On the Complexity of Semantic Equivalences for Pushdown Automata and BPA

Antonín Kučera[⋆1] and Richard Mayr[2]

[1] Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic,
tony@fi.muni.cz
[2] Department of Computer Science, Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee 51, D-79110 Freiburg, Germany.
mayrri@informatik.uni-freiburg.de

Abstract. We study the complexity of comparing pushdown automata (PDA) and context-free processes (BPA) to finite-state systems, w.r.t. strong and weak simulation preorder/equivalence and strong and weak bisimulation equivalence. We present a complete picture of the complexity of all these problems. In particular, we show that strong and weak simulation preorder (and hence simulation equivalence) is $EXPTIME$-complete between PDA/BPA and finite-state systems in both directions. For PDA the lower bound even holds if the finite-state system is fixed, while simulation-checking between BPA and any fixed finite-state system is already polynomial. Furthermore, we show that weak (and strong) bisimilarity between PDA and finite-state systems is $PSPACE$-complete, while strong (and weak) bisimilarity between two PDAs is $EXPTIME$-hard.

1 Introduction

Transition systems are a fundamental and widely accepted model of processes with discrete states and dynamics (such as computer programs). Formally, a transition system is a triple $\mathcal{T} = (S, Act, \rightarrow)$ where S is a set of *states* (or *processes*), Act is a finite set of *actions*, and $\rightarrow \subseteq S \times Act \times S$ is a *transition relation*. We write $s \stackrel{a}{\rightarrow} t$ instead of $(s, a, t) \in \rightarrow$ and we extend this notation to elements of Act^* in the natural way. A state t is *reachable* from a state s, written $s \rightarrow^* t$, iff $s \stackrel{w}{\rightarrow} t$ for some $w \in Act^*$.

In the *equivalence-checking* approach to formal verification, one describes the *specification* (the intended behavior) and the actual *implementation* of a given process as states in transition systems, and then it is shown that they are *equivalent*. Here the notion of equivalence can be formalized in various ways according to specific needs of a given practical problem (see, e.g., [15] for an overview). It seems, however, that *simulation* and *bisimulation* equivalence are of special importance as their accompanying theory has been developed very intensively and found its way to many practical applications. Let $\mathcal{T} = (S, Act, \rightarrow)$ be a transition system. A binary relation $R \subseteq S \times S$ is a *simulation* iff whenever $(s, t) \in R$, then for each $s \stackrel{a}{\rightarrow} s'$ there is some $t \stackrel{a}{\rightarrow} t'$ such that $(s', t') \in R$. A process s is *simulated* by t, written $s \sqsubseteq t$, iff there is a simulation R such

⋆ Supported by the Grant Agency of the Czech Republic, grant No. 201/00/0400.

that $(s,t) \in R$. Processes s, t are *simulation equivalent*, written $s \simeq t$, iff they can simulate each other. A *bisimulation* is a symmetric simulation relation, and two processes s and t are *bisimilar* iff they are related by some bisimulation. In order to abstract from internal ('invisible') transitions of a given system, simulations and bisimulations are sometimes considered in their *weak* forms. Here, the silent steps are usually modeled by a distinguished action τ, and the *extended* transition relation $\Rightarrow \subseteq S \times Act \times S$ is defined by $s \stackrel{a}{\Rightarrow} t$ iff either $s = t$ and $a = \tau$, or $s \stackrel{\tau^i}{\rightarrow} s' \stackrel{a}{\rightarrow} t' \stackrel{\tau^j}{\rightarrow} t$ for some $i, j \in \mathbb{N}_0$ and $s', t' \in S$.

Simulations (and bisimulations) can also be viewed as *games* [12, 14] between two players, the attacker and the defender. In a simulation game the attacker wants to show that $s \not\sqsubseteq t$, while the defender attempts to frustrate this. Imagine that there are two tokens put on states s and t. Now the two players, attacker and defender, start to play a *simulation game* which consists of a (possibly infinite) number of *rounds* where each round is performed as follows: The attacker takes the token which was put on s originally and moves it along a transition labeled by (some) a; the task of the defender is to move the other token along a transition with the same label. If one player cannot move then the other player wins. The defender wins every infinite game. It can be easily shown that $s \sqsubseteq t$ iff the defender has a universal winning strategy. The only difference between a simulation game and a *bisimulation game* is that the attacker can *choose* his token at the beginning of every round (the defender has to respond by moving the other token). Again we get that $s \sim t$ iff the defender has a winning strategy. Corresponding 'weak forms' of the two games are defined in the obvious way. We use the introduced games at some points to give a more intuitive justification for our claims. Simulations and bisimulations can also be used to relate states of *different* transition systems; formally, two systems are considered to be a single one by taking the disjoint union.

In this paper we mainly consider processes of *pushdown automata*, which are interpreted as a (natural) model of sequential systems with mutually recursive procedures. A pushdown automaton is a tuple $\Delta = (Q, \Gamma, Act, \delta)$ where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, Act is a finite *input alphabet*, and $\delta : (Q \times \Gamma) \rightarrow \mathcal{P}(Act \times (Q \times \Gamma^*))$ is a *transition function* with finite image (here $\mathcal{P}(M)$ denotes the power set of M). We can assume (w.l.o.g.) that each transition increases the height (or length) of the stack by at most one (each PDA can be efficiently transformed to this kind of normal form). In the rest of this paper we adopt a more intuitive notation, writing $pA \stackrel{a}{\rightarrow} q\beta \in \delta$ instead of $(a,(q,\beta)) \in \delta(p,A)$. To Δ we associate the transition system $\mathcal{T}_\Delta$ where $Q \times \Gamma^*$ is the set of states (we write $p\alpha$ instead of (p,α)), Act is the set of actions, and the transition relation is determined by $pA\alpha \stackrel{a}{\rightarrow} q\beta\alpha$ iff $pA \stackrel{a}{\rightarrow} q\beta \in \delta$.

Let A, B be classes of processes. The problem whether a given process s of A is simulated (or weakly simulated) by a given process t of B is denoted by $A \sqsubseteq B$ (or $A \sqsubseteq_w B$, respectively). Similarly, the problem if s and t are simulation equivalent, weakly simulation equivalent, bisimilar, or weakly bisimilar, is denoted by $A \simeq B$, $A \simeq_w B$, $A \sim B$, or $A \approx B$, respectively. The classes of all pushdown processes and finite-state processes (i.e., processes of finite-state transition systems) are denoted **PDA** and **FS**, respectively. **BPA** (basic process algebra), also called context-free processes, is the subclass of **PDA** where $|Q| = 1$, i.e., without a finite-control.

The state of the art for simulation:
It has been known for some time that strong simulation preorder between **PDA** and **FS** is decidable in exponential time. This is because one can reduce the simulation problem to the model-checking problem with **PDA** and a fixed formula of the modal μ-calculus (see, e.g., [6, 4]). As model checking **PDA** with the modal μ-calculus is $EXPTIME$-complete [17] the result follows. A $PSPACE$ lower bound for the **FS** $\sqsubseteq$ **BPA** problem and a co-$\mathcal{NP}$ lower bound for the **BPA** $\sqsubseteq$ **FS** and **BPA** $\simeq$ **FS** problems have been shown in [6]. Furthermore, an $EXPTIME$ lower bound for the **FS** $\sqsubseteq$ **PDA** and **FS** $\simeq$ **PDA** problems have been shown in [4], but in these constructions the finite-state systems were not fixed. The problems of comparing two different BPA/PDA processes w.r.t. simulation preorder/equivalence are all undecidable.
Our contribution:
We show that the problems **BPA** $\sqsubseteq$ **FS**, **FS** $\sqsubseteq$ **BPA** and **BPA** $\simeq$ **FS** are $EXPTIME$-complete, but polynomial for every fixed finite-state system. On the other hand, the problems **PDA** $\sqsubseteq$ **FS**, **FS** $\sqsubseteq$ **PDA** and **PDA** $\simeq$ **FS** are $EXPTIME$-complete, even for a fixed finite-state system. Here, the main point are the lower bounds, which require some new insights into the power of the defender in simulation games. The matching upper bounds are obtained by a straightforward extension of the above mentioned reduction to the model-checking problem with the modal μ-calculus.
The state of the art for bisimulation:
It was known that strong and weak bisimulation equivalence between **PDA** and **FS** is decidable in exponential time, because one can construct (in polynomial time) characteristic modal μ-calculus formulae for the finite-state system and thus reduce the problem to model checking the **PDA** with a modal μ-calculus formula [11], which is decidable in exponential time [17]. The best known lower bound for the **PDA** $\approx$ **FS** problem was $PSPACE$-hardness, which even holds for a fixed finite state system [8]. The problem **PDA** $\sim$ **FS** is also $PSPACE$-hard, but polynomial in the size of the **PDA** for every fixed finite-state system [8]. Interestingly, the problem **BPA** $\approx$ **FS** (and **BPA** $\sim$ **FS**) is polynomial [7]. The symmetric problem of **PDA** $\sim$ **PDA** is decidable [9, 13], but the complexity is not known. So far, the best known lower bound for it was $PSPACE$-hardness [8]. The decidability of the **PDA** $\approx$ **PDA** problem is still open.
Our contribution:
We show that the problems **PDA** $\sim$ **FS** and **PDA** $\approx$ **FS** are $PSPACE$-complete by improving the known $EXPTIME$ upper bound to $PSPACE$. Furthermore, we show that the symmetric problem **PDA** $\sim$ **PDA** is $EXPTIME$-hard, by improving the known $PSPACE$ lower bound to $EXPTIME$. This new $EXPTIME$ lower bound even holds for the subclass of normed **PDA**.

