On the complexity of checking semantic equivalences between pushdown processes and finite-state processes

On the Complexity of Checking Semantic Equivalences between Pushdown Processes and Finite-state Processes[☆]

Antonín Kučera[a,1], Richard Mayr[b]

[a] *Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic*
[b] *School of Informatics, LFCS, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK*

Abstract

Simulation preorder/equivalence and bisimulation equivalence are the most commonly used equivalences in concurrency theory. Their standard definitions are often called strong simulation/bisimulation, while weak simulation/bisimulation abstracts from internal τ-actions.

We study the computational complexity of checking these strong and weak semantic preorders/equivalences between pushdown processes and finite-state processes. We present a complete picture of the computational complexity of these problems and also study fixed-parameter tractability in two important input parameters: x, the size of the finite control of the pushdown process, and y, the size of the finite-state process.

All simulation problems are generally **EXPTIME**-complete and only become polynomial if *both* parameters x and y are fixed.

Weak bisimulation equivalence is **PSPACE**-complete, but becomes polynomial if and only if parameter x is fixed.

Strong bisimulation equivalence is **PSPACE**-complete, but becomes polynomial if *either* parameter x or y is fixed.

Keywords: Pushdown automata, Verification, Simulation, Bisimulation.
ACM Subject Classification: F1.1, F3.1, F4.1, F4.3.

1. Introduction

Semantic equivalence checking is an important technique in formal verification of software systems. The idea is to compare the behavior of a given program (the *implementation*) with its intended behavior (the *specification*). Since the two behaviors are formalized as transition systems, the comparison means proving some kind of semantic equivalence

[☆] This journal paper is mostly based on three conference presentations [33, 41, 36].
URL: `http://www.fi.muni.cz/usr/kucera` (Antonín Kučera), `http://homepages.inf.ed.ac.uk/rmayr` (Richard Mayr)

[1] Antonín Kučera is supported by the research center Institute for Theoretical Computer Science, project No. 1M0545, and by Czech Science Foundation, project No. P202/10/1469.

between the initial states of the two transition systems. Since such proofs cannot be completed by humans for programs of realistic size, a natural question is whether the problem is decidable and what is its complexity. This question has been considered for many computational models and a large number of results have been achieved during the last decade (see [45, 17, 29, 9, 30, 11, 48] for surveys of some subfields).

As semantic equivalences we consider *simulation equivalence* (defined indirectly via *simulation preorder*), as well as *bisimulation equivalence* [46, 44]. We consider both the standard (i.e., strong) variants of these equivalences and the weak variants which abstract from internal τ-actions. These are some of the most important semantic equivalences in the linear/branching time spectrum of van Glabbeek [55, 54].

In this paper we consider pushdown processes, the class of processes whose behavior is definable by pushdown automata (PDA), as well as finite-state systems, the class of processes which are definable by finite-state labeled transition systems. The importance of PDA has recently been recognized also in areas different from theory of formal languages. In particular, PDA are a natural and convenient model for sequential programs with recursive procedure calls (see, e.g., [5, 6, 18, 20, 19]). Global data of such a program is stored in the finite control, and the stack symbols correspond to activation records of individual procedures. A procedure call is thus modeled by pushing a new symbol onto the stack, and a return from the procedure is modeled by popping the symbol from the stack. Consequently, a PDA is seen as a finite description of a computational behavior rather than a language acceptor in this context[2]. The behavior of a given PDA Δ is formally defined by the associated transition system $\mathcal{T}_\Delta$, where the states are configurations of Δ and $p\alpha \stackrel{a}{\rightarrow} q\beta$ if this move is consistent with the transition function of Δ. Hence, $\mathcal{T}_\Delta$ has infinitely many states.

A special subclass of PDA are the *context-free processes*, which correspond to PDA with just one control-state. The class of context-free processes is also called *Basic Process Algebra* [8, 7]. [3] The classes of all pushdown processes, context-free processes and finite-state processes are denoted **PDA**, **BPA** and **FS**, respectively.

Let A, B be classes of processes. The problem whether a given process s of class A is simulated (or weakly simulated) by a given process t of class 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.

Our contribution: We study the computational complexity of these problems, with a particular focus on fixed-parameter tractability w.r.t. two important input parameters:

x: The size the the finite control of the pushdown automaton, i.e., the global data of the considered recursive program. In particular, the finite control has only size 1 for all BPA-processes.

[2] From the language-theoretic point of view, the definition of PDA adopted in this area corresponds to the subclass of real-time PDA. It does not mean that the concept of ε-transitions vanished—it has only been replaced by 'silent' transitions with a distinguished label τ which may (but does not have to) be taken into account by a given semantic equivalence.

[3] This is because stateless PDA correspond to a natural fragment of ACP known as BPA (Basic Process Algebra; see [8]). BPA cannot model global data, but they are sufficiently powerful to model, e.g., the interprocedural data-flow [18]. It is worth noting that the expressive power of PDA is strictly greater than the one of BPA w.r.t. most of the considered semantic equivalences.

y**:** The size of the finite-state process, i.e., the size of the specification.

While these problems are generally **PSPACE**- or **EXPTIME**-complete, most of them become polynomial if one (or both) of these parameters x and y are fixed. Thus, most equivalence checking problems between PDA and finite-state systems are fixed-parameter tractable.

Our results can be summarized as follows (see also the table in Section 7).

- All (strong and weak) simulation problems are generally **EXPTIME**-complete, and the lower bound even holds for the simpler BPA case, i.e., **PDA**/**BPA** $\equiv$ **FS** for $\equiv \in \{\sqsubseteq, \sqsupseteq, \simeq, \sqsubseteq_w, \sqsupseteq_w, \simeq_w\}$. All these problems become polynomial if and only if *both* parameters x and y are fixed.

- Weak bisimulation equivalence (**PDA** $\approx$ **FS**) is generally **PSPACE**-complete. However, it becomes polynomial if and only if parameter x is fixed. In particular, the problem **BPA** $\approx$ **FS** (for $x = 1$) is polynomial.

- Strong bisimulation equivalence (**PDA** $\sim$ **FS**) is generally **PSPACE**-complete. However, it becomes polynomial if *either* parameter x or y is fixed. In other words, this problem is only hard to solve if both parameters x and y are large.

Furthermore, we show that **PDA** $\sim$ **FS** is easier than the general **PDA** $\sim$ **PDA** problem. In the appendix of this paper we show **EXPTIME**-hardness of **PDA** $\sim$ **PDA**.

The results in this journal paper are mostly based on three of our conference presentations [33, 41, 36]. Some earlier (and weaker) results have appeared in [34, 37]. All proofs of theorems are given in this paper, except for Theorem 18. The bisimulation basis construction required for this proof was introduced in [38] and generalized in [40].

Related work: The 'symmetric' equivalence checking problem where two pushdown processes (or context-free processes) are compared to each other has also been studied.

While all simulation problems between PDA/BPA are undecidable [21], some bisimulation problems are decidable.

Baeten, Bergstra, and Klop [7] proved that strong bisimilarity is decidable for *normed* BPA (a PDA is normed if the stack can be emptied from every reachable configuration). Simpler proofs were given later in [14, 22, 26], and there is even a polynomial-time algorithm [24]. The decidability result has been extended to all (not necessarily normed) BPA in [15], and an elementary (**2-EXPTIME**) upper complexity bound is due to [12]. Recently, **PSPACE**-hardness of this problem has been established in [49]. Strong bisimilarity was shown to be decidable also for normed PDA [51]. Later, Sénizergues proved that bisimilarity is decidable for all PDA processes [47]. (See the appendix of this paper for an **EXPTIME** lower bound.)

Weak bisimilarity is undecidable for PDA [50], and in fact for a very modest subclass of PDA known as one-counter nets (Petri nets with only one unbounded place) [42]. Moreover, weak bisimilarity is even undecidable for PDA with just 2 control-states [43]. Is is an open question if weak bisimilarity is decidable for PDA with just 1 control state (i.e., BPA). The best known lower bound for the **BPA** $\approx$ **BPA** problem is **EXPTIME**-hardness [43].

2. Preliminaries

2.1. Transition Systems and Semantic Equivalences

Definition 1. *A* labeled transition systems *is a triple* $\mathcal{T} = (S, Act, \rightarrow)$ *where* S *is a set of* states*,* Act *is a finite set of* actions*, and* $\rightarrow \subseteq S \times Act \times S$ *is a* transition relation*. We write* $s \xrightarrow{a} t$ *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* $s \xrightarrow{w} t$ *for some* $w \in Act^*$*.*

The action τ *is a special 'silent' internal action. The* extended transition relation *'*$\stackrel{a}{\Rightarrow}$*' is defined by* $s \stackrel{a}{\Rightarrow} t$ *iff either* $s = t$ *and* $a = \tau$*, or* $s \xrightarrow{\tau^i} s' \xrightarrow{a} t' \xrightarrow{\tau^j} t$ *for some* $i, j \in \mathcal{N}_0$ *and* $s', t' \in S$*.*

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., [55] for an overview).

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.

Definition 2. *Let* $\mathcal{T} = (S, Act, \rightarrow)$ *be a labeled transition system.*

Simulation *A binary relation* $R \subseteq S \times S$ *is a* simulation *iff for all pairs* $(s, t) \in R$ *and all actions* $a \in Act$ *if* $s \xrightarrow{a} s'$ *there exists some transition* $t \xrightarrow{a} t'$ *such that* $(s', t') \in R$*.*

Simulations are closed under union and the largest simulation relation on $\mathcal{T}$ *is a preorder, denoted by* $\sqsubseteq$*. A process* s *is* simulated *by* t*, written* $s \sqsubseteq t$*, iff there is a simulation* R *such that* $(s, t) \in R$*. Processes* s, t *are* simulation equivalent*, written* $s \simeq t$*, iff they can simulate each other, i.e.,* $s \sqsubseteq t$ *and* $t \sqsubseteq s$*.*

Bisimulation *A symmetric simulation relation is called a* bisimulation*. The largest bisimulation relation is an equivalence called* strong bisimulation*. It is denoted by* $\sim$*.*

Weak simulation *In weak simulation one abstracts from internal* τ *actions. A binary relation* $R \subseteq S \times S$ *is a* weak simulation *iff for all pairs* $(s, t) \in R$ *and all actions* $a \in Act$ *if* $s \xrightarrow{a} s'$ *there exists some transition* $t \stackrel{a}{\Rightarrow} t'$ *such that* $(s', t') \in R$*.*

Weak simulations are closed under union and the largest weak simulation relation on $\mathcal{T}$ *is a preorder, denoted by* $\sqsubseteq_w$*. A process* s *is* weakly simulated *by* t*, written* $s \sqsubseteq t$*, iff there is a weak simulation* R *such that* $(s, t) \in R$*. Processes* s, t *are* weakly simulation equivalent*, written* $s \simeq_w t$*, iff they can simulate each other, i.e.,* $s \sqsubseteq_w t$ *and* $t \sqsubseteq_w s$*.*

Weak Bisimulation *A symmetric weak simulation relation is called a* weak bisimulation*. The largest weak bisimulation relation is an equivalence, which is called* weak bisimulation equivalence *and denoted by* $\approx$*.*

Definition 3. *(Bisimulation up-to k) Let $\mathcal{T} = (S, Act, \rightarrow)$ be a transition system. The relation $\approx_k \subseteq S \times S$ for $k \in \mathcal{N}_0$ is called weak bisimulation up-to k. These relations are defined inductively as follows: $\approx_0 = S \times S$ and for $k > 0$ we have $(s,t) \in \approx_k$ iff*

- *For all $a \in Act$, if $s \stackrel{a}{\Rightarrow} s'$ then there exists some transition $t \stackrel{a}{\Rightarrow} t'$ s.t. $(s',t') \in \approx_{k-1}$, and*
- *For all $a \in Act$, if $t \stackrel{a}{\Rightarrow} t'$ then there exists some transition $s \stackrel{a}{\Rightarrow} s'$ s.t. $(s',t') \in \approx_{k-1}$*

The relation $\sim_k$ for $k \in \mathcal{N}_0$ is called strong bisimulation up-to k. The relations $\sim_k$ are defined analogously to $\approx_k$ with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$.

