakly regular. By Lemma 19 $(\alpha, \Delta') \not\approx (\alpha', \Delta')$ if and only if M accepts w. The result follows, since the class *EXPTIME* is closed under complement. $\square$

The following general reduction from weak bisimilarity of weakly regular BPA to weak regularity of BPA was shown by Srba in [15].

Lemma 21 *(Srba [15])*
The weak bisimilarity problem for weakly regular BPA can be reduced in polynomial time to the weak regularity problem for BPA.

Theorem 22 *Weak regularity of BPA is EXPTIME-hard.*

PROOF. The result follows directly from Lemma 20 and Lemma 21. □

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 [19] 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 [19] 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 [20] (although PDA and PA are incomparable).

We show the undecidability of weak bisimilarity of PDA with 2 control-states by a reduction from Post's correspondence problem, which is known to be undecidable [13].

Post's correspondence problem (PCP)

Instance: A non-unary alphabet Σ and two sets of words $A = \{u_1, \ldots, u_n\}$ and $B = \{v_1, \ldots, v_n\}$ where $u_i, v_i \in \Sigma^+$.
Question: Do there exist finitely many indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ s.t. $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$?

For every instance of PCP we now construct a PDA with just 2 control-states and two configurations α, α' s.t. $\alpha \approx \alpha'$ iff the instance of PCP has a solution.

The set of control-states of our PDA is $\{p, q\}$, the set of stack symbols is $\Gamma := \{U_i \mid 1 \leq i \leq n\} \cup \{V_i \mid 1 \leq i \leq n\} \cup \{T, T', T_1, T_1', T_2, T_2', G_l, G_r\}$ and the set of actions is $\{a, b, c, \tau\} \cup \{1, \ldots, n\} \cup \Sigma$. The two PDA processes α, α' are defined with initial configurations $\alpha := pTB$ and $\alpha' := pT'B$. The transition rules of the PDA are defined as follows.

1. $pT \xrightarrow{a} pT_1$

2. $pT \xrightarrow{\tau} pG_r$

3. $pT' \xrightarrow{\tau} pG_r$

4. $pG_r \xrightarrow{\tau} pG_rV_i$ for all $i \in \{1, \ldots, n\}$

5. $pG_r \xrightarrow{a} pT_1'$

6. $pT_1 \xrightarrow{a} pG_l$

7. $pT_1' \xrightarrow{a} pG_lB$

8. $pT_1' \xrightarrow{a} pT_2'$

9. $pG_l \xrightarrow{\tau} pG_lU_i$ for all $i \in \{1, \ldots, n\}$

10. $pG_l \xrightarrow{\tau} pT_2$

11. $pT_2 \xrightarrow{a} pT_3$

12. $pT_2' \xrightarrow{a} pT_3'$

13. $pT_3 \xrightarrow{b} p$

14. $pT_3' \xrightarrow{b} p$

15. $pT_3 \xrightarrow{c} q$

16. $pT_3' \xrightarrow{c} q$

17. $pU_i \xrightarrow{i} p$ for all $i \in \{1, \ldots, n\}$

18. $pV_i \xrightarrow{i} p$ for all $i \in \{1, \ldots, n\}$

19. $qU_i \xrightarrow{u_i} q$ for all $i \in \{1, \ldots, n\}$

20. $qV_i \xrightarrow{v_i} q$ for all $i \in \{1, \ldots, n\}$

Remark 23 *In the rules 19. and 20. one uses not a single action but a whole string in Σ^+. This is just a short notation which can also be expressed by standard rules. For example, let $u_i = u_i^1 \ldots u_i^k$. To express rule 19 one can define new constants $U_i^1, \ldots U_i^{k-1}$ and rules $qU_i \xrightarrow{u_i^1} qU_i^1$, $qU_i^j \xrightarrow{u_i^{j+1}} qU_i^{j+1}$ (for*

$1 \leq j \leq k-2$) and $qU_i^{k-1} \xrightarrow{u_i^k} q$. Similarly for V_i.*

Remark 24 *Note that the stack symbol B cannot be removed from the stack. In particular this implies that $p\gamma B\delta \approx p\gamma B$ and $q\gamma B\delta \approx q\gamma B$ for all γ, δ.*