Due to space constraints, several proofs are omitted. They can be found in the full version of the paper [5].

2 Lower Bounds

In this section we prove that all of the problems **BPA** $\sqsubseteq$ **FS**, **FS** $\sqsubseteq$ **BPA** and **BPA** $\simeq$ **FS** are $EXPTIME$-hard. The problems **PDA** $\sqsubseteq$ **FS**, **FS** $\sqsubseteq$ **PDA**, **PDA** $\simeq$ **FS** are $EXPTIME$-hard even for a *fixed* finite-state system. Moreover, we show $EXPTIME$-hardness of the **PDA** $\sim$ **PDA** problem.

An *alternating LBA* is a tuple $\mathcal{M} = (S, \Sigma, \gamma, s_0, \vdash, \dashv, \pi)$ where $S, \Sigma, \gamma, s_0, \vdash$, and $\dashv$ are defined as for ordinary non-deterministic LBA. In particular, S is a finite set of control states (we reserve 'Q' to denote a set of control states of pushdown automata), $\vdash, \dashv \in \Sigma$ are the left-end and right-end markers, respectively, 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. We assume (w.l.o.g.) that γ is defined so that

- 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 computational 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 *computational 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 computational 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 computational tree T such that all leafs of T are accepting configurations. The acceptance problem for alternating LBA is known to be $EXPTIME$-complete.

In subsequent proofs we often use $M_\star$ to denote the set $M \cup \{\star\}$ where M is a set and $\star \notin M$ is a fresh symbol.

Theorem 1. *The problem* ***BPA*** $\sqsubseteq$ ***FS*** *is* $EXPTIME$*-hard.*

Proof. Let $\mathcal{M} = (S, \Sigma, \gamma, s_0, \vdash, \dashv, \pi)$ be an alternating LBA and $w \in \Sigma^*$ an input word. We construct (in polynomial time) a BPA system $\Delta = (\Gamma, Act, \delta)$, a finite-state system $\mathcal{F} = (S, Act, \to)$, and processes α and X of Δ and $\mathcal{F}$, resp., such that $\mathcal{M}$ accepts w iff $\alpha \not\sqsubseteq X$. Let n be the length of w. We put $\Gamma = S_\star \times \Sigma \cup S \times \Sigma_\star \times \{0, \cdots, n+2\} \cup S \times \Sigma \times \{W\} \cup \{T, Z\}$. Configurations of $\mathcal{M}$ are encoded by strings over $S_\star \times \Sigma$ of length $n+2$. A configuration usv, where $u, v \in \Sigma^*$ and $s \in S$, is written as $\langle \star, v(k) \rangle \langle \star, v(k-1) \rangle \cdots \langle \star, v(2) \rangle \langle s, v(1) \rangle \langle \star, u(m) \rangle \cdots \langle \star, u(1) \rangle$ where k and m are the lengths of v and u, resp., and $v(i)$ denotes the i^{th} symbol of v (configurations are represented in a 'reversed order' since we want to write the top stack symbol on the left-hand side). Elements of $S \times \Sigma_\star \times \{0, \cdots, n+2\}$ are used as top stack symbols when pushing a new configuration to the stack (see below); they should

be seen as a finite memory where we keep (and update) the information about the position of the symbol which will be guessed by the next transition (as we count symbols from zero, the bounded counter reaches the value $n+2$ after guessing the last symbol), about the control state which is to be pushed, and about the (only) symbol of the form $\langle s, a\rangle$ which was actually pushed. The Z is a special 'bottom' symbol which can emit all actions and cannot be popped. The role of symbols of $S \times \Sigma \times \{W\} \cup \{T\}$ will be clarified later. The set of actions is $Act = \{a, c, f, s, d, t\} \cup (S_\star \times \Sigma)$, and δ consists of the following transitions:

1. $(\langle s, \star\rangle, i) \xrightarrow{a} (\langle s, \star\rangle, i+1)\,\langle \star, A\rangle$ — for all $A \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
2. $(\langle s, \star\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle s, A\rangle$ — for all $A \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
3. $(\langle s, A\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle \star, B\rangle$ — for all $A, B \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
4. $(\langle s, A\rangle, n+2) \xrightarrow{c} (\langle s, A\rangle, W)$ — for all $A \in \Sigma$, $s \in S$;
5. $(\langle s, A\rangle, W) \xrightarrow{d} \varepsilon$ — for all $s \in S$, $A \in \Sigma$ such that s is not rejecting;
6. $(\langle s, A\rangle, W) \xrightarrow{f} (\langle s', \star\rangle, 0)$ — for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) \in \{\forall, \exists\}$ and $s' = \mathit{first}(s, A)$;
7. $(\langle s, A\rangle, W) \xrightarrow{s} (\langle s', \star\rangle, 0)$ — for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) \in \{\forall, \exists\}$ and $s' = \mathit{second}(s, A)$;
8. $(\langle s, A\rangle, W) \xrightarrow{f} (\langle s', \star\rangle, 0)$ — for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) = \exists$ and $s' = \mathit{second}(s, A)$;
9. $(\langle s, A\rangle, W) \xrightarrow{s} (\langle s', \star\rangle, 0)$ — for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) = \exists$ and $s' = \mathit{first}(s, A)$;
10. $(\langle s, A\rangle, W) \xrightarrow{y} T$ — for all $s \in S$, $y \in \{f, s\}$ such that $\pi(s) = \mathit{acc}$;
11. $T \xrightarrow{t} T$
12. $Z \xrightarrow{y} Z$ — for all $y \in Act$;
13. $\langle x, A\rangle \xrightarrow{\langle x, A\rangle} \varepsilon$ — for all $x \in S_\star$, $A \in \Sigma$.

The process α corresponds to the initial configuration of $\mathcal{M}$, i.e.,

$$\alpha \quad = \quad (\langle s_0, \vdash\rangle, n{+}2)\,\langle \star, \dashv\rangle\,\langle \star, w(n)\rangle \cdots \langle \star, w(2)\rangle\,\langle \star, w(1)\rangle\,\langle s_0, \vdash\rangle\, Z$$

The behavior of α can be described as follows: whenever the top stack symbol is of the form $(\langle s, A\rangle, W)$, we know that the previously pushed configuration contains the symbol $\langle s, A\rangle$. If s is *rejecting*, no further transitions are possible. Otherwise, $(\langle s, A\rangle, W)$ can either disappear (emitting the action d—see rule 5), or it can perform one of the f and s actions as follows:

- If s is *universal* or *existential*, $(\langle s, A\rangle, W)$ can emit either f or s, storing $\mathit{first}(s, A)$ or $\mathit{second}(s, A)$ in the top stack symbol, respectively (rules 6, 7).
- If s is *existential*, $(\langle s, A\rangle, W)$ can also emit f and s while storing $\mathit{second}(s, A)$ and $\mathit{first}(s, A)$, respectively (rules 8, 9).
- If s is *accepting*, $(\langle s, A\rangle, W)$ emits f or s and pushes the symbol T which can do the action t forever (rules 10, 11).