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.

Simulations and bisimulations can also be viewed as *games* [52, 53] 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. The initial configuration of the game is given as the pair of states (s,t). 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: instead of the relation $\stackrel{a}{\rightarrow}$, the players use the relation $\stackrel{a}{\Rightarrow}$ with possibly several extra weak internal τ steps.

The relations $\sim_k$ and $\approx_k$ also have a game theoretic characterization. We have $s \sim_k t$ (resp. $t \approx_k t$) iff the defender has a strategy by which he can avoid losing the strong (resp. weak) bisimulation game from (s,t) for at least k rounds.

2.2. Temporal Logic and Characteristic Formulae

Hennessy-Milner Logic [23] is a simple modal logic that is interpreted on labeled transition systems $\mathcal{T} = (S, Act, \rightarrow)$. The formulae have the following syntax.

$$\Phi ::= \mathit{true} \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \langle a \rangle \Phi$$

where $a \in Act$.

The denotation $[\![\Phi]\!]$ of a formula Φ is a subset of S, which is defined inductively over the structure of Φ as follows.

$$\begin{aligned} [\![\mathit{true}]\!] &:= S \\ [\![\neg\Phi]\!] &:= S - [\![\Phi]\!] \\ [\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\ [\![\langle a \rangle \Phi]\!] &:= \{ s \in S \mid \exists s' \in S.\ s \stackrel{a}{\rightarrow} s' \ \wedge \ s' \in [\![\Phi]\!] \} \end{aligned}$$

One also writes $s \models \Phi$ for $s \in [\![\Phi]\!]$.

Disjunction can be defined via negation and conjunction by $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$. The universal one-step next operator can be defined by $[a]\Phi = \neg\langle a\rangle\neg\Phi$.

The logic EF extends Hennessy-Milner Logic with an operator $\Diamond$ for reachability whose semantics is defined as follows.

$$[\![\Diamond\Phi]\!] \quad := \quad \{s \in S \mid \exists s' \in S.\ s \xrightarrow{*} s' \ \wedge \ s' \in [\![\Phi]\!]\}$$

EF can also be defined as a fragment of computation-tree logic (CTL) [16].

We consider a slight extension of EF by adding another operator $\Diamond_\tau$ that expresses reachability by a sequence of τ-actions.

$$[\![\Diamond_\tau\Phi]\!] \quad := \quad \{s \in S \mid \exists s' \in S.\ s \xrightarrow{\tau^*} s' \ \wedge \ s' \in [\![\Phi]\!]\}$$

We now show a connection between temporal logic and bisimulation equivalence. We recall some results from [28] (see also [30] for a more recent survey). A characteristic formula of a finite-state system F w.r.t. $\sim$ (resp. $\approx$) is a formula $\Theta_F^{\sim}$ (resp. $\Theta_F^{\approx}$) s.t. for every general system G which uses the same set of actions as F we have that $G \models \Theta_F^{\sim} \iff G \sim F$ (resp. $G \models \Theta_F^{\approx} \iff G \approx F$). It has been shown in [28] that characteristic formulae for finite-state systems w.r.t. $\sim$ and $\approx$ can be effectively constructed in the temporal logic EF (a simple fragment of CTL [16]), by using the following theorem.

Theorem 1. *(from [28])*
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 $g' \approx_n f'$.*

On every system without τ-actions the relation $\approx$ coincides with $\sim$. Therefore, we obtain the following corollary.

Corollary 1. *Let F be a finite-state system with n states and G a general system. States $g \in G$ and $f \in F$ are strongly bisimilar iff the following conditions hold:*

1. *$g \sim_n f$, and*
2. *For each state g' which is reachable from g there is a state $f' \in F$ such that $g' \sim_n f'$.*

Now we construct formulae $\Phi_{k,f}^{\sim}$ for states f in F w.r.t. $\sim_k$ that satisfy $g \models \Phi_{k,f}^{\sim} \iff g \sim_k f$. The family of $\Phi_{k,f}^{\sim}$ formulae is defined inductively on k as follows:

$$\begin{aligned}
\Phi_{0,f}^{\sim} \quad &:= \quad \mathit{true} \\
\Phi_{k+1,f}^{\sim} \quad &:= \quad \left(\bigwedge_{a \in Act} \bigwedge_{f' \in S(f,a)} \langle a\rangle \Phi_{k,f'}^{\sim} \right) \wedge \left(\bigwedge_{a \in Act} ([a](\bigvee_{f' \in S(f,a)} \Phi_{k,f'}^{\sim})) \right)
\end{aligned}$$

where $S(f,a) = \{f' \mid f \stackrel{a}{\rightarrow} f'\}$ and empty conjunctions are equivalent to *true*. The first subformulae of $\Phi^{\sim}_{k+1,f}$ specifies that every attacker move $f \stackrel{a}{\rightarrow} f'$ in the finite-state system F can be matched by a defender move in G. The second subformulae of $\Phi^{\sim}_{k+1,f}$ specifies that every attacker move in G can be matched by some defender move $f \stackrel{a}{\rightarrow} f'$ in the finite-state system F.

Thus, by Corollary 1, the characteristic formula $\Theta^{\sim}_{f}$ (w.r.t. $\sim$) for a process f of a finite-state system $\mathcal{F} = (F, Act, \rightarrow)$ with n states is

$$\Theta^{\sim}_{f} \;=\; \Phi \wedge \neg\Diamond\Psi$$

where $\Phi = \Phi^{\sim}_{n,f}$ and

$$\Psi = \left(\bigwedge_{f' \in F} \neg\Phi^{\sim}_{n,f'} \right)$$

Observe that the formulae Φ and Ψ are Hennessey-Milner Logic formulae [23], since they do not contain the reachability operator $\Diamond$, but only one-step next modalities $\langle a \rangle$ and $[a]$ and boolean operators. The formula $\Theta^{\sim}_{f}$ contains only a single reachability operator. Furthermore, the nesting-depth of one-step next operators in Φ and Ψ is n.

Now we construct the characteristic EF-formula $\Theta^{\approx}_{f}$ (w.r.t. $\approx$) for a process f of a finite-state system $\mathcal{F} = (F, Act, \rightarrow)$ with n states. Without restriction we assume that $\Rightarrow = \rightarrow$ in $\mathcal{F}$, i.e., that the transitive closure w.r.t. τ-transitions is already computed in F (this can be done in polynomial time).

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

$$\begin{aligned} \Phi_{0,f} &:= \mathit{true} \\ \Phi_{k+1,f} &:= \left(\bigwedge_{a \in Act} \bigwedge_{f' \in S(f,a)} \Diamond_a \Phi_{k,f'} \right) \wedge \left(\bigwedge_{a \in Act} (\neg\Diamond_a (\bigwedge_{f' \in S(f,a)} \neg\Phi_{k,f'})) \right) \end{aligned}$$

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*. The first subformulae of $\Phi^{\approx}_{k+1,f}$ specifies that every attacker move $f \stackrel{a}{\Rightarrow} f'$ in the finite-state system F can be matched by a long defender move in G. The second subformulae of $\Phi^{\approx}_{k+1,f}$ specifies that every long attacker move in G can be matched by some defender move $f \stackrel{a}{\Rightarrow} f'$ in the finite-state system F.

Thus, by Theorem 1, the characteristic formula $\Theta^{\approx}_{f}$ for a process f of a finite-state system $\mathcal{F} = (F, Act, \rightarrow)$ with n states is

$$\Theta^{\approx}_{f} \;=\; \Phi^{\approx}_{n,f} \wedge \neg\Diamond \left(\bigwedge_{f' \in F} \neg\Phi^{\approx}_{n,f'} \right)$$

In general, the characteristic formula $\Theta^{\approx}_{f}$ has exponential size in $n = |F|$ if it is represented as a tree of subformulae. However, $\Theta^{\approx}_{f}$ has only a polynomial number of *different*

subformulae (the $O(n^2)$ formulae $\Phi^{\approx}_{k,f}$ for $0 \leq k \leq n$ and $f' \in F$ where $|F| = n$). Thus $\Theta^{\approx}_{f}$ can be compactly represented by a DAG (directed acyclic graph) of polynomial size, instead of a tree of exponential size. In this case the construction of $\Theta^{\approx}_{f}$ takes only polynomial time.

2.3. Pushdown Processes

In this paper we mainly consider processes that are described by *pushdown automata* (PDA). Here the PDA are not used as language acceptors, but as a model of sequential systems with mutually recursive procedures. In order to emphasize this point, we speak of *pushdown processes*. Furthermore, pushdown processes are usually defined to have a particular initial state (see Subsection 2.4).

Definition 4. *A pushdown automaton (**PDA**) 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 δ is a finite set of transition rules, which have one of the following forms.*

- $p \xrightarrow{a} q$*, where $p, q \in Q$ and $a \in Act$.*
- $p \xrightarrow{a} q\beta$*, where $p, q \in Q$, $a \in Act$ and $\beta \in \Gamma^*$.*
- $pA \xrightarrow{a} q\beta$*, where $p, q \in Q$, $a \in Act$, $A \in \Gamma$ and $\beta \in \Gamma^*$.*

Δ induces a labeled transition system $\mathcal{T}_\Delta = (S, Act, \rightarrow)$, where $S = Q \times \Gamma^*$ is the set of states (we simply write $p\alpha$ instead of (p, α)), Act is the set of actions, and the transition relation is defined as follows: If $(p \xrightarrow{a} q) \in \delta$ then we have $p\alpha \xrightarrow{a} q\alpha$ for every $\alpha \in \Gamma^*$. If $(p \xrightarrow{a} q\beta) \in \delta$ then we have $p\alpha \xrightarrow{a} q\beta\alpha$ for every $\alpha \in \Gamma^*$. If $(pA \xrightarrow{a} q\beta) \in \delta$ then we have $pA\alpha \xrightarrow{a} q\beta\alpha$ for every $\alpha \in \Gamma^*$.