Lemma 25 *Let $m, m' \in \mathbb{N}$ and $i_1, \ldots, i_m \in \mathbb{N}$ and $j_1, \ldots, j_{m'} \in \mathbb{N}$. Then $pU_{i_1} \ldots U_{i_m} B \approx pV_{j_1} \ldots V_{j_{m'}} B$ iff $m = m'$ and $\forall k \in \{1, \ldots, m\}. i_k = j_k$.*

PROOF. Directly from the definition of rules 17 and 18. □

Lemma 26 *Let $m, m' \in \mathbb{N}$ and $i_1, \ldots, i_m \in \mathbb{N}$ and $j_1, \ldots, j_{m'} \in \mathbb{N}$. Then $qU_{i_1} \ldots U_{i_m} B \approx qV_{j_1} \ldots V_{j_{m'}} B$ iff $u_{i_1} \ldots u_{i_m} = v_{j_1} \ldots v_{j_{m'}}$.*

PROOF. Directly from the definition of rules 19 and 20. □

Lemma 27 *Let $m, m' \in \mathbb{N}$ and $i_1, \ldots, i_m \in \mathbb{N}$ and $j_1, \ldots, j_{m'} \in \mathbb{N}$. Then $pT_3 U_{i_1} \ldots U_{i_m} B \approx pT_3' V_{j_1} \ldots V_{j_{m'}} B$ iff $m = m'$ and $\forall k \in \{1, \ldots, m\}. i_k = j_k$ and $u_{i_1} \ldots u_{i_m} = v_{j_1} \ldots v_{j_{m'}}$.*

PROOF. Directly from the definition of rules 13–16, and Lemma 25 and Lemma 26. □

Lemma 28 *Let $\alpha := pTB$ and $\alpha' := pT'B$. Then $\alpha \approx \alpha'$ iff the PCP has a solution.*

PROOF.

(1) We assume that the PCP has a solution and show that in this case the defender in the weak bisimulation game has a universal winning strategy. Let the solution of the PCP be $i_1, \ldots, i_m \in \{1, \ldots, n\}$ s.t. $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. We start at configuration α, α'. The attacker must play $\alpha \xrightarrow{a} pT_1 B$ (by rule 1), otherwise the defender could make the two processes syntactically equal in the same round. The defender can respond by a long move $\alpha' = pT'B \overset{a}{\Rightarrow} pT_1' V_{i_1} \ldots V_{i_m} B$ (by rules 3,4,5). If the attacker now uses rule 6. (or 7.) then the defender can use rule 7. (or 6. resp.) to make the two processes weakly bisimilar and wins. (Here we use the fact that symbol B cannot be removed; see Remark 24.) Thus the attacker must play $pT_1' V_{i_1} \ldots V_{i_m} B \xrightarrow{a} pT_2' V_{i_1} \ldots V_{i_m} B$ by rule 8. The defender can respond by the long move $pT_1 B \overset{a}{\Rightarrow} pT_2 U_{i_1} \ldots U_{i_m} B$ by rules 6.,9. and 10. Now if the attacker plays $pT_2 U_{i_1} \ldots U_{i_m} B \xrightarrow{a} pT_3 U_{i_1} \ldots U_{i_m} B$ by rule 11., then the defender replies $pT_2' V_{i_1} \ldots V_{i_m} B \xrightarrow{a} pT_3' V_{i_1} \ldots V_{i_m} B$ by rule 12.,

and vice-versa. By Lemma 27 we have $pT_3U_{i_1}\dots U_{i_m}B \approx pT'_3V_{i_1}\dots V_{i_m}B$ and thus the defender can win. Therefore $\alpha \approx \alpha'$.