If $(\langle s, A\rangle, W)$ disappears, the other symbols stored in the stack subsequently perform their symbol-specific actions and disappear (rule 13). If s is not accepting and $(\langle s, A\rangle, W)$ emits f or s, a new configuration is guessed and pushed to the stack; the construction of δ ensures that

- exactly $n + 2$ symbols are pushed (rules 1–4);
- at most one symbol of the form $\langle s', B\rangle$ is pushed; moreover, the s' must be the control state stored in the top stack symbol. After pushing $\langle s', B\rangle$, the B is also remembered in the top stack symbol (rule 2);
- if no symbol of the form $\langle s', B\rangle$ is pushed, no further transitions are possible after guessing the last symbol of the configuration (there are no transitions for symbols of the form $(\langle s', *\rangle, n + 2)$);
- after pushing the last symbol, the action c is emitted and a 'waiting' symbol $(\langle s', B\rangle, W)$ is pushed.

Now we define the finite-state system $\mathcal{F}$. The set of states of $\mathcal{F}$ is given by

$$S = \{X, F, S, U, C_0, \cdots, C_n\} \cup \{C_0, \cdots, C_n\} \times \{0, \cdots, n + 1\} \times (S_\star \times \Sigma)^4_\star.$$

Transitions of $\mathcal{F}$ are

1. $X \xrightarrow{a} X,\ X \xrightarrow{c} F,\ X \xrightarrow{c} S,\ X \xrightarrow{c} C_i$ for every $0 \le i \le n$;
2. $F \xrightarrow{f} X,\ F \xrightarrow{y} U$ for every $y \in Act - \{f\}$;
3. $S \xrightarrow{s} X,\ S \xrightarrow{y} U$ for every $y \in Act - \{s\}$;
4. $C_i \xrightarrow{d} (C_i, 0, \star, \star, \star, \star),\ C_i \xrightarrow{y} U$ for every $0 \le i \le n$, $y \in Act - \{d\}$;
5. $U \xrightarrow{y} U$ for every $y \in Act$;
6. $(C_i, j, \star, \star, \star, \star) \xrightarrow{y} (C_i, j{+}1, \star, \star, \star, \star)$ for all $0 \le i \le n$, $0 \le j < i$, and $y \in S_\star \times \Sigma$;
7. $(C_i, i, \star, \star, \star, \star) \xrightarrow{y} (C_i, i{+}1, y, \star, \star, \star)$ for all $0 \le i \le n$ and $y \in S_\star \times \Sigma$;
8. $(C_i, i{+}1, y, \star, \star, \star) \xrightarrow{z} (C_i, (i{+}2) \bmod (n{+}2), y, z, \star, \star)$ for all $0 \le i \le n$ and $y, z \in S_\star \times \Sigma$;
9. $(C_i, j, y, z, \star, \star) \xrightarrow{u} (C_i, (j{+}1) \bmod (n{+}2), y, z, \star, \star)$ for all $0 \le i \le n$, $i{+}2 \le j \le n{+}1$, and $y, z \in S_\star \times \Sigma$;
10. $(C_i, j, y, z, \star, \star) \xrightarrow{u} (C_i, j{+}1, y, z, \star, \star)$ for all $0 \le i \le n$, $0 \le j < i$, and $y, z, u \in S_\star \times \Sigma$;
11. $(C_i, i, y, z, \star, \star) \xrightarrow{u} (C_i, i{+}1, y, z, u, \star)$ for all $0 \le i \le n$ and $y, z, u \in S_\star \times \Sigma$;
12. $(C_i, i{+}1, y, z, u, \star) \xrightarrow{v} (C_i, (i{+}2) \bmod (n{+}2), y, z, u, v)$ for all $0 \le i \le n$ and $y, z, u, v \in S_\star \times \Sigma$;
13. $(C_i, (i{+}2) \bmod (n{+}2), y, z, u, v) \xrightarrow{x} U$ for all $0 \le i \le n$, $x \in Act$, and $y, z, u, v \in S_\star \times \Sigma$ such that (y, z) and (u, v) are *not* compatible pairs (see below).

A fragment of $\mathcal{F}$ is shown in Fig. 1. The role of states of the form $(C_i, 0, \star, \star, \star, \star)$ and their successors (which are not drawn in Fig. 1) is clarified below.

Now we prove that $\mathcal{M}$ accepts w iff $\alpha \not\sqsubseteq X$. Intuitively, the simulation game between α and X corresponds to constructing a branch in a computational tree for $\mathcal{M}$

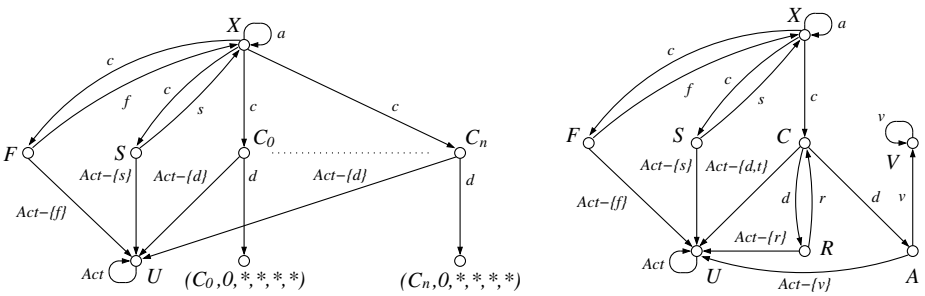

Fig. 1. The systems $\mathcal{F}$ and $\mathcal{F}'$ (successors of $(C_i, 0, \star, \star, \star, \star)$ in $\mathcal{F}$ are omitted).

on w. The attacker (who plays with α) wants to show that there is an accepting computational tree, while the defender aims to demonstrate the converse. The attacker is therefore 'responsible' for choosing the appropriate successors of all existential configurations (selecting those for which an accepting subtree exists), while the defender chooses successors of universal configurations (selecting those for which no accepting subtree exists). The attacker wins iff the constructed branch reaches an accepting configuration. The choice is implemented as follows: after pushing the last symbol of a configuration, the attacker has to emit the c action and push a 'waiting' symbol (see above). The defender can reply by entering the state F, S, or one of the C_i states. Intuitively, he chooses among the possibilities of selecting the first or the second successor, or checking that the i^{th} symbol of the lastly pushed configuration was guessed correctly (w.r.t. the previous configuration). Technically, the choice is done by forcing the attacker to emit a specific action in the *next* round—observe that if the defender performs, e.g., the $X \xrightarrow{c} F$, transition, then the attacker *must* use one of his f transitions in the next round, because otherwise the defender would go immediately to the state U where he can simulate 'everything', i.e., the attacker loses the game. As the defender is responsible only for selecting the successors of *universal* configurations, the attacker has to follow his 'dictate' only if the lastly pushed configuration was universal; if it was existential, he can choose the successor according to *his* own will (see the rules 6–9 in the definition of δ). If the lastly pushed configuration was rejecting, the attacker cannot perform any further transitions from the waiting symbol, which means that the defender wins. If the configuration was accepting and the defender enters F of S via the action c, then the attacker wins; first he replaces the waiting symbol with T, emitting f or s, resp. (so that the defender has to go back to X) and then he does the action t. The purpose of the states C_i (and their successors) is to ensure that the attacker cannot gain anything by 'cheating', i.e., by guessing configurations incorrectly. If the defender is suspicious that the attacker has cheated when pushing the last configuration, he can 'punish' the attacker by going (via the action c) to one of the C_i states. Doing so, he forces the attacker to *remove* the waiting symbol in the next round (see the rule 5 in the definition of δ). Now the atacker can only pop symbols from the stack and emit the symbol-specific actions. The defender 'counts' those actions and 'remembers' the symbols at positions i and $i + 1$ in the lastly and the previously pushed configurations. After the defender collects the four symbols, he either enters a universal state U (i.e., he wins the game), or gets 'stuck' (which means that the attacker wins). It depends on whether the two pairs

of symbols are compatible w.r.t. the transition function γ of $\mathcal{M}$ or not (here we use a folklore technique of checking the consistency of successive configurations of Turing machines). Observe that if the lastly pushed configuration was accepting, the defender still has a chance to perform a consistency check (in fact, it is his 'last chance' to win the game). On the other hand, if the defender decides to check the consistency right at the beginning of the game (when the attacker plays the c transition from α), he inevitably loses because the attacker reaches the bottom symbol Z in $n+2$ transitions and then he can emit the action t. It follows that the attacker has a winning strategy iff $\mathcal{M}$ accepts w. □

Theorem 2. *The problem **PDA** $\sqsubseteq$ **FS** is $EXPTIME$-hard even for a fixed finite-state process.*

Proof. We modify the construction of Theorem 1. Intuitively, we just re-implement the cheating detection so that the compatibility of selected pairs of symbols is checked by the pushdown system and not by $\mathcal{F}$ (now we can store the four symbols in the finite control). However, it must still be the defender who selects the (position of the) pair. This can be achieved with a fixed number of states (see [5]).