Note that in configurations where the stack is empty (denoted as $p\varepsilon$) only transition rules of the first two types are applicable.

As a shorthand notation, we use a rule of the form $A \xrightarrow{a} B$ (where $A, B \in \Gamma$) to denote the entire set of rules $\{pA \xrightarrow{a} pB \mid p \in Q\}$.

We can assume (w.l.o.g.) that each transition increases the height (or length) of the stack by at most one (i.e., $|\beta| \leq 2$), since each PDA can be efficiently transformed into this kind of normal form.

***BPA** (basic process algebra [8]), also called context-free processes, is the subclass of **PDA** where $|Q| = 1$, i.e., without a finite control. In this case we do not write the control-state, i.e., we wite α instead of $p\alpha$.*

2.4. The Problem

We consider the complexity of checking semantic equivalences between pushdown processes and finite-state systems. The semantic equivalences we consider are strong and weak simulation preorder and equivalence and strong and weak bisimulation equivalence.

Let $P = (Q, \Gamma, Act_P, \delta)$ be a pushdown automaton and $\mathcal{T}_P = (S_P, Act_P, \rightarrow_P)$ be the induced labeled transition system according to Definition 4. Let $F = (S_F, Act_F, \rightarrow_F)$ be a finite-state system and $\equiv$ be a preorder/equivalence $\in \{\sqsubseteq, \sqsupseteq, \simeq, \sqsubseteq_w, \sqsupseteq_w, \simeq_w, \sim, \approx\}$. We require that initial states $p_0\alpha_0 \in S_P$ of P and $s_0 \in S_F$ of F are defined.

We say that P and F are in preorder/equivalence relation $\equiv$ (denoted by $P \equiv F$) iff their respective initial states are in preorder/equivalence relation in the combined

transition system, i.e., iff we have that $p_0\alpha_0 \equiv s_0$ in the combined labeled transition system $(S_P \cup S_F, Act_P \cup Act_F, \rightarrow_P \cup \rightarrow_F)$. Using this notation, we can now define the preorder/equivalence checking problems for $\equiv \in \{\sqsubseteq, \sqsupseteq, \simeq, \sqsubseteq_w, \sqsupseteq_w, \simeq_w, \sim, \approx\}$.

PDA $\equiv$ FS

Instance: A pushdown process P and a finite-state system F.
Question: $P \equiv F$?

The complexity of many of these problems depends on the size of both input parameters, particularly the size of the finite-state system and the size of the finite control of the pushdown automaton. In some cases the problems are fixed-parameter tractable, i.e., become polynomial if the size one of the input parameters (e.g., the finite control of the PDA) is fixed. Other problems are **PSPACE**- or **EXPTIME**-hard, even if some input parameters are fixed. Thus we study the following questions.

- What is the general complexity of **PDA $\equiv$ FS** ?
- What is the complexity of **PDA $\equiv$ FS** if the size of the finite-state system is fixed?
- What is the complexity of **PDA $\equiv$ FS** if the size of the finite control of the PDA is fixed? In particular, what if the finite control of the PDA has size 1, i.e., what is the complexity of **BPA $\equiv$ FS** ?
- What is the complexity of **PDA $\equiv$ FS** if both the size of the finite control of the PDA and the size of the finite system are fixed? (Note that there are still infinitely many non-trivial instances in this case, because of the transition rules of the PDA.)

In this paper we give a complete picture of the computational complexity of all these problems of semantic preorder/equivalence checking between pushdown processes and finite-state systems.

3. Strong Simulation Preorder and Equivalence

We consider strong simulation preorder between PDA and finite-state systems in both directions, and simulation equivalence. For $\equiv \in \{\sqsubseteq, \sqsupseteq, \simeq\}$ we consider the problem.

PDA $\equiv$ FS

Instance: A pushdown process P and a finite-state system F.
Question: $P \equiv F$?

We show that all three problems (for $\equiv \in \{\sqsubseteq, \sqsupseteq, \simeq\}$) have the same complexity. The problems are **EXPTIME**-complete in general, and only fixed-parameter tractable if *both* the size of F and the size of the finite control of P are fixed. If just one of these parameters is fixed then the problem stays **EXPTIME**-hard.

The following table summarizes the complexity results for strong simulation preorder/equivalence. In the cases where a parameter size is fixed, the upper bounds are interpreted as 'for every fixed size', while the lower bounds are interpreted as 'for some fixed size'.

Complexity	General	fixed PDA control (even size 1; BPA)	fixed F	both fixed
PDA $\sqsubseteq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\sqsupseteq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\simeq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**

3.1. Upper Bounds

If the automata do not contain any internal τ-transitions then strong and weak simulation equivalence coincide. Thus all upper complexity bounds for weak simulation carry over to strong simulation. The results of Subsection 4.1 imply the following two theorems.

Theorem 2. *The problems* ***PDA*** $\sqsubseteq$ ***FS****,* ***PDA*** $\sqsupseteq$ ***FS****, and* ***PDA*** $\simeq$ ***FS*** *are in* ***EXPTIME****.*

Proof. Directly from Theorem 9. □

Theorem 3. *If the size of the finite control of the PDA and the size of the finite-state system are bounded by fixed constants, then the problems* ***PDA*** $\sqsubseteq$ ***FS****,* ***PDA*** $\sqsupseteq$ ***FS****, and* ***PDA*** $\simeq$ ***FS*** *are decidable in polynomial time.*

Proof. Directly from Theorem 10. □

3.2. Lower Bounds

The **EXPTIME** lower bounds in this section are shown by a reduction from the **EXPTIME**-complete acceptance problem for alternating linear-bounded automata (LBA).

First we show that the problem **BPA** $\sqsubseteq$ **FS** is **EXPTIME**-hard, and then we show that the problem **PDA** $\sqsubseteq$ **FS** is **EXPTIME**-hard even for a *fixed* finite-state system. Then we show the same **EXPTIME**-hardness result for the reverse direction $\sqsupseteq$. Finally, we show **EXPTIME**-hardness of $\simeq$ by a simple reduction from $\sqsubseteq$ to $\simeq$.

Definition 5. An *alternating LBA* [56] is an alternating Turing machine [25] whose tape is linearly bounded in the size of the input word. An alternating LBA $\mathcal{M}$ is described by 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), Σ is the set of tape symbols, $\vdash, \dashv \in \Sigma$ are the left-end and right-end markers of the tape, 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. A configuration of the LBA is of the form $usAv$, such that

- $u \in \Sigma^*$ is the tape content to the left of the read-write head
- $s \in S$ is the current control-state

- $A \in \Sigma$ is the tape symbol located at the current head position
- $v \in \Sigma^*$ is the tape content to the right of the read-write head
- If the head is at the leftmost position then $u = \varepsilon$ and $A = \vdash$, otherwise the first symbol of u is $\vdash$.
- If the head is at the rightmost position then $v = \varepsilon$ and $A = \dashv$, otherwise the last symbol of v is $\dashv$.

The transition function $\gamma : S \times \Sigma \mapsto 2^{S \times \Sigma \times \{L,R\}}$ describes the dynamics of the system. To any control-state $s \in S$ and tape symbol $A \in \Sigma$ at the current head position, it assigns a set of possible moves. Each move is described by a tuple from $S \times \Sigma \times \{L, R\}$, i.e., a move consists of a new control-state, a new tape symbol which is written to the tape at the current head position, and a direction L (left) or R (right) for the read-write head to move to.

The head cannot move beyond the left-and and right-end markers, and these markers cannot be removed. This is ensured by the following restrictions on δ.

- $(t, B, D) \in \delta(s, \vdash)$ implies $D = R$ and $B = \vdash$.
- $(t, B, D) \in \delta(s, \dashv)$ implies $D = L$ and $B = \dashv$.

This induces a transition system on the set of configurations as follows.

- If $(t, B, R) \in \gamma(s, A)$ then $usAv \to uBtv$
- If $(t, B, L) \in \gamma(s, A)$ then $uCsAv \to utCBv$

where $A, B, C \in \Sigma$, $s, t \in S$ and $u, v \in \Sigma^*$.

We assume (w.l.o.g.) there are either exactly two moves (in normal configurations) or none (in final configurations). I.e., γ satisfies the following two conditions:

- for all $s \in S$ and $A \in \Sigma$ such that $\pi(s) = \forall$ or $\pi(s) = \exists$ we have that $|\gamma(s, A)| = 2$. We fix an order on the two possible moves, i.e., on the elements of $\gamma(s, A)$. So $\gamma(s, A) = \{(s_1, A_1, D_1,), (s_2, A_2, D_2)\}$ for some $s_1, s_2 \in S$, $A_1, A_2 \in \Sigma$ and $D_1, D_2 \in \{L, R\}$. We then define $\mathit{first}(s, A) := s_1$ as the new control-state of the first move and $\mathit{second}(s, A) := s_2$ as the new control-state of the second move. (It is possible that $s_1 = s_2$.) Therefore, each configuration of $\mathcal{M}$ where the control-state is universal or existential has exactly two immediate successors (configurations reachable in one computation step).
- for all $s \in S$ and $A \in \Sigma$ such that $\pi(s) = \mathit{acc}$ or $\pi(s) = \mathit{rej}$ we have that $\gamma(s, A) = \emptyset$, i.e., each configuration of $\mathcal{M}$ where the control-state is accepting or rejecting is 'terminated' (without any successors).

A *computation-tree* for $\mathcal{M}$ on an input 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; see above);
- 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 exists a computation-tree T such that all leafs of T are accepting configurations. The acceptance problem for alternating LBA is known to be **EXPTIME**-complete [56].

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 4. *The problem* $\boldsymbol{BPA \sqsubseteq FS}$ *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 process $\Delta = (\Gamma, Act, \delta)$ and a process α of Δ and a finite-state system $\mathcal{F} = (S', Act, \rightarrow)$ and a process X of $\mathcal{F}$, such that $\mathcal{M}$ accepts w iff $\alpha \not\sqsubseteq X$.