(2) Now we assume that the PCP does not have a solution and show that the attacker has a universal winning strategy. First the attacker plays $\alpha = pTB \xrightarrow{a} pT_1B$ by rule 1. The defender can only reply by a move $\alpha' = pT'B \overset{a}{\Rightarrow} pT'_1V_{j_1}\dots V_{j_{m'}}B$ for some $m' \in \mathbb{N}$ and $j_1,\dots,j_{m'} \in \mathbb{N}$ by rules 3–5. Then the attacker plays $pT'_1V_{j_1}\dots V_{j_{m'}}B \xrightarrow{a}\xrightarrow{a} pT'_3V_{j_1}\dots V_{j_{m'}}B$ by rules 8. and 12. The defender can only reply by a move $pT_1B \overset{a}{\Rightarrow}\overset{a}{\Rightarrow} pT_3U_{i_1}\dots U_{i_m}B$ for some $m \in \mathbb{N}$ and $i_1,\dots,i_m \in \mathbb{N}$ by rules 6.,9.,10. and 11. Since the PCP does not have a solution, it follows that $i_1,\dots,i_m \neq j_1,\dots,j_{m'}$ or $u_{i_1}\dots u_{i_m} \neq v_{j_1}\dots v_{j_{m'}}$. Thus, by Lemma 27, we have $pT_3U_{i_1}\dots U_{i_m}B \not\approx pT'_3V_{j_1}\dots V_{j_{m'}}B$ and the attacker can win. Therefore we obtain $\alpha \not\approx \alpha'$. □

Theorem 29 *Weak bisimilarity is undecidable for pushdown automata with only 2 control-states.*

PROOF. For every instance of PCP, a PDA with only 2 control-states and processes α, α' can be effectively constructed as shown above. Then the result follows from the undecidability of PCP and Lemma 28. □

Remark 30 The pushdown automata used in the construction above are not normed, since the symbol B can never be removed from the stack. This feature is used in rule 7. to 'cut off' the rest of the stack below the symbol B. One attempt for an alternative solution with a normed PDA would be to make it possible to remove the whole stack content by τ-actions instead. However, this would require at least one extra control-state. Using either state p or q for this task would break the whole construction later in the rules 17.–20.

A simple way of showing undecidability of weak bisimilarity for *normed* PDA with 3 control-states is to modify the above construction as follows. Add a new control-state r and rules $p \xrightarrow{d} r$, $q \xrightarrow{d} r$ and $rX \xrightarrow{\tau} r$ for all $X \in \Gamma$. It is easy to see that this new PDA still satisfies $\alpha \approx \alpha'$ iff the PCP has a solution. It is trivially normed, but has 3 control-states. The decidability of weak bisimilarity for *normed* PDA with 2 control-states is left open.

Remark 31 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 [19] used an arbitrary number of reversals.) Thus, weak bisimilarity is also undecidable for 1-reversal-bounded PDA with just two control-states.

Remark 32 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 [19]	? **EXPTIME-hard**

	nBPA	**tnBPA**
$\sim$	P-complete [6,1]	P-complete [6,1]
$\approx$	? **EXPTIME-hard**	decidable [5] *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 *NP*-hardness [15]. For totally normed BPA, weak regularity coincides with boundedness, which is NLogspace-complete [18].

Finally, we have shown that even a slight extension of BPA, adding a finite-control of the minimal non-trivial size 2, already makes weak bisimilarity undecidable.

References

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

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

[3] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[4] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

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

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

[7] P. Jančar. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of CAAP'95*, volume 915 of *LNCS*, pages 349–363. Springer, 1995.

[8] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *Proc. of MFCS 2002*, volume 2420, pages 433–445. Springer Verlag, 2002.

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

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

[11] R. Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In *Proc. of ICALP 2003*, volume 2719 of *LNCS*. Springer Verlag, 2003.

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

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

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

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

[17] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proceedings of STACS 2002*, volume 2285 of *LNCS*, pages 535–546. Springer, 2002.

[18] J. Srba. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard. In *Proc. of ICALP 2002*, volume 2380 of *LNCS*, pages 716–727. Springer Verlag, 2002.

[19] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proc. of CONCUR 2002*, volume 2421 of *LNCS*, pages 579–593. Springer Verlag, 2002.

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

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

[22] C. Stirling. The joys of bisimulation. In *Proc. of MFCS'98*, volume 1450 of *LNCS*, pages 142–151. Springer Verlag, 1998.

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