Theorem 3. *The problem **FS** $\sqsubseteq$ **BPA** is $EXPTIME$-hard.*

Proof. The technique is similar to the one of Theorem 1 (see [5]).

Theorem 4. *The problem **FS** $\sqsubseteq$ **PDA** is $EXPTIME$-hard even for a fixed finite-state process.*

An immediate consequence of Theorem 1 and Theorem 2 is the following:

Corollary 5. *The problem **BPA** $\simeq$ **FS** is $EXPTIME$-hard. Moreover, the problem **PDA** $\simeq$ **FS** is $EXPTIME$-hard even for a fixed finite-state process.*

Proof. There is a simple (general) reduction from the $A \sqsubseteq B$ problem to the $A \simeq B$ problem (where A, B are classes of processes) which applies also in this case—given processes $p \in A$ and $q \in B$, we construct processes p', q' such that p' has only the transitions $p' \xrightarrow{a} p, p' \xrightarrow{a} q$, and q' has only the transition $q' \xrightarrow{a} q$. It follows immediately that $p' \simeq q'$ iff $p \sqsubseteq q$. □

The problem of **PDA** $\sim$ **PDA** is decidable, but the exact complexity is not known. The decision procedures described in [9, 13] do not give any upper complexity bound. So far, the best known lower bound for this problem was $PSPACE$-hardness [8]. However, while the problem **PDA** $\sim$ **FS** is $PSPACE$-complete (see Section 3) the problem **PDA** $\sim$ **PDA** is at least $EXPTIME$-hard. This $EXPTIME$ lower bound even holds for the subclass of normed **PDA** (a **PDA** is *normed* iff from every reachable configuration it is possible to empty the stack).

The proof of the following theorem uses a technique which can be traced back to Jančar [1]; a more explicit formulation is due to Srba [10] who used the technique in the different context of Basic Parallel Processes. The main idea is that in a bisimulation game the defender can force the attacker to do certain things according to the defender's choices. The full proof can be found in [5].

Theorem 6. *The problem* ***PDA*** $\sim$ ***PDA*** *is* $EXPTIME$*-hard, even for normed* ***PDA****.*

Proof. (sketch) The proof is done by a polynomial-time reduction of the ($EXPTIME$-complete) acceptance problem of alternating LBA to the **PDA** $\sim$ **PDA** problem. The bisimulation game proceeds as follows. The attacker guesses LBA configurations and pushes them onto the stack. The defender is forced to copy these moves. At existential control states (of the LBA) the attacker chooses the successor control state, and at the universal control states (of the LBA) the defender gets to choose the successor control state (this requires the technique mentioned above where the defender forces the attacker to do certain things). At any time, the defender can force the attacker to enter a so-called check-phase. In this check-phase it is verified if the LBA configuration at the top of the stack is really a successor configuration (according to the transition rules of the LBA) of the LBA configuration that was pushed onto the stack before. If not, then the defender wins the bisimulation game. This construction forces the attacker to play 'honestly', i.e., to correctly simulate the behavior of the LBA. If an accepting configuration (of the LBA) is reached in this way then the attacker wins the bisimulation game (having proved, despite the interference of the defender's choices at the universal control states, that the alternating LBA accepts). Otherwise, the bisimulation game goes on forever and the defender wins. This construction ensures that the attacker has a winning strategy if and only if the alternating LBA accepts. Thus, the alternating LBA accepts iff the two PDAs are not bisimilar. □

3 Upper Bounds

The next theorem extends the result for strong simulation which appeared in [6]; the proof is based on the same idea, but the constructed formula φ is now completely fixed.

Theorem 7. *The problems* ***PDA*** $\sqsubseteq_w$ ***FS****,* ***FS*** $\sqsubseteq_w$ ***PDA****, and* ***PDA*** $\simeq_w$ ***FS*** *are in* $EXPTIME$*.*

Proof. All of the above mentioned problems are polynomially reducible to the model-checking problem with pushdown automata and a fixed formula φ of the modal μ-calculus (which is decidable in deterministic exponential time [17]).

Let $\varphi \equiv \nu X.\Box_a \Diamond_b \langle c \rangle X$, where $\Box_a \psi \equiv \nu Y.(\psi \wedge [a]Y)$ and $\Diamond_b \psi \equiv \nu Z.(\psi \vee \langle b \rangle Z)$. Intuitively, $\Box_a \psi$ says that each state which is reachable from a given process via a finite sequence of a-transitions satisfies ψ, and $\Diamond_b \psi$ says that a given process can reach a state satisfying ψ via a finite sequence of b-transitions. Hence, the meaning of φ can be explained as follows: a process satisfies φ iff after each finite sequence of a-transitions it can perform a finite sequence of b-transitions ended with one c-transition so that the state which is entered again satisfies φ (we refer to [3] for a precise definition of the syntax and semantics of the modal μ-calculus). Now let $\Delta = (Q, \Gamma, Act, \delta)$ be a pushdown system, $\mathcal{F} = (F, Act, \rightarrow)$ a finite-state system, $p\alpha$ a process of Δ, and f a process of $\mathcal{F}$. We construct a pushdown system $\Delta = (Q \times F \times Act \times \{0,1\}, \Gamma \cup \{Z\}, \{a,b,c\}, \delta')$ (where $Z \notin \Gamma$ is a new bottom symbol) which 'alternates' the $\stackrel{x}{\Rightarrow}$ transitions of Δ and $\mathcal{F}$, remembering the 'x' in its finite control. Formally, δ' is constructed as follows:

- for all $qA \stackrel{x}{\rightarrow} r\beta \in \delta$ and $g \in F$ we add $(q,g,\tau,0)A \stackrel{a}{\rightarrow} (r,g,x,0)\beta$ to δ';

- for all $qA \stackrel{\tau}{\rightarrow} r\beta \in \delta$, $x \in Act$, and $g \in F$ we add $(q,g,x,0)A \stackrel{a}{\rightarrow} (r,g,x,0)\beta$ to δ';
- for all $q \in Q$, $g \in F$, $x \in Act$, and $Y \in \Gamma \cup \{Z\}$ we add $(q,g,x,0)Y \stackrel{b}{\rightarrow} (q,g,x,1)Y$ to δ';
- for each transition $g \stackrel{x}{\rightarrow} g'$ of $\mathcal{F}$ and all $q \in Q, Y \in \Gamma \cup \{Z\}$ we add $(q,g,x,1)Y \stackrel{b}{\rightarrow} (q,g',\tau,1)Y$ to δ';
- for all $g \stackrel{\tau}{\rightarrow} g'$ of $\mathcal{F}$, $x \in Act$, $q \in Q$, and $Y \in \Gamma \cup \{Z\}$ we add $(q,g,x,1)Y \stackrel{b}{\rightarrow} (q,g',x,1)Y$ to δ';
- for all $q \in Q$, $g \in F$, and $Y \in \Gamma \cup \{Z\}$ we add $(q,g,\tau,1)Y \stackrel{c}{\rightarrow} (q,g,\tau,0)Y$ to δ';