The idea for the construction is that the simulation game between α and X constructs a branch in a computation-tree of $\mathcal{M}$ on w. Since the attacker wants to show that $\mathcal{M}$ accepts w, he gets to choose the successor state at the existential control-states, while the defender gets to choose the successor state at the universal control-states. The attacker gets to choose the tape symbols in the encoded configurations of $\mathcal{M}$, which are stored on the stack of α. In order to avoid 'cheating' by the attacker (i.e., choosing tape symbols which do not encode a correct computation of $\mathcal{M}$ on w), the defender can always demand a 'check', which he wins if the attacker cheated. Thus the attacker can only win the simulation game if he can reach an accepting configuration of $\mathcal{M}$ in a correct simulation of its computation on w.

Let n be the length of w. We let

$$\Gamma = S_\star \times \Sigma \ \cup \ S \times \Sigma_\star \times \{0, \cdots, n+2\} \ \cup \ S \times \Sigma \times \{W\} \ \cup \ \{T, Z\}$$

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).

Unlike pushdown automata, BPA processes do not have a finite control. However, the current symbol at the top of the stack can be used in this function (i.e., as a finite memory), as long as the height of the stack does not decrease. Symbols from the set $S \times \Sigma_\star \times \{0, \cdots, n+2\}$ are used as top stack symbols when pushing a new configuration onto 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 $A \in \Sigma$, $s \in S$, $0 \le i \le n+1$;
2. $(\langle s, \star\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle s, A\rangle$ — for all $A \in \Sigma$, $s \in S$, $0 \le i \le n+1$;
3. $(\langle s, A\rangle, i) \xrightarrow{a} (\langle s, A\rangle, i+1)\,\langle \star, B\rangle$ — for all $A, B \in \Sigma$, $s \in S$, $0 \le i \le n+1$;
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 BPA process α of the system Δ is defined as follows.

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

It encodes the initial configuration of $\mathcal{M}$, with the input word w stored on the stack in reverse order. 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. $(C_i, i{+}1, y, \star, \star, \star) \xrightarrow{z} (C_i, (i{+}2) \mod (n{+}2), y, z, \star, \star)$ for all $0 \le i \le n$ and $y, z \in S_\star \times \Sigma$;
9. $(C_i, j, y, z, \star, \star) \xrightarrow{u} (C_i, (j{+}1) \mod (n{+}2), y, z, \star, \star)$ for all $0 \le i \le n$, $i{+}2 \le j \le n{+}1$, and $y, z, u \in S_\star \times \Sigma$;
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. $(C_i, i{+}1, y, z, u, \star) \xrightarrow{v} (C_i, (i{+}2) \mod (n{+}2), y, z, u, v)$ for all $0 \le i \le n$ and $y, z, u, v \in S_\star \times \Sigma$;
13. $(C_i, (i{+}2) \mod (n{+}2), y, z, u, v) \xrightarrow{x} U$ 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).

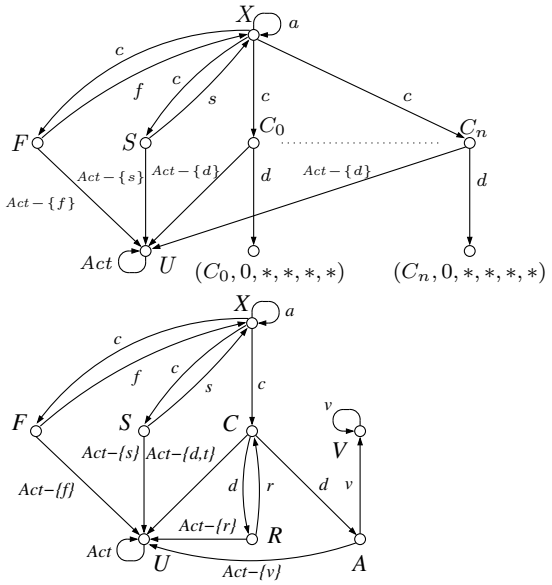

Figure 1: The systems $\mathcal{F}$ and $\mathcal{F}'$ (successors of $(C_i, 0, \star, \star, \star, \star)$ in $\mathcal{F}$ are omitted).

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.

As already mentioned above, the BPA process α is defined as the process in Δ with initial state

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

Similarly the finite-state process X is defined as the process in $\mathcal{F}$ with initial state X.

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 computation-tree for $\mathcal{M}$ on w. The attacker (who plays with α) wants to show that there is an accepting computation-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 attacker 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 5. *The problem* $\boldsymbol{PDA \sqsubseteq FS}$ *is* ***EXPTIME****-hard even for a fixed finite-state process.*

Proof. We modify the construction of Theorem 4. Intuitively, we just re-implement the cheating detection so that the compatibility of selected pairs of symbols is checked by the pushdown automaton 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 automaton $(Q, \Gamma, Act, \delta')$, where Γ is the same as in Theorem 4, 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 4) 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 4 augmented with the control-state g.

The proof follows the line of Theorem 4. 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 $p_1X \xrightarrow{d} p_1X$ 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. □

Now the show the **EXPTIME** lower bounds for the other direction of simulation.

Theorem 6. *The problem* $\boldsymbol{FS \sqsubseteq BPA}$ *is* ***EXPTIME****-hard.*

Proof. The technique is similar to the one of Theorem 4. 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 4;
4. $(C_i, (i{+}2) \mod (n{+}2), y, z, u, v) \xrightarrow{t} X$ 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 computation-tree for $\mathcal{M}$ on w. The attacker (who plays with X now) wants to show that there is an accepting computation-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. □

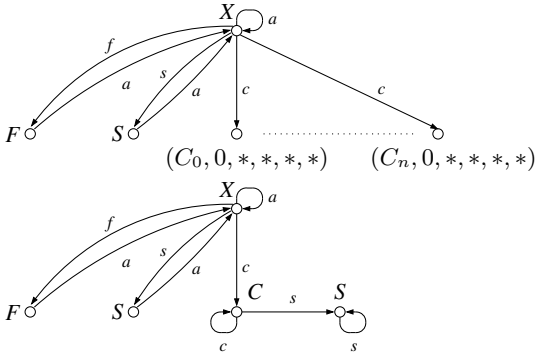

Figure 2: The systems $\bar{\mathcal{F}}$ and $\bar{\mathcal{F}}'$ (successors of $(C_i, 0, \star, \star, \star, \star)$ in $\bar{\mathcal{F}}$ are omitted).

Theorem 7. *The problem* $\boldsymbol{FS \sqsubseteq PDA}$ *is* ***EXPTIME****-hard even for a fixed finite-state process.*

Proof. The required modification of the proof of Theorem 6 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 automaton, 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 automaton (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. □

Finally, we show the **EXPTIME** lower bound for checking simulation equivalence.

Theorem 8. *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 $p \sqsubseteq q$. □

4. Weak Simulation Preorder and Equivalence

We consider weak simulation preorder between PDA and finite-state systems in both directions, and weak simulation equivalence. For $\equiv \in \{\sqsubseteq_w, \sqsupseteq_w, \simeq_w\}$ we consider the problem.

PDA $\equiv$ FS
Instance: A pushdown process P and a finite-state system F.
Question: $P \equiv F$?

We show that all three problems (for $\equiv \in \{\sqsubseteq_w, \sqsupseteq_w, \simeq_w\}$) have the same complexity. The problems are **EXPTIME**-complete in general, and only fixed-parameter tractable if *both* the size of F and the finite control of P are fixed. If just one of these parameters is fixed then the problem stays **EXPTIME**-hard.

The following table summarizes the complexity results for strong simulation preorder/equivalence. In the cases where a parameter size is fixed, the upper bounds are interpreted as 'for every fixed size', while the lower bounds are interpreted as 'for some fixed size'.

Complexity	General	fixed PDA control (even size 1; BPA)	fixed F	both fixed
PDA $\sqsubseteq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\sqsupseteq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\simeq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**

4.1. Upper Bounds

Theorem 9. *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 [58]).

Let

$$\varphi := \nu X.\Box_a \Diamond_b \langle c \rangle X$$

where $\Box_a \psi = \nu Y.(\psi \wedge [a]Y)$ and $\Diamond_b \psi = \mu Z.(\psi \vee \langle b \rangle Z)$. 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 (see [32, 16] for a precise definition of the syntax and semantics of the modal μ-calculus). Now let $\Delta = (Q, \Gamma, Act, \delta)$ be a pushdown automaton, $\mathcal{F} = (F, Act, \rightarrow)$ a finite-state system, $p\alpha$ a process of Δ, and f a process of $\mathcal{F}$. We construct a pushdown automaton $\Delta = (Q \times F \times Act \times \{0,1\}, \Gamma \cup \{Z\}, \{a, b, c\}, \delta')$ (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

$$p\alpha \sqsubseteq_w f \quad \text{iff} \quad (p, f, \tau, 0)\alpha Z \models \varphi$$

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). □

Theorem 10. *If the size of the finite control of the PDA and the size of the finite-state system are bounded by fixed constants k_1 and k_2, respectively, then the problems* ***PDA*** $\sqsubseteq_w$ ***FS***, ***PDA*** $\sqsupseteq_w$ ***FS***, *and* ***PDA*** $\simeq_w$ ***FS*** *are decidable in polynomial time.*

PROOF. The complexity result of [58] 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 our given PDA (with $\leq k_1$ control-states) with a given (fixed) finite-state process (with $\leq k_2$ states), as in Theorem 9 we obtain a pushdown automaton with a *fixed* number of control-states, and the result follows. (In other words, the algorithm in the proof of Theorem 9 is only exponential in $k_1 * k_2$ and polynomial if k_1, k_2 are fixed.) □

4.2. Lower Bounds

If the automata do not contain any internal τ-transitions then strong and weak simulation equivalence coincide. Thus all lower complexity bounds for strong simulation carry over to weak simulation. The results of Subsection 3.2 imply the following two theorems.

Theorem 11. *The three problems* $\boldsymbol{BPA} \sqsubseteq_w \boldsymbol{FS}$, $\boldsymbol{BPA} \sqsupseteq_w \boldsymbol{FS}$, *and* $\boldsymbol{BPA} \simeq_w \boldsymbol{FS}$ *are all* ***EXPTIME****-hard.*

PROOF. Directly from the Theorems 4,6 and 8. □

Theorem 12. *The three problem* $\boldsymbol{PDA} \sqsubseteq_w \boldsymbol{FS}$, $\boldsymbol{PDA} \sqsupseteq_w \boldsymbol{FS}$ *and* $\boldsymbol{PDA} \simeq_w \boldsymbol{FS}$ *are* ***EXPTIME****-hard, even for a fixed finite-state process.*

PROOF. Directly from the Theorems 5, 7 and 8. □

5. Strong Bisimulation Equivalence

We consider the following problem.

PDA $\sim$ FS
Instance: A pushdown process P and a finite-state system F.
Question: $P \sim F$?