We claim that $p\alpha \sqsubseteq_w f$ iff $(p,f,\tau,0)\alpha Z \models \varphi$. Indeed, each sequence of a-transitions of $(p,f,\tau,0)\alpha Z$ corresponds to some $\stackrel{x}{\Rightarrow}$ move of $p\alpha$ and vice versa; and after each such sequence, the 'token' can be switched from 0 to 1 (performing b), and now each sequence of b's ended with one c corresponds to a $\stackrel{x}{\Rightarrow}$ move of f. Then, the token is switched back to 0 and the computation proceeds in the same way. φ says that this can be repeated forever, unless we reach a state which cannot do any a when the token is set to 0. The new bottom symbol Z has been added to ensure that $(p,f,\tau,0)\alpha Z$ cannot get stuck just due to the emptiness of the stack. The **FS** $\sqsubseteq_w$ **PDA** direction is handled in a very similar way (the roles of $p\alpha$ and f are just interchanged). □

Corollary 8. *The problems **BPA** $\sqsubseteq_w$ **FS**, **FS** $\sqsubseteq_w$ **BPA**, and **BPA** $\simeq_w$ **FS** are decidable in polynomial time for (any)* fixed *finite-state process.*

Proof. The complexity result of [17] says that model-checking with any fixed formula of the modal μ-calculus and pushdown processes with a *fixed* number of control states is decidable in polynomial time. By synchronizing a given BPA process with a given (fixed) finite-state process as in Theorem 7 we obtain a pushdown system with a fixed number of control states, and the result follows. □

Now we show that the problem **PDA** $\approx$ **FS** is in $PSPACE$. First, we recall some results from [2]. A characteristic formula of a finite-state system F w.r.t. $\approx$ is a formula Θ_F s.t. for every general system G which uses the same set of actions as F we have that $G \models \Theta_F \iff G \approx F$. It has been shown in [2] that characteristic formulae for finite-state systems w.r.t. $\approx$ can be effectively constructed in the temporal logic EF (a simple fragment of CTL), by using the following theorem (here, $\approx_k$ denotes 'weak bisimilarity up-to k', which means that the defender has a strategy to defend for at least k rounds in the weak bisimulation game).

Theorem 9. *(taken from [2])*
Let F be a finite-state system with n states and G a general system. States $g \in G$ and $f \in F$ are weakly bisimilar iff the following conditions hold: (1) $g \approx_n f$ and (2) For each state g' which is reachable from g there is a state $f' \in F$ such that $g' \approx_n f'$.

One constructs characteristic formulae $\Phi_{k,f}$ for states f in F w.r.t. $\approx_k$ that satisfy $g \models \Phi_{k,f} \iff g \approx_k f$. The family of $\Phi_{k,f}$ formulae is defined inductively on k as follows:

$$\Phi_{0,f} := true$$

$$\Phi_{k+1,f} := \left(\bigwedge_{a \in Act} \bigwedge_{f' \in S(f,a)} \Diamond_a \Phi_{k,f'} \right) \wedge \left(\bigwedge_{a \in Act} (\neg \Diamond_a (\bigwedge_{f' \in S(f,a)} \neg \Phi_{k,f'})) \right)$$

where $S(f,a) = \{f' \mid f \stackrel{a}{\rightarrow} f'\}$ and $\Diamond_\tau$ means "reachable via a finite number of τ-transitions" and $\Diamond_a := \Diamond_\tau \langle a \rangle \Diamond_\tau$ for $a \neq \tau$.

Empty conjunctions are equivalent to *true*. Thus, by Theorem 9, the characteristic formula Θ_f for a process f of a finite-state system $\mathcal{F} = (F, Act, \rightarrow)$ with n states is

$$\Theta_f \equiv \Phi_{n,f} \wedge \neg \Diamond \left(\bigwedge_{f' \in F} \neg \Phi_{n,f'} \right)$$

So one can reduce the problem **PDA** $\approx$ **FS** to a model checking problem for pushdown automata and (a slight extension of) the logic EF. The following proof-sketch uses many results by Walukiewicz [16]. For a complete proof, it would be necessary to repeat many of these, so we just sketch the main ideas and the crucial modification of the algorithm from [16]. It has been shown by Walukiewicz in [16] that model checking pushdown automata with the logic EF is $PSPACE$-complete. But our result does not follow directly from that. First, our characteristic formulae use a slight extension of EF, because of the $\Diamond_\tau$ operator (normal EF has only the $\Diamond$ operator). However, the model checking algorithm of [16] can trivially be generalized to this extension of EF, without increasing its complexity. The second, and more important problem is that the size of the characteristic formula Θ_F is exponential in n (where n is the number of states of F). However, a closer analysis of the model checking algorithm presented in [16] reveals that its complexity does not depend directly on the size of the formula, but rather on the number of its distinct subformulae. More precisely, this algorithm uses a so-called assumption function that assigns sets of subformulae to every control-state of the PDA. Of course, each EF formula has only a polynomial number of subformulae and hence the assumption function can be represented in polynomial space. However, it is also true for our characteristic formula Θ_F — although its size is exponential in n, the number of its distinct subformulae $\Phi_{k,f}$ is bounded by $\mathcal{O}(n^2)$, because $0 \leq k \leq n$ and F has only n states. Hence, we can run the mentioned model-checking algorithm for EF. Instead of 'unwinding' the $\Phi_{k,f}$ subformulae, we keep the abbreviations $\Phi_{k,f}$ as long as possible and expand them only (on-the-fly) when necessary (using the inductive definitions above). Thus, the whole algorithm works in polynomial space and we obtain the following theorem.

Theorem 10. *The problem* ***PDA*** $\approx$ ***FS*** *is in* $PSPACE$.

4 Conclusions

The following table summarizes the complexity of all problems of comparing PDA and BPA to finite-state systems w.r.t. strong and weak simulation preorder/equivalence and strong and weak bisimilarity. **FS** means a finite-state system that is part of the input of the problem, while $\mathcal{F}$ means "any fixed finite-state system" for the upper complexity bounds and "some fixed finite-state system" for the lower complexity bounds.

	$\sqsubseteq_w$ **FS** $\sqsubseteq$ **FS**	$\sqsubseteq_w \mathcal{F}$ $\sqsubseteq \mathcal{F}$	**FS** $\sqsubseteq_w$ **FS** $\sqsubseteq$	$\mathcal{F} \sqsubseteq_w$ $\mathcal{F} \sqsubseteq$	$\simeq_w$ **FS** $\simeq$ **FS**	$\simeq_w \mathcal{F}$ $\simeq \mathcal{F}$	$\approx$ **FS** $\sim$ **FS**	$\sim \mathcal{F}$	$\approx \mathcal{F}$
BPA	EXPTIME complete	in P	EXPTIME complete	in P	EXPTIME complete	in P	in P	in P	in P
PDA	EXPTIME complete	EXPTIME complete	EXPTIME complete	EXPTIME complete	EXPTIME complete	EXPTIME complete	PSPACE complete	in P	PSPACE complete

Finally, we have also shown (in Theorem 6) that the problem **PDA** $\sim$ **PDA** of checking bisimilarity of two pushdown systems is $EXPTIME$-hard. Thus, it is harder than the problem **PDA** $\sim$ **FS** of checking bisimilarity of a pushdown system and a finite-state system, which is only $PSPACE$-complete.

References

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

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

[3] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[4] A. Kučera. On simulation-checking with sequential systems. In *Proceedings of ASIAN 2000*, volume 1961 of *LNCS*, pages 133–148. Springer, 2000.

[5] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. Technical report FIMU-RS-2002-01, Faculty of Informatics, Masaryk University, 2002.

[6] A. Kučera and R. Mayr. Simulation preorder over simple process algebras. *Information and Computation*, 173(2):184–198, 2002.

[7] 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(1–2):677–700, 2002.

[8] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proceedings of IFIP TCS'2000*, volume 1872 of *LNCS*, pages 474–488. Springer, 2000.

[9] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite outdegree. In *Proceedings of 39th Annual Symposium on Foundations of Computer Science*, pages 120–129. IEEE Computer Society Press, 1998.

[10] 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.

[11] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

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

[13] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255:1–31, 2001.

[14] W. Thomas. On the ehrenfeucht-fraïssé game in theoretical computer science. In *Proceedings of TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer, 1993.

[15] R. van Glabbeek. The linear time—branching time spectrum. *Handbook of Process Algebra*, pages 3–99, 1999.

[16] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proceedings of FST&TCS 2000*, volume 1974 of *LNCS*, pages 127–138. Springer, 2000.