We show that this problem is **PSPACE**-complete in general, but fixed-parameter tractable. If either the size of F or the size of the finite control of P is fixed, then the problem is polynomial.

The following table summarizes the complexity results for strong bisimulation. In the cases where a parameter size is fixed, the upper bounds are interpreted as 'for every fixed size', while the lower bounds are interpreted as 'for some fixed size'.

Complexity	General	fixed PDA control	fixed F	both fixed
PDA $\sim$ FS	**PSPACE**-complete	**P**	**P**	**P**

5.1. Upper Bounds

If the automata do not contain any internal τ-transitions then strong and weak bisimulation equivalence coincide. Thus all upper complexity bounds for weak bisimulation carry over to strong bisimulation. The results of Subsection 6.1 imply the following two theorems.

Theorem 13. ***PDA*** $\sim$ ***FS*** *is decidable in polynomial space.*

Proof. Directly from Theorem 17. □

Theorem 14. *If the size of the finite control of the PDA is bounded by a fixed constant k, then the problem* ***PDA*** $\sim$ ***FS*** *is decidable in polynomial time.*

Proof. Directly from Theorem 18. □

If the finite-state system F is fixed then **PDA** $\sim$ **FS** is also polynomial. (However, this result only holds for strong bisimilarity, not for weak bisimilarity (see Theorem 19)). The proof is done by a reduction to a model checking problem for pushdown automata with the characteristic formula for the finite-state system w.r.t. strong bisimulation.

Theorem 15. *Let F be a fixed finite-state system. For every pushdown process P, checking if $P \sim F$ requires only polynomial time in the size of P.*

Proof. Using the construction from Subsection 2.2, one can reduce the problem $P \sim F$ to a model checking problem for the pushdown process P and a formula in the temporal logic EF (a fragment of CTL [16]). Let f be the initial state of F and $\Theta_f^\sim = \Phi \wedge \neg\Diamond\Psi$ the characteristic formula for F w.r.t. $\sim$, as defined in Subsection 2.2. The formulae Φ and Ψ are Hennessey-Milner Logic formulae [23], and the nesting-depth of one-step next operators in Φ and Ψ is $n = |F|$. Note that $\Theta_f^\sim$ and n are fixed, since F is fixed. We have $P \sim F \iff P \models \Theta_f^\sim$.

Let m be the size of (the description of) P. We assume that P has $\leq m$ control-states and $\leq m-1$ different stack symbols. Now we show that this model checking problem can be solved in time polynomial in m. Let P' be a state that is reachable from P. The nesting-depth of one-step next operators in the Hennessy-Milner Logic formulae Φ and Ψ is n. Therefore, it depends only on the first n steps of the computations of P and P' whether they satisfy Φ and Ψ, respectively. Thus, it depends only on the control-states of P and P' and on the first n stack symbols of P and P' whether they satisfy Φ and Ψ, respectively, because deeper stack symbols cannot be accessed in n steps.

There are at most m different possibilities for the control-state and at most m^n different possibilities for the first n stack symbols (including the cases where the stack has height $< n$). For each of these m^{n+1} configurations with stack-height $\leq n$, we check if it satisfies Φ or Ψ. Each of those checks can be done in $\mathcal{O}(m^n)$ time. Furthermore, for each α of these n^{m+1} configurations with stack-height $\leq n$, we check if P can reach some configuration of the form $\alpha\beta$ for some β. (where β represents the stack contents below the first n stack symbols. So β does not matter for Φ and Ψ.) Each of those generalized reachability-checks can be done in $\mathcal{O}(m^3n^2)$ time [10].

Therefore the whole property $P \models \Theta_f^\sim$ can be checked in time $\mathcal{O}(m^n + m^{n+1} * m^n + m^{n+1} * m^3n^2) = \mathcal{O}(m^{2n+1}n^2)$. Thus the time used by the algorithm is polynomial in m, the size of P, but exponential in n. □

5.2. Lower Bounds

The **PSPACE** lower bound for the general **PDA** $\sim$ **FS** problem is shown by a reduction from the **PSPACE**-complete problem of quantified boolean formulae (QBF) [56].

Let $n \in \mathcal{N}$ and let $x_1, \ldots, x_n$ be boolean variables. Without restriction we assume that n is even. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. For technical reasons we use the variables in descending index order $x_n, x_{n-1}, \ldots, x_1$. The quantified boolean formula Q is given by

$$Q := \forall x_n \exists x_{n-1} \ldots \forall x_2 \exists x_1 (Q_1 \wedge \cdots \wedge Q_k)$$

where the Q_i are clauses. The **PSPACE**-complete problem is if Q is true. We reduce this problem to the bisimulation problem by constructing a pushdown process P and a finite-state system F s.t. Q is true iff $P \sim F$.

F is defined as follows: The initial state is s_n.

$$\begin{array}{lcll}
s_i & \xrightarrow{a} & s_{i-1} & \text{for } 1 \leq i \leq n \\
t_i & \xrightarrow{a} & t_{i-1} & \text{for } 1 \leq i \leq n \\
s_i & \xrightarrow{a} & t_{i-1} & \text{for } 1 \leq i \leq n \text{ where } i \text{ is odd} \\
t_i & \xrightarrow{a} & s_{i-1} & \text{for } 1 \leq i \leq n \text{ where } i \text{ is even} \\
s_0 & \xrightarrow{a} & u & \\
t_0 & \xrightarrow{a} & u & \\
u & \xrightarrow{a} & u & \\
t_0 & \xrightarrow{a} & w_n & \\
w_i & \xrightarrow{a} & w_{i-1} & \text{for } 1 \leq i \leq n
\end{array}$$

Note that the size of F is not fixed, but linear in n. Figure 3 illustrates the construction.

Now we define the pushdown process P. Initially the stack is empty and the initial control-state is p_n. For $1 \leq j \leq k$ and $1 \leq l \leq n$ we define $Q_j(X_l)$ iff X_l makes the clause Q_j true and $Q_j(\bar{X}_l)$ iff $\bar{X}_l$ makes Q_j true. The transitions of P are as follows:

$$\begin{array}{lcll}
p_i & \xrightarrow{a} & p_{i-1} X_i & \text{for } 1 \leq i \leq n \\
p_i & \xrightarrow{a} & p_{i-1} \bar{X}_i & \text{for } 1 \leq i \leq n \\
p_i & \xrightarrow{a} & t_{i-1} & \text{for } 1 \leq i \leq n \text{ where } i \text{ is odd} \\
p_i & \xrightarrow{a} & s_{i-1} & \text{for } 1 \leq i \leq n \text{ where } i \text{ is even} \\
p_0 & \xrightarrow{a} & q_j & \text{for } 0 \leq j \leq k \\
q_0 & \xrightarrow{a} & q_0 & \\
q_j X_l & \xrightarrow{a} & q_j X_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(X_l). \\
q_j X_l & \xrightarrow{a} & q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(X_l). \\
q_j \bar{X}_l & \xrightarrow{a} & q_j \bar{X}_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(\bar{X}_l). \\
q_j \bar{X}_l & \xrightarrow{a} & q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(\bar{X}_l).
\end{array}$$

Additionally we define that in the control-states s_i and t_i the stack is ignored and the systems behaves just like s_i and t_i in the system F of Figure 3.

We now show that Q is true iff $P \sim F$ (i.e., iff $p_n \varepsilon \sim s_n$).

First we need some terminology to describe possible stack contents of P that encode variable assignments for a subset of the variables of the form $x_n, \ldots, x_{m+1}$ (for some m

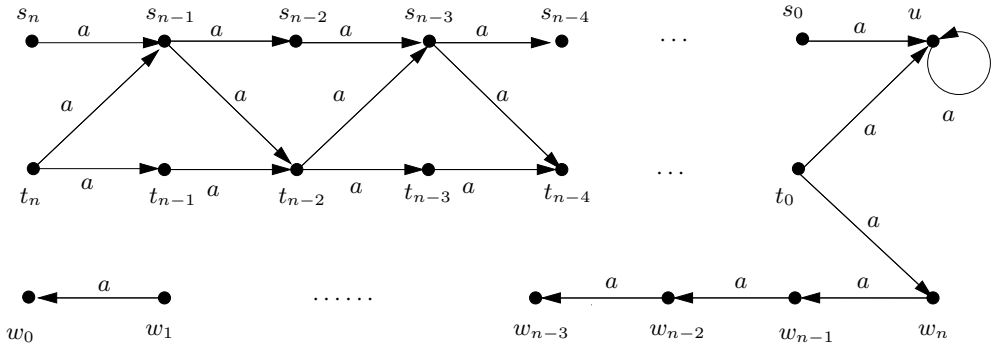

Figure 3: The finite-state system F used in the reduction from QBF to strong bisimulation.

with $n \geq m \geq 0$). Let V_m be the set of all strings of the form $\alpha_n \alpha_{n-1} \ldots \alpha_{m+1}$, where α_i is either X_i or $\bar{X}_i$. For every such string $\alpha \in V_m$ we define the formula Q_α as the formula obtained from Q by setting the variables $x_n x_{n-1} \ldots x_{m+1}$ according to α. So, if m is even, we obtain $Q_\alpha = \forall x_m \exists x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$ and if m is odd we obtain $Q_\alpha = \exists x_m \forall x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$. In particular, if $m = n$ then $\alpha = \varepsilon$ and $Q_\alpha = Q$. On the other hand, if $m = 0$ then all variables in Q are set by α.

Lemma 1. *Let $n \geq m \geq 0$ and $\alpha \in V_m$.*

1. $s_m \not\sim t_m$
2. $s_m \sim p_m\alpha$ *iff* Q_α *is true*
3. $t_m \sim p_m\alpha$ *iff* Q_α *is false*

Proof. We prove the property by induction on m.

The base case is $m = 0$. Here all variables are already set by α.

1. The attacker has the following winning strategy in the bisimulation game between s_0 and t_0. The attacker moves $t_0 \xrightarrow{a} w_n$ and the defender can only respond by $s_0 \xrightarrow{a} u$. Now u has an infinite 'a'-loop while w_n can only do exactly n 'a'-actions. Thus $u \not\sim w_n$ and $s_0 \not\sim t_0$.
2. If Q_α is true then all clauses $Q_j(\alpha)$ are true. Thus all possible successor states $q_j\alpha$ of $p_0\alpha$ in the pushdown process have the same behavior, an infinite 'a'-loop. Therefore $u \sim q_j\alpha$ for all $0 \leq j \leq k$ and finally $s_0 \sim p_0\alpha$.
3. If Q_α is false then there exists at least one $1 \leq j \leq k$ s.t. $Q_j(\alpha)$ is false. Consider the bisimulation game between t_0 and $p_0\alpha$. If the attacker moves $t_0 \xrightarrow{a} u$ then the defender responds by $p_0\alpha \xrightarrow{a} q_0\alpha$ (and vice-versa). If the attacker moves $t_0 \xrightarrow{a} w_n$ then the defender responds by $p_0\alpha \xrightarrow{a} q_j\alpha$ (and vice-versa). If the attacker moves $p_0\alpha \xrightarrow{a} q_{j'}\alpha$ for some j' where $Q_{j'}(\alpha)$ is false then the defender responds by $t_0 \xrightarrow{a} w_n$. If the attacker moves $p_0\alpha \xrightarrow{a} q_{j'}\alpha$ for some j' where $Q_{j'}(\alpha)$ is true then the defender responds by $t_0 \xrightarrow{a} u$. In any case the defender wins and we have $t_0 \sim p_0\alpha$.