[17] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

Appendix

Theorem 2. *The problem* ***PDA*** $\sqsubseteq$ ***FS*** *is* $EXPTIME$*-hard even for a fixed finite-state process.*

Proof. We modify the construction of Theorem 1. Intuitively, we just re-implement the cheating detection so that the compatibility of selected pairs of symbols is checked by the pushdown system and not by $\mathcal{F}$ (now we can store the four symbols in the finite control). However, it must still be the defender who selects the (position of the) pair; we show how to achieve this with a fixed number of states.

First, we define $Act = \{a, c, f, s, d, t, v, r\}$ and instead of $\mathcal{F}$ we take the system $\mathcal{F}'$ of Fig. 1 (which is fixed). Now we construct a pushdown system $(Q, \Gamma, Act, \delta')$, where Γ is the same as in Theorem 1, the set of control states is

$$Q \quad = \quad \{g, p_0, \cdots, p_{n+1}\} \quad \cup \quad \{c_0, \cdots, c_n\} \times \{0, \cdots, n+1\} \times (S_\star \times \Sigma)^4_\star$$

and δ' is constructed as follows:

1. for each transition $X \xrightarrow{y} \alpha$ of δ which has *not* been defined by the rule 5. or 13. (see the proof of Theorem 1) we add to δ' the transition $gX \xrightarrow{y} g\alpha$;
2. for each 'waiting' symbol X of Γ (i.e., a symbol of the form $(\langle s, A\rangle, W)$) we add to δ' the transition $gX \xrightarrow{d} p_0\varepsilon$;
3. for all $0 \leq i \leq n$ and $X \in \Gamma$ we add to δ' the transitions $p_iX \xrightarrow{d} p_iX$, $p_iX \xrightarrow{r} p_{i+1}X$, and $p_iX \xrightarrow{v} (c_i, 0, \star, \star, \star, \star)X$;
4. for all $X \in \Gamma$ we add to δ' the transitions $p_{n+1}X \xrightarrow{t} p_{n+1}X$;
5. finally, we add to δ' the transitions which perform consistency checks; they are (informally) described below.

The initial configuration of Δ is the α of Theorem 1 augmented with the control state g.

The proof follows the line of Theorem 1. The only difference is how the defender checks the consistency of the lastly and the previously pushed configurations. If he wants to perform such a check, he replies by $X \xrightarrow{c} C$ when the attacker enters a 'waiting' state via his c-transition. It means that the attacker is forced to pop the waiting symbol and change the control state to p_0 via a d-transition in the next round (rule 2). Intuitively, the attacker 'offers' the defender a possibility to check the pair of symbols at positions 0 and 1. Now we distinguish two cases:

- If the defender wants to accept the proposal, he replies by $C \xrightarrow{d} A$; it means that the attacker must emit the action v in the next round and change the control state to $(c_0, 0, \star, \star, \star, \star)$. From now on the attacker will only pop symbols from the stack, emitting the action v, until he finds the four symbols or reaches the bottom of stack. If the collected pairs of symbols are compatible (or if the bottom of stack is reached), the attacker emits t and wins; otherwise, he becomes 'stuck' and the defender wins.

– If the defender does not want to accept the proposal (i.e., if he wants to check pairs at another position), he replies by $C \xrightarrow{d} R$, forcing the attacker to use his (only) r-transition in the next round (the control state is changed from p_0 to p_1). The defender replies by $R \xrightarrow{r} C$. Now the attacker must use his $p_1 X \xrightarrow{d} p_1 X$ transition, which is in fact an offer to check symbols at positions 1 and 2. Now the game continues in the same fashion.

If the defender does not accept any 'offer' from the attacker (i.e., if the attacker reaches the control state p_{n+1}), the attacker wins by emitting the action t (rule 4). Now we can readily confirm that the attacker has a winning strategy iff $\mathcal{M}$ accepts w. □

Theorem 3. *The problem* ***FS*** $\sqsubseteq$ ***BPA*** *is* $EXPTIME$*-hard.*

Proof. The technique is similar to the one of Theorem 1 Given an alternating LBA $\mathcal{M} = (S, \Sigma, \gamma, s_0, \vdash, \dashv, \pi)$ and $w \in \Sigma^*$, we construct (in polynomial time) a finite-state system $\bar{\mathcal{F}} = (S, Act, \rightarrow)$, a BPA system $\bar{\Delta} = (\Gamma, Act, \delta)$, and processes X and α of $\bar{\mathcal{F}}$ and $\bar{\Delta}$, resp., such that $\mathcal{M}$ accepts w iff $X \not\sqsubseteq \alpha$. The set of states of $\bar{\mathcal{F}}$ is

$$\{X, F, S\} \quad \cup \quad \{C_0, \cdots, C_n\} \times \{0, \cdots, n+1\} \times (S_\star \times \Sigma)^4_\star,$$

and the set of actions Act is $\{a, c, f, s, t\} \cup (S_\star \times \Sigma)$. Transitions of $\bar{\mathcal{F}}$ look as follows:

1. $X \xrightarrow{a} X,\ X \xrightarrow{f} F, X \xrightarrow{s} S, X \xrightarrow{c} (C_i, 0, \star, \star, \star, \star)$ for all $0 \leq i \leq n$;
2. $F \xrightarrow{a} X, S \xrightarrow{a} X$;
3. we add all transitions given by the rules 6.–12. in the definition of the system $\mathcal{F}$ in (the proof of) Theorem 1;
4. $(C_i, (i{+}2) \bmod (n{+}2), y, z, u, v) \xrightarrow{t} X$ for all $0 \leq i \leq n$ and $y, z, u, v \in S_\star \times \Sigma$ such that (y, z) and (u, v) are not compatible pairs.

The stack alphabet Γ of $\bar{\Delta}$ is $(S_\star \times \Sigma) \cup (S \times \Sigma_\star \times \{0, \cdots, n{+}2\}) \cup \{U, Z\}$. Here U is the 'universal' symbol (which can simulate everything), and Z is a bottom symbol. δ consists of the following transitions:

1. $(\langle s, \star\rangle, i) \xrightarrow{a} (\langle s, \star\rangle, i+1)\,\langle \star, A\rangle$ for all $A \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
2. $(\langle s, \star\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle s, A\rangle$ for all $A \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
3. $(\langle s, A\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle \star, B\rangle$ for all $A, B \in \Sigma$, $s \in S$, $0 \leq i \leq n+1$;
4. $(\langle s, \star\rangle, i) \xrightarrow{x} U$ for all $s \in S$, $0 \leq i \leq n{+}1$, and $x \in \{f, s, c\}$;
5. $(\langle s, A\rangle, i) \xrightarrow{x} U$ for all $s \in S$, $0 \leq i \leq n{+}1$, and $x \in \{f, s, c\}$;
6. $(\langle s, A\rangle, n+2) \xrightarrow{f} (\langle s', \star\rangle, 0)$ for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) \in \{\forall, \exists\}$ and $s' = \mathit{first}(s, A)$;
7. $(\langle s, A\rangle, n+2) \xrightarrow{s} (\langle s', \star\rangle, 0)$ for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) \in \{\forall, \exists\}$ and $s' = \mathit{second}(s, A)$;
8. $(\langle s, A\rangle, n+2) \xrightarrow{f} (\langle s', \star\rangle, 0)$ for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) = \forall$ and $s' = \mathit{second}(s, A)$;

9.	$(\langle s, A\rangle, n+2) \xrightarrow{s} (\langle s', \star\rangle, 0)$	for all $s, s' \in S$, $A \in \Sigma$ such that $\pi(s) = \forall$ and $s' = \mathit{first}(s, A)$;
10.	$(\langle s, A\rangle, n+2) \xrightarrow{c} \varepsilon$	for all $s \in S$ and $A \in \Sigma$;
11.	$(\langle s, A\rangle, n+2) \xrightarrow{a} U$	for all $s \in S$ and $A \in \Sigma$;
12.	$(\langle s, A\rangle, n+2) \xrightarrow{x} U$	for all $s \in S$, $A \in \Sigma$, and $x \in \{f, s\}$ such that $\pi(s) = \mathit{rej}$;
13.	$\langle x, A\rangle \overset{\langle x,A\rangle}{\rightarrow} \varepsilon$	for all $x \in S_\star$ and $A \in \Sigma$;
14.	$\langle x, A\rangle \overset{\langle y,B\rangle}{\rightarrow} U$	for all $x, y \in S_\star$ and $A, B \in \Sigma$ such that $x \neq y$ or $A \neq B$;
15.	$Z \xrightarrow{x} U$	for all $X \in \mathit{Act} - \{t\}$;
16.	$U \xrightarrow{x} U$	for all $X \in \mathit{Act}$;