Finally, the 'only-if' part of (2) and (3) follows from the fact $s_0 \not\sim t_0$ which has been shown in (1).

The induction hypothesis is that the property already holds for $m-1$. Now we assume $n \geq m \geq 1$ and show the induction step from $m-1$ to m.

1. If m is even then the attacker moves $t_m \xrightarrow{a} t_{m-1}$ and the defender can only respond by $s_m \xrightarrow{a} s_{m-1}$. The resulting pair is non-bisimilar by induction hypothesis. Thus the attacker can win and $s_m \not\sim t_m$.
   If m is odd then the attacker moves $s_m \xrightarrow{a} s_{m-1}$ and the defender can only respond by $t_m \xrightarrow{a} t_{m-1}$. The resulting pair is non-bisimilar by induction hypothesis. Thus the attacker can win and $s_m \not\sim t_m$.
2. Consider the case where Q_α is true. We handle odd and even values of m separately.
   (a) If m is even then $Q_\alpha = \forall x_m \exists x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$ is true. Since x_m is universally quantified, the formula remains true whatever value is chosen for x_m, i.e., both $Q_{\alpha X_m}$ and $Q_{\alpha \bar{X}_m}$ are true.
   If the attacker moves $p_m\alpha \xrightarrow{a} s_{m-1}\alpha$ then the defender responds by $s_m \xrightarrow{a} s_{m-1}$ (and vice-versa). The resulting pair is bisimilar by definition.
   If the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ or $p_m\alpha \xrightarrow{a} p_{m-1}\alpha \bar{X}_m$ then the defender responds by $s_m \xrightarrow{a} s_{m-1}$. In both cases the resulting pair is bisimilar by induction hypothesis, because both formulae $Q_{\alpha X_m}$ and $Q_{\alpha \bar{X}_m}$ are true.
   (b) If m is odd then $Q_\alpha = \exists x_m \forall x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$ is true. Since x_m is existentially quantified, there must be at least one right value for x_m, i.e., at least one of the formulae $Q_{\alpha X_m}$ or $Q_{\alpha \bar{X}_m}$ must be true.
   If the attacker moves $p_m\alpha \xrightarrow{a} t_{m-1}\alpha$ then the defender responds by $s_m \xrightarrow{a} t_{m-1}$ (and vice-versa). The resulting pair is bisimilar by definition.
   If the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ then the defender responds either by $s_m \xrightarrow{a} s_{m-1}$ or by $s_m \xrightarrow{a} t_{m-1}$, depending on whether $Q_{\alpha X_m}$ is true or not. The resulting pair is bisimilar by induction hypothesis.
   Similarly, if the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha \bar{X}_m$ then the defender responds either by $s_m \xrightarrow{a} s_{m-1}$ or by $s_m \xrightarrow{a} t_{m-1}$, depending on whether $Q_{\alpha \bar{X}_m}$ is true or not. The resulting pair is bisimilar by induction hypothesis.
   Finally, if the attacker moves $s_m \xrightarrow{a} s_{m-1}$ then the defender responds either by $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ or $p_m\alpha \xrightarrow{a} p_{m-1}\alpha \bar{X}_m$, depending on whether $Q_{\alpha X_m}$ or $Q_{\alpha \bar{X}_m}$ is true. There is at least one right possible choice, because at least one of these formulae must be true. The resulting pair is bisimilar by induction hypothesis.

   In every case the defender had a winning strategy and thus $s_m \sim p_m\alpha$.
3. Now we consider the case where Q_α is false. We handle odd and even values of m separately.
   (a) If m is even then $Q_\alpha = \forall x_m \exists x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$ is false. Since x_m is universally quantified, at least one of the two formulae $Q_{\alpha X_m}$ and $Q_{\alpha \bar{X}_m}$ must be false.
   If the attacker moves $p_m\alpha \xrightarrow{a} s_{m-1}\alpha$ then the defender responds by $t_m \xrightarrow{a} s_{m-1}$ (and vice-versa). The resulting pair is bisimilar by definition.
   If the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ then the defender responds either by $t_m \xrightarrow{a} s_{m-1}$ or by $t_m \xrightarrow{a} t_{m-1}$, depending on whether $Q_{\alpha X_m}$ is true or not. The resulting pair is bisimilar by induction hypothesis.

Similarly, if the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha\bar{X}_m$ then the defender responds either by $t_m \xrightarrow{a} s_{m-1}$ or by $t_m \xrightarrow{a} t_{m-1}$, depending on whether $Q_{\alpha\bar{X}_m}$ is true or not. The resulting pair is bisimilar by induction hypothesis. Finally, if the attacker moves $t_m \xrightarrow{a} t_{m-1}$ then the defender responds either by $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ or by $p_m\alpha \xrightarrow{a} p_{m-1}\alpha\bar{X}_m$, depending on whether $Q_{\alpha X_m}$ or $Q_{\alpha\bar{X}_m}$ is false. There is at least one possible such choice, because at least one of these formulae must be false. The resulting pair is bisimilar by induction hypothesis.

(b) If m is odd then $Q_\alpha = \exists x_m \forall x_{m-1} \ldots \exists x_1 (Q_1(\alpha) \wedge \cdots \wedge Q_k(\alpha))$ is false. This implies that the formula stays false whatever value is chosen for x_m, i.e., both formulae $Q_{\alpha X_m}$ and $Q_{\alpha\bar{X}_m}$ are false.
If the attacker moves $p_m\alpha \xrightarrow{a} t_{m-1}\alpha$ then the defender responds by $t_m \xrightarrow{a} t_{m-1}$ (and vice-versa). The resulting pair is bisimilar by definition.
If the attacker moves $p_m\alpha \xrightarrow{a} p_{m-1}\alpha X_m$ or $p_m\alpha \xrightarrow{a} p_{m-1}\alpha\bar{X}_m$ then the defender responds by $t_m \xrightarrow{a} t_{m-1}$. In both cases the resulting pair is bisimilar by induction hypothesis, because both formulae $Q_{\alpha X_m}$ and $Q_{\alpha\bar{X}_m}$ are false.

In every case the defender had a winning strategy and thus $t_m \sim p_m\alpha$.

The 'only-if' part of the properties (2) and (3) follows from the fact that $s_m \not\sim t_m$, which has been shown in (1). □

Theorem 16. *The problem* ***PDA*** $\sim$ ***FS*** *is* ***PSPACE****-hard.*

Proof. We instantiate Lemma 1 with $m = n$ and obtain that Q (i.e., Q_ε) is true iff $s_n \sim p_n\varepsilon$. Then the result follows from the **PSPACE**-hardness of QBF. □

6. Weak Bisimulation Equivalence

We consider the following problem.

PDA $\approx$ FS
Instance: A pushdown process P and a finite-state system F.
Question: $P \approx F$?

This problem is **PSPACE**-complete in general. Unlike strong bisimulation, it is fixed-parameter tractable in only one parameter. If the size of the finite control of P is fixed, then the problem is polynomial. However, it stays **PSPACE**-complete for a small fixed finite-state system F (with only 3 states).

The following table summarizes the complexity results for strong bisimulation. In the cases where a parameter size is fixed, the upper bounds are interpreted as 'for every fixed size', while the lower bounds are interpreted as 'for some fixed size'.

Complexity	General	fixed PDA control	fixed F (even size 3)	both fixed
PDA $\approx$ FS	**PSPACE**-compl.	**P**	**PSPACE**-compl.	**P**

6.1. Upper Bounds

Now we show that the problem **PDA** $\approx$ **FS** is in **PSPACE**. As shown in Subsection 2.2, the problem **PDA** $\approx$ **FS** can be reduced to a model checking problem for pushdown automata and the characteristic formula $\Theta_f^{\approx}$ in the temporal logic EF. The EF-formula $\Theta_f^{\approx}$ has polynomial size when described as a DAG and can be constructed in polynomial time in the size of **FS**.

It has been shown by Walukiewicz [57] that model checking pushdown automata with the temporal logic EF is **PSPACE**-complete. A closer analysis of the model checking algorithm presented in [57] reveals that its complexity remains in **PSPACE** for all EF-formulae which are described by a polynomial-size DAG [59]. Finally, our characteristic formula Θ_f uses a slight extension of EF, because of the $\Diamond_\tau$ operator (normal EF has only the $\Diamond$ operator). However, the model checking algorithm of [57] can trivially be generalized to this extension of EF without increasing its complexity.

Thus, model checking pushdown automata with the formula $\Theta_f^{\approx}$ can be decided in polynomial space. So the whole algorithm to check weak bisimilarity works in polynomial space and we obtain the following theorem.

Theorem 17. ***PDA*** $\approx$ ***FS*** *is decidable in polynomial space.*

Now we consider the problem of fixed-parameter tractability for given bounds on the size of the finite control of the PDA.

We showed in [35, 38] that the problem **BPA** $\approx$ **FS** is polynomial (i.e., for pushdown automata with a finite control of the fixed size 1). This result was generalized to pushdown automata with a finite control of an arbitrary fixed size k in [40].

Theorem 18. *([40]) If the size of the finite control of the PDA is bounded by a fixed constant k, then the problem* ***PDA*** $\approx$ ***FS*** *is decidable in polynomial time.*

6.2. Lower Bounds

The problem **PDA** $\approx$ **FS** is **PSPACE**-hard, even for a small fixed finite-state system with 3 states (shown in Figure 4). The proof is done by a reduction from the **PSPACE**-complete problem [56] if a single tape, linearly space-bounded, nondeterministic Turing-machine M accepts a given input w. There is a constant k s.t. if M accepts an input w then it has an accepting computation that uses only $k \cdot |w|$ space. For any such M and w we construct a pushdown automaton P s.t.

- If M accepts w then P is not weakly bisimilar to any finite-state system.
- If M doesn't accept w then P is weakly bisimilar to the finite-state system F of Figure 4.