The process α corresponds to the initial configuration of $\mathcal{M}$, i.e.,

$$\alpha \quad = \quad (\langle s_0, w(1)\rangle, n{+}2)\, \langle \star, \dashv\rangle\, \langle \star, w(n)\rangle \cdots \langle \star, w(2)\rangle\, \langle \star, w(1)\rangle\, \langle s_0, \vdash\rangle\, Z$$

Again, the simulation game corresponds to constructing a branch in a computational tree for $\mathcal{M}$ on w. The attacker (who plays with X now) wants to show that there is an accepting computational tree, while the defender wants to prove the converse. It means that the attacker chooses successors of existential configurations, and the defender chooses successors of universal configurations. At the beginning, the attacker has to use one of his f, s, or c transitions (if he uses $X \xrightarrow{a} X$, the defender wins by pushing U; see the rule 11). It corresponds to choosing the first or the second successor, or forcing a consistency check. As the defender is responsible for choosing the successors of universal configurations, he can 'ignore' the attacker's choice if the lastly pushed configuration was universal (rules 6.–9.). If the lastly pushed configuration was accepting, the defender gets 'stuck' and loses. If it was rejecting, the attacker's only chance is to use one of his c-transitions and perform a consistency check (if the attacker emits any other action, the defender wins by pushing U—see the rules 11, 12). The consistency check is implemented as follows: first, the attacker chooses the (index of the) pair to be verified by one of his $X \xrightarrow{c} (C_i, 0, \star, \star, \star, \star)$ transitions (observe that if this transition is used 'too early', i.e., before the whole configuration is pushed, the defender wins by pushing U—see the rules 4, 5). Now the attacker has to guess the symbols which are stored in the stack, remembering the four crucial symbols. If he makes an incorrect guess, the defender pushes U and wins (rule 14). Otherwise, the defender has to pop symbols from the stack (rule 13). If the collected pairs of symbols are compatible, the attacker gets stuck (and the defender wins). Otherwise, the attacker wins by emitting t. The bottom symbol Z ensures that the attacker loses if he decides to make a consistency check right at beginning of the game, because then the defender reaches U before the attacker can emit t. Also observe that we cannot use U as the bottom symbol, because then the attacker would not be able to check the consistency of symbols at positions n and $n+1$ in the first two configurations (the attacker's t-transition would be matched by U). We see that the attacker has a winning strategy iff $\mathcal{M}$ accepts w. □

Theorem 4. *The problem* ***FS*** $\sqsubseteq$ ***PDA*** *is* $EXPTIME$*-hard even for a fixed finite-state process.*

Proof. The required modification of the proof of Theorem 3 is quite straightforward. Instead of $\bar{\mathcal{F}}$ we take the system $\bar{\mathcal{F}}'$ of Fig. 2. The consistency check is performed by the pushdown system, but the attacker still selects the index of the pair he wants to verify by performing the corresponding number of c-transitions. The only thing the defender can do is to 'count' those c's in the finite control of the pushdown system (if the attacker uses more than $n+1$ c-transitions, the defender can push U and thus he wins). When the attacker emits the first s, the defender has to start the consistency check of the previously selected pairs—he successively pops symbols from the stack (emitting s) until he collects the four symbols. If they are compatible, the defender can go on and perform an infinite number of s-transitions (and hence he wins); otherwise, the defender gets stuck and the attacker wins. □

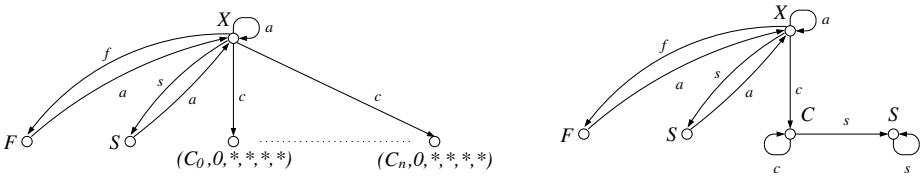

Fig. 2. The systems $\bar{\mathcal{F}}$ and $\bar{\mathcal{F}}'$ (successors of $(C_i, 0, \star, \star, \star, \star)$ in $\bar{\mathcal{F}}$ are omitted).

Theorem 6. *The problem* ***PDA*** $\sim$ ***PDA*** *is* $EXPTIME$*-hard, even for normed* ***PDA****.*

Proof. Let $\mathcal{M} = (S, \Sigma, \gamma, s_0, \vdash, \dashv, \pi)$ be an alternating LBA and $w \in \Sigma^*$ an input word. Let n be the length of w. We construct (in polynomial time) a PDA (Q, Γ, Act, δ) and processes $\alpha := (q_0, 0)q_0w$ and $\alpha' := (q_0', 0)q_0w$ such that $\mathcal{M}$ accepts w iff $\alpha \not\sim \alpha'$.

We represent an LBA configuration as uqv where u is the tape to the left of the head, q is the control-state, and v is the tape under the head and to the right of it. Let $S' := \{q' \mid q \in S\}$ and $S'' := \{q'' \mid q \in S\}$. $Q := S \times \{0, \ldots, n-1\} \cup (S \times \Sigma) \times \{1, \ldots, n\} \cup S' \times \{0, \ldots, n-1\} \cup (S' \times \Sigma) \times \{1, \ldots, n\} \cup S'' \times \{0, \ldots, n-1\} \cup (S'' \times \Sigma) \times \{1, \ldots, n\} \cup \{(\tilde{q}, 0) \mid q \in S\} \cup (S \times S) \times \{0\} \cup \{q_c, q_c''\} \cup \{x\}$. For every state $q \in Q$, the states q', q'' and $\tilde{q}$ are seen as being associated to q. $\Gamma := \Sigma \cup S$, $Act := \Sigma \cup S \times \Sigma \cup S \cup \{a, c, w, e\} \cup \{\lambda\}$ and the set of transitions Δ is defined as follows:

1. $(q,i) \stackrel{X}{\to} (q, i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
2. $(q,i) \stackrel{(q,Y)}{\to} ((q,Y), i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \le i \le n-1$;
3. $((q,Y),i) \stackrel{X}{\to} ((q,Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \le i \le n-1$;
4. $((q,Y), n) \stackrel{q_1}{\to} (q_1, 0)$ if $\pi(q) = \exists$ and $q_1 \in \gamma(q, Y)$;