The construction of P is as follows. Let $n := k \cdot |w| + 1$ and Σ be the set of tape symbols of M. Configurations of M are encoded as sequences of n symbols of the form $v_1 q v_2$ where $v_1, v_2 \in \Sigma^*$ are sequences of tape symbols of M and q is a state of the finite control of M. The sequence v_1 are the symbols to the left of the head and v_2 are

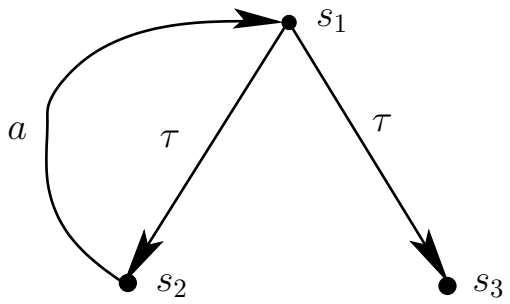

Figure 4: The finite-state system F with initial state s_1.

the symbols under the head and to the right of it. (v_1 can be empty, but v_2 can't.) Let p_0 be the initial control-state of P and let the stack be initially empty. Initially, P is in the phase 'guess' where it guesses an arbitrarily long sequence $c_1\#c_2\#\ldots\#c_m$ of configurations of M (each of these c_i has length n) and stores them on the stack. The pushdown automaton can guess a sequence of length n by n times guessing a symbol and storing it on the stack. The number of symbols guessed (from 1 to n) is counted in the finite control of the pushdown automaton. The number m is not counted in the finite control, since it can be arbitrarily large. The configuration c_m at the bottom of the stack must be accepting (i.e., the state q in c_m must be accepting) and the configuration c_1 at the top must be the initial configuration with the input w and the initial control-state of M. All this is done with silent τ-actions. At the end of this phase P is in the control state p. Then there are two possible transitions: (1) $p \xrightarrow{\tau} p_0A$ where the special symbol $A \notin \Sigma$ is written on the stack and the guessing phase starts again. (2) $p \xrightarrow{\tau} p_{verify}$ where the pushdown automaton enters the new phase 'verify'.

In the phase 'verify' the pushdown automaton P pops symbols from the stack (by action τ). At any time in this phase it can (but need not) enter the special phase 'check'. For a 'check' it reads three symbols from the stack. These symbols are part of some configuration c_i. Then it pops $n-2$ symbols and then reads the three symbols at the same position in the next configuration c_{i+1} (unless the bottom of the stack is reached already). In a correct computation step from c_i to c_{i+1} the second triple of symbols depends on the first and on the definition of M. If these symbols in the second triple are as they should be in a correct computation step of M from c_i to c_{i+1} then the 'check' is successful and it goes back into the phase 'verify'. Otherwise the 'check' has failed and P is in the control-state *fail*. Here there are two possible transitions: (1) $fail \xrightarrow{\tau} p_2$. In the control-state p_2 the stack is ignored and the pushdown automaton from then on behaves just like the state s_2 in the finite-state system F of Figure 4. (2) $fail \xrightarrow{\tau} p_3$. In the control-state p_3 again the stack is ignored and from then on the pushdown automaton behaves just like the state s_3 in the finite-state system F of Figure 4. The intuition is that if the sequence of configurations represents a correct computation of M then no 'check' can fail, i.e., the control-state *fail* cannot be reached. However, if the sequence isn't a correct computation then there must be at least one error somewhere and thus the control-state *fail* can be reached by doing the 'check' at the right place.

So far, all actions have been silent τ-actions. The only case where a visible action can occur is the following: The pushdown automaton P is in phase 'verify' or 'check' (but not in state *fail*) and reads the special symbol A from the stack. Then it does the visible action 'a' and goes to the control-state p_{verify}. If P reaches the bottom of the stack while being in phase 'verify' or 'check' then it is in a deadlock.

Lemma 2. *If M accepts the input w then P is not weakly bisimilar to any finite-state system.*

Proof. We assume the contrary and derive a contradiction. Assume that there is a finite-state system F' with r states s.t. $P \approx F'$. Since M accepts w, there exists an accepting computation sequence $c = c_1 \# c_2 \# \ldots \# c_m$ where all c_i are configurations of M, c_1 is the initial configuration of M with input w, c_m is accepting and for all $i \in \{1, \ldots, m-1\}$ $c_i \rightarrow c_{i+1}$ is a correct computation step of M.

P can (by a sequence of τ-steps) reach the configuration $\alpha := p_{verify}\,(cA)^{r+1}c$. Since c is an accepting computation sequence of M, none of the checks can fail. Thus the *only* sequence of actions that α can do is $\tau^{mn+m-1}(a\tau^{mn+m-1})^{r+1}$. (In particular, no infinite loop is possible.)

We assumed that $P \approx F'$. Thus there must be some state f of F' s.t. $\alpha \approx f$. Since F' has only r states, it follows from the Pumping Lemma for regular languages that $\alpha \not\approx f$ and we have a contradiction. □

Lemma 3. *Let F be the finite-state system from Figure 4. If M doesn't accept the input w then $P \approx F$.*

Proof. Since there is no accepting computation of M on w, any reachable configuration of P belongs to one of the following three sets.

1. Let C_1 be the set of configurations of P where either P is in phase 'guess' or P is in phase 'verify' or 'check' s.t. a check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* can be reached with only τ-actions.
2. Let C_2 be the set of configurations of P where either the finite control of P is in state p_2 or P is in phase 'verify' or 'check', there is at least one symbol A on the stack and no check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* cannot be reached with only τ-actions, but possibly after another 'a' action.
3. Let C_3 be the set of configurations of P where either the finite control of P is in state p_3 or P is in phase 'verify' or 'check', there is no symbol A on the stack and no check can fail, i.e. the control-state *fail* cannot be reached.

The following relation is a weak bisimulation:

$$\{(\alpha_1, s_1) \mid \alpha_1 \in C_1\} \cup \{(\alpha_2, s_2) \mid \alpha_2 \in C_2\} \cup \{(\alpha_3, s_3) \mid \alpha_3 \in C_3\}$$

We consider all possible attacks.

1. Note that no $\alpha_1 \in C_1$ can do action 'a'.

- If the attacker makes a move from a configuration in C_1 with control-state *fail* to p_2/p_3 then the defender responds by a move $s_1 \xrightarrow{\tau} s_1/s_2$. These are weakly bisimilar to p_2/p_3 by definition. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1, \alpha_1' \in C_1$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_2$ (this is only possible if there is at least one symbol A on the stack) then the defender responds by making a move $s_1 \xrightarrow{\tau} s_2$. If the attacker makes a move $\alpha_1 \xrightarrow{\tau} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_3$ (this is only possible if there is no symbol A on the stack) then the defender responds by making a move $s_1 \xrightarrow{\tau} s_3$.
- If the attacker makes a move $s_1 \xrightarrow{\tau} s_2/s_3$ then the defender makes a sequence of τ-moves where a 'check' fails and goes (via the control-state *fail*) to a configuration with control-state p_2/p_3. This is weakly bisimilar to s_2/s_3 by definition.

2. If α_2 is a configuration with control-state p_2 then this is bisimilar to s_2 by definition.
   - If the attacker makes a move $\alpha_2 \xrightarrow{\tau} \alpha_2'$ with $\alpha_2, \alpha_2' \in C_2$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_2 \xrightarrow{a} \alpha_2'$ (this is only possible if the symbol A is at the top of the stack) then the control-state of α_2' is p_{verify} and $\alpha_2' \in C_1$. Thus the defender can respond by $s_2 \xrightarrow{a} s_1$.
   - If the attacker makes a move $s_2 \xrightarrow{a} s_1$ then the defender responds as follows: First he makes a sequence of τ-moves $\alpha_2 \xrightarrow{\tau^*} \alpha_2'$ that pops symbols from the stack without doing any 'check' until the special symbol A is at the top. Then he makes a move $\alpha_2' \xrightarrow{a} \alpha_2''$. By definition the control-state of α_2'' is p_{verify} and $\alpha_2'' \in C_1$.
3. A configuration $\alpha_3 \in C_3$ can never reach a configuration where it can do action 'a'. The only possible action is τ. Thus $\alpha_3 \approx s_3$.

Since the initial configuration of P is in C_1 and the initial state of F is s_1, we get $P \approx F$. □

Theorem 19. *Checking weak bisimilarity of pushdown processes and finite-state systems is **PSPACE**-hard, even for the fixed finite-state system F of Figure 4.*

Proof. By reduction of the acceptance problem for single tape nondeterministic linear space-bounded Turing machines. Let M, w, P and F be defined as above. If M accepts w then by Lemma 2 P is not weakly bisimilar to any finite-state system and thus $P \not\approx F$. If M doesn't accept w then by Lemma 3 $P \approx F$. □

7. Conclusion

We have shown a complete picture of the computational complexity of checking (weak and strong) simulation preorder/equivalence and bisimulation equivalence between pushdown processes and finite-state systems. Furthermore, we have shown under which condition these problems are fixed-parameter tractable. The following table summarizes the complexity results.

Complexity	General	fixed PDA control (even size 1; BPA)	fixed F	both fixed
PDA $\sqsubseteq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\sqsupseteq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\simeq_w$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\sqsubseteq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\sqsupseteq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\simeq$ **FS**	**EXPTIME**-compl.	**EXPTIME**-compl.	**EXPTIME**-compl.	**P**
PDA $\approx$ **FS**	**PSPACE**-complete	**P**	**PSPACE**-complete	**P**
PDA $\sim$ **FS**	**PSPACE**-complete	**P**	**P**	**P**

We draw the following conclusions from these results.

- Simulation is computationally harder than bisimulation, both in general and for fixed-parameter tractability.

  The same trend also holds for the problems where one compares two infinite-state systems, e.g., **PDA** $\sqsubseteq$ **PDA**, **BPA** $\sqsubseteq$ **BPA**, **PDA** $\approx$ **PDA**, etc. For example, **BPA** $\sqsubseteq$ **BPA** (and all other simulation problems for **BPA**/**PDA**) are undecidable, **PDA** $\approx$ **PDA** is undecidable [50], **BPA** $\approx$ **BPA** is **EXPTIME**-hard [43] and **PDA** $\sim$ **PDA** is decidable and **EXPTIME**-hard [47, 36] and **BPA** $\sim$ **BPA** is decidable in 2-**EXPTIME** and **PSPACE**-hard [13, 49]. (See [48] for a general overview.)

  One reason for this trend is that bisimulation checking problems can be reduced in polynomial time to simulation checking problems for a large class of process models, which includes pushdown processes and finite-state systems [39].

- Fixed-parameter tractability is important.

  While many of these semantic preorder/equivalence checking problems have a high complexity, they are all fixed-parameter tractable. In some cases (e.g., strong bisimulation) it even suffices to fix just one parameter to make the problem polynomial.

References

[1] Proceedings of CONCUR'96, vol. 1119 of Lecture Notes in Computer Science, Springer, 1996 (1996).