5. $(q', i) \xrightarrow{X} (q', i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
6. $(q', i) \xrightarrow{(q,Y)} ((q', Y), i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;
7. $((q', Y), i) \xrightarrow{X} ((q', Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
8. $((q', Y), n) \xrightarrow{q_1} (q'_1, 0)$ if $\pi(q) = \exists$ and $q_1 \in \gamma(q, Y)$;
9. $((q, Y), n) \xrightarrow{a} (\tilde{q}_1, 0)$ if $\pi(q) = \forall$ and $q_1 = \mathit{first}(q, Y)$;
10. $((q, Y), n) \xrightarrow{a} (\tilde{q}_2, 0)$ if $\pi(q) = \forall$ and $q_2 = \mathit{second}(q, Y)$;
11. $((q, Y), n) \xrightarrow{a} ((q_1, q_2), 0)$ if $\pi(q) = \forall$, $q_1 = \mathit{first}(q, Y)$, $q_2 = \mathit{second}(q, Y)$;
12. $((q', Y), n) \xrightarrow{a} (\tilde{q}_1, 0)$ if $\pi(q) = \forall$ and $q_1 = \mathit{first}(q, Y)$;
13. $((q', Y), n) \xrightarrow{a} (\tilde{q}_2, 0)$ if $\pi(q) = \forall$ and $q_2 = \mathit{second}(q, Y)$;
14. $((q_1, q_2), 0) \xrightarrow{q_1} (q_1, 0)$
15. $((q_1, q_2), 0) \xrightarrow{q_2} (q_2, 0)$
16. $(\tilde{q}_1, 0) \xrightarrow{q_1} (q'_1, 0)$
17. $(\tilde{q}_1, 0) \xrightarrow{q_2} (q_2, 0)$
18. $(\tilde{q}_2, 0) \xrightarrow{q_1} (q_1, 0)$
19. $(\tilde{q}_2, 0) \xrightarrow{q_2} (q'_2, 0)$
20. $((q, Y), n) \xrightarrow{w} q_c$ if $\pi(q) = acc$;
21. $(q, i) \xrightarrow{X} (q'', i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
22. $(q, i) \xrightarrow{(q,Y)} ((q'', Y), i+1)Yq$ for all $q \in S$, $0 \leq i \leq n-1$;
23. $((q, Y), i) \xrightarrow{X} ((q'', Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
24. $(q', i) \xrightarrow{X} (q'', i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
25. $(q', i) \xrightarrow{(q,Y)} ((q'', Y), i+1)Yq$ for all $q \in S$, $0 \leq i \leq n-1$;
26. $((q', Y), i) \xrightarrow{X} ((q'', Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
27. $(q'', i) \xrightarrow{X} (q, i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
28. $(q'', i) \xrightarrow{(q,Y)} ((q, Y), i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;
29. $((q'', Y), i) \xrightarrow{X} ((q, Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
30. $((q'', Y), n) \xrightarrow{q_1} (q_1, 0)$ if $\pi(q) = \exists$ and $q_1 \in \gamma(q, Y)$;
31. $(q'', i) \xrightarrow{X} (q'', i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
32. $(q'', i) \xrightarrow{(q,Y)} ((q'', Y), i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;
33. $((q'', Y), i) \xrightarrow{X} ((q'', Y), i+1)X$ for all $q \in S$, $X, Y \in \Sigma$, $0 \leq i \leq n-1$;
34. $((q'', Y), n) \xrightarrow{q_1} (q''_1, 0)$ if $\pi(q) = \exists$ and $q_1 \in \gamma(q, Y)$;
35. $((q'', Y), n) \xrightarrow{a} (\tilde{q}_1, 0)$ if $\pi(q) = \forall$ and $q_1 = \mathit{first}(q, Y)$;
36. $((q'', Y), n) \xrightarrow{a} (\tilde{q}_2, 0)$ if $\pi(q) = \forall$ and $q_2 = \mathit{second}(q, Y)$;
37. $((q'', Y), n) \xrightarrow{a} ((q_1, q_2), 0)$ if $\pi(q) = \forall$, $q_1 = \mathit{first}(q, Y)$, $q_2 = \mathit{second}(q, Y)$;
38. $(q, i) \xrightarrow{c} q_c$ for all $q \in S$, $0 \leq i \leq n-1$;
39. $((q, Y), i) \xrightarrow{c} q_c$ for all $q \in S$, $0 \leq i \leq n$;
40. $(q', i) \xrightarrow{c} q_c$ for all $q \in S$, $0 \leq i \leq n-1$;
41. $((q', Y), i) \xrightarrow{c} q_c$ for all $q \in S$, $0 \leq i \leq n$;

42. $(q'', i) \xrightarrow{c} q''_c$ for all $q \in S, 0 \leq i \leq n-1$;
43. $((q'', Y), i) \xrightarrow{c} q''_c$ for all $q \in S, 0 \leq i \leq n$;
44. $((q'', Y), n) \xrightarrow{w} q''_c$ if $\pi(q) = acc$;

Furthermore, at control-state q''_c the system emits exactly $n+4$ times the action 'c' and then deadlocks. At control-state q_c the system behaves deterministically as follows:

1. First read the top 3 symbols from the stack (while emitting 'c' actions) and remember them.
2. Then pop $n-2$ symbols from the stack (by 'c' actions). Thus, one is at the same position in the previous LBA-configuration that is stored on the stack.
3. Read another 3 symbols from the stack and check if there is an error (according to the transition rules of the LBA). If yes, then deadlock. If no, then emit the special action 'e'.

Finally, we add some extra transitions to make the whole system normed. For every control-state $q \in Q$ we add a transition $q \xrightarrow{\lambda} x$. $x \in Q$ is a special control-state where the stack is emptied. So we have transitions $xG \xrightarrow{\lambda} x$ for every stack symbol $G \in \Gamma$. The system deadlocks when the stack is empty. There are no other transitions at control-state x and the action λ is not enabled anywhere else. This means that if action λ is done once then the system does λ exactly m times (where m is the current size of the stack) and then stops. These extra transitions do not influence the property we want to show, i.e., that $\alpha \not\sim \alpha'$ iff the alternating LBA accepts. This is because throughout the whole bisimulation game between α and α', the size of the stack of α is always the same as the size of the stack of α'. So the attacker cannot win if he chooses a transition with action λ.

The construction above ensures that the attacker plays only in one process (on the α-side; the q-side), while the defender only plays in the other process (on the α'-side; the q'-side). In the important cases the attacker cannot play on the q'-side, because the defender could then immediately make the two processes equal and win. In the rest of the cases it does not matter on which side the attacker plays. In the bisimulation game, configurations of the LBA are pushed onto the stack. The attacker determines which symbols are pushed (rules 1–3). We say that the attacker 'cheats' if he pushes an LBA configuration onto the stack that is not a successor of the previous one (according to the transition rules of the LBA). The attacker also determines the successor-control-state in those cases where the control-state is labeled as existential (rule 4). However, the defender determines the successor-control-state in those cases where the control-state is labeled as universal (rules 9–19). The defender can also, in any step, go from the q' domain of control-states go to the q'' domain of control-states (rules 24–26). By doing so, he threatens to make the two processes equal in the very next step (rules 27–37 and 21–23). The only way for the attacker to avoid this, is to do the action 'c' and go to the control-state q_c, while the defender is forced to go to the control-state q''_c in the other process (rules 38–43). Processes with control states q_c or q''_c a said to be in the 'check-phase'. In the control-state q_c it is checked if the two most recently pushed LBA configurations on the stack have an error at this particular point (according to the transitions of the LBA). In this way it is checked if the attacker has 'cheated' in the

bisimulation game by breaking the rules of the LBA and pushing wrong configurations on the stack. If the attacker has cheated (i.e., an error is found) then the defender wins, since both processes are deadlocked after $n + 4$ 'c'-actions. If the attacker was honest (i.e., there is no error) then the attacker wins, since he can do the action 'e' at the end, and the defender cannot. This construction ensures that the attacker never cheats, i.e., never pushes wrong LBA configurations onto the stack.

We now show that the LBA accepts the input w iff $\alpha \not\sim \alpha'$.

If the LBA accepts w then the attacker has the following winning strategy. The attacker plays honest and in the α process. He pushes a successive sequence of LBA configurations onto the stack. The defender is forced to do the same in the α' process. The attacker gets to choose the successor-control-states at the existential states and the defender chooses the successors at the universal states. Since the attacker plays honest, the defender would lose if he went to the q'' domain of control-states and forced a check-phase. Since the LBA accepts w, the attacker can eventually reach an accepting control-state and then do the action 'w' (rule 20). If the defender is still in the q' domain of control-states, he loses immediately. If the defender is in the q'' domain of control-states then a check-phase is initiated. The attacker will still win after $n + 4$ 'c' actions and the final (winning) 'e' action, since he has not cheated. If the defender initiates the check-phase too early, such that the stack bottom is reached during the check-phase, then the attacker still wins. In this particular case more 'c' actions are possible in q''_c than in q_c. Thus $\alpha \not\sim \alpha'$.

If the LBA does not accept w then the defender has the following winning strategy. If the attacker plays on the α' side then the defender makes the two processes equal. If the attacker does not play honest then the defender goes to the q'' domain and so threatens to make the two processes equal in the next step, unless the attacker does the 'c' action and begins a check-phase. In this check-phase the defender wins after $n + 4$ 'c'-actions (deadlock in both processes), because the attacker has cheated. If the attacker himself goes to the q'' domain of control-states, then the defender can immediately make the two processes equal and win. The definition of the rules 9–19 ensures that the defender gets to choose the successor-control-state at the universal states. Thus, since the LBA does not accept w, the attacker can never reach an accepting control-state (unless by cheating). So the defender can defend forever and wins. Thus $\alpha \sim \alpha'$. □