[2] Proceedings of CONCUR'99, vol. 1664 of Lecture Notes in Computer Science, Springer, 1999 (1999).

[3] Proceedings of CAV 2001, vol. 2102 of Lecture Notes in Computer Science, Springer, 2001 (2001).

[4] Proceedings of CONCUR 2002, vol. 2421 of Lecture Notes in Computer Science, Springer, 2002 (2002).

[5] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: Proceedings of TACAS 2004, vol. 2988 of Lecture Notes in Computer Science, Springer, 2004.

[6] R. Alur, K. Etessami, M. Yannakakis, Analysis of recursive state machines, in: Proceedings of CAV 2001 [3], pp. 207–220.

[7] J. Baeten, J. Bergstra, J. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, Journal of the Association for Computing Machinery 40 (1993) 653–682.

[8] J. Baeten, W. Weijland, Process Algebra, No. 18 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

[9] A. Bouajjani, Languages, rewriting systems, and verification of infinite-state systems, in: Proceedings of ICALP 2001, vol. 2076 of Lecture Notes in Computer Science, Springer, 2001.

[10] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: application to model checking, in: Proceedings of CONCUR'97, vol. 1243 of Lecture Notes in Computer Science, Springer, 1997.

[11] O. Burkart, D. Caucal, F. Moller, B. Steffen, Verification on infinite structures, in: J. Bergstra, A. Ponse, S. Smolka (eds.), Handbook of Process Algebra, Elsevier, 2001.
[12] O. Burkart, D. Caucal, B. Steffen, An elementary decision procedure for arbitrary context-free processes, in: Proceedings of MFCS'95, vol. 969 of Lecture Notes in Computer Science, Springer, 1995.
[13] O. Burkart, D. Caucal, B. Steffen, Bisimulation collapse and the process taxonomy, in: Proceedings of CONCUR'96 [1], pp. 247–262.
[14] D. Caucal, Graphes canoniques des graphes algébriques, Informatique Théorique et Applications (RAIRO) 24 (4) (1990) 339–352.
[15] S. Christensen, H. Hüttel, C. Stirling, Bisimulation equivalence is decidable for all context-free processes, Information and Computation 121 (1995) 143–148.
[16] E. Clark, O. Grumberg, D. Peled, Model Checking, The MIT Press, 1999.
[17] J. Esparza, Decidability of model checking for infinite-state concurrent systems, Acta Informatica 34 (1997) 85–107.
[18] J. Esparza, J. Knoop, An automata-theoretic approach to interprocedural data-flow analysis, in: Proceedings of FoSSaCS'99, vol. 1578 of Lecture Notes in Computer Science, Springer, 1999.
[19] J. Esparza, A. Kučera, S. Schwoon, Model-checking LTL with regular valuations for pushdown systems, Information and Computation 186 (2) (2003) 355–376.
[20] J. Esparza, S. Schwoon, A BDD-based model checker for recursive programs, in: Proceedings of CAV 2001 [3], pp. 324–336.
[21] E. Friedman, The inclusion problem for simple languages, Theoretical Computer Science 1 (4) (1976) 297–316.
[22] J. Groote, A short proof of the decidability of bisimulation for normed BPA processes, Information Processing Letters 42 (1992) 167–171.
[23] M. Hennessy, R. Milner, On observing nondeterminism and concurrency, vol. 85 of LNCS, 1980.
[24] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial algorithm for deciding bisimilarity of normed context-free processes, Theoretical Computer Science 158 (1996) 143–159.
[25] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.
[26] H. Hüttel, C. Stirling, Actions speak louder than words: Proving bisimilarity for context-free processes, Journal of Logic and Computation 8 (4) (1998) 485–509.
[27] P. Jančar, Undecidability of bisimilarity for Petri nets and some related problems, Theoretical Computer Science 148 (2) (1995) 281–301.
[28] P. Jančar, A. Kučera, R. Mayr, Deciding bisimulation-like equivalences with finite-state processes, Theoretical Computer Science 258 (1–2) (2001) 409–433.
[29] P. Jančar, F. Moller, Techniques for decidability and undecidability of bisimilarity, in: Proceedings of CONCUR'99 [2], pp. 30–45.
[30] P. Jančar, A. Kučera, Equivalence checking on infinite-state systems: Techniques and results, Theory and Practice of Logic Programming 6 (3) (2006) 227–264.
[31] P. Jančar, J. Srba, Undecidability of bisimilarity by defender's forcing, Journal of the ACM 55 (1) (2008) 1–26.
[32] D. Kozen, Results on the propositional μ-calculus, Theoretical Computer Science 27 (1983) 333–354.
[33] A. Kučera, On simulation-checking with sequential systems, in: Proceedings of ASIAN 2000, vol. 1961 of Lecture Notes in Computer Science, Springer, 2000.
[34] A. Kučera, R. Mayr, Simulation preorder on simple process algebras, in: Proceedings of ICALP'99, vol. 1644 of Lecture Notes in Computer Science, Springer, 1999.
[35] A. Kučera, R. Mayr, Weak bisimilarity with infinite-state systems can be decided in polynomial time, in: Proceedings of CONCUR'99 [2], pp. 368–382.
[36] A. Kučera, R. Mayr, On the complexity of semantic equivalences for pushdown automata and BPA, in: Proceedings of MFCS 2002, vol. 2420 of Lecture Notes in Computer Science, Springer, 2002.
[37] A. Kučera, R. Mayr, Simulation preorder over simple process algebras, Information and Computation 173 (2) (2002) 184–198.
[38] A. Kučera, R. Mayr, Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time, Theoretical Computer Science 270 (1–2) (2002) 677–700.
[39] A. Kučera, R. Mayr, Why is simulation harder than bisimulation?, in: Proceedings of CONCUR 2002 [4], pp. 594–609.
[40] A. Kučera, R. Mayr, A generic framework for checking semantic equivalences between pushdown automata and finite-state automata, in: Proceedings of IFIP TCS 2004, 2004.
[41] R. Mayr, On the complexity of bisimulation problems for pushdown automata, in: Proceedings of

IFIP TCS'2000, vol. 1872 of Lecture Notes in Computer Science, Springer, 2000.
[42] R. Mayr, Undecidability of weak bisimulation equivalence for 1-counter processes, in: Proceedings of ICALP 2003, vol. 2719 of Lecture Notes in Computer Science, Springer, 2003.
[43] R. Mayr, Weak bisimilarity and regularity of context-free processes is exptime-hard, TCS 330 (2005) 553–575.
[44] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[45] F. Moller, Infinite results, in: Proceedings of CONCUR'96 [1], pp. 195–216.
[46] D. Park, Concurrency and automata on infinite sequences, in: Proceedings 5^{th} GI Conference, vol. 104 of Lecture Notes in Computer Science, Springer, 1981.
[47] G. Sénizergues, Decidability of bisimulation equivalence for equational graphs of finite out-degree, in: Proceedings of FOCS'98, IEEE Computer Society Press, 1998.
[48] J. Srba, Roadmap of infinite results, EATCS Bulletin (78) (2002) 163–175.
[49] J. Srba, Strong bisimilarity and regularity of basic process algebra is PSPACE-hard, in: Proceedings of ICALP 2002, vol. 2380 of Lecture Notes in Computer Science, Springer, 2002.
[50] J. Srba, Undecidability of weak bisimilarity for pushdown processes, in: Proceedings of CONCUR 2002 [4], pp. 579–593.
[51] C. Stirling, Decidability of bisimulation equivalence for normed pushdown processes, Theoretical Computer Science 195 (1998) 113–131.
[52] C. Stirling, The joys of bisimulation, in: Proceedings of MFCS'98, vol. 1450 of Lecture Notes in Computer Science, Springer, 1998.
[53] W. Thomas, On the Ehrenfeucht-Fraïssé game in theoretical computer science, in: Proceedings of TAPSOFT'93, vol. 668 of Lecture Notes in Computer Science, Springer, 1993.
[54] R. van Glabbeek, The linear time—branching time spectrum II: The semantics of sequential systems with silent moves, in: Proceedings of CONCUR'93, vol. 715 of Lecture Notes in Computer Science, Springer, 1993.
[55] R. van Glabbeek, The linear time—branching time spectrum, Handbook of Process Algebra (1999) 3–99.
[56] J. van Leeuwen (ed.), Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity, Elsevier, 1990.
[57] I. Walukiewicz, Model checking CTL properties of pushdown systems, in: Proceedings of FST&TCS 2000, vol. 1974 of Lecture Notes in Computer Science, Springer, 2000.
[58] I. Walukiewicz, Pushdown processes: Games and model-checking, Information and Computation 164 (2) (2001) 234–263.
[59] I. Walukiewicz, personal communication (2006).

Appendix

Strong bisimulation equivalence between two pushdown automata is decidable [47], but no upper complexity bound is known. The best known lower bound is **EXPTIME**-hardness, shown here.

Theorem 20. *The problem* ***PDA*** $\sim$ ***PDA*** *is* ***EXPTIME****-hard.*

Proof. We reduce the **EXPTIME**-complete [56] acceptance problem for alternating linear-bounded automata (LBA) to the **PDA** $\sim$ **PDA** problem. The intuition for the construction is as follows.

The attacker in the bisimulation game has a winning strategy if and only if the alternating LBA accepts. The bisimulation game encodes the computation of the alternating LBA. The sequence of traversed LBA configurations is stored on the stack of both pushdown automata. If the current LBA control-state is existential, then the attacker gets to choose the successor control-state. If the current LBA control-state is universal, then the defender gets to choose the successor control-state. The attacker chooses the tape symbols of the next LBA configuration. There is no immediate guarantee that these are the right tape contents w.r.t the LBA computation (i.e., the attacker could cheat here). However, after every step the defender gets the option of challenging the attackers choice of tape symbols (i.e., to verify if they are incompatible with the rules of the LBA computation). If such a challenge reveals an error (i.e., cheating) of the attacker, then the defender wins the game. This forces the attacker to play 'honest' and thus to faithfully simulate the LBA computation. If this faithful simulation of the alternating LBA finally reaches the accepting state of the LBA, then a special action becomes enabled in only one of the two pushdown automata and thus the attacker wins the bisimulation game. Otherwise, the defender wins the game.

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 $S'' := \{q'' \mid q \in S\}$. $Q := S \times \{0, \ldots, n-1\} \cup (S \times \Sigma) \times \{1, \ldots, n\} \cup S' \times \{0, \ldots, n-1\} \cup (S' \times \Sigma) \times \{1, \ldots, n\} \cup S'' \times \{0, \ldots, n-1\} \cup (S'' \times \Sigma) \times \{1, \ldots, n\} \cup \{(\tilde{q}, 0) \mid q \in S\} \cup (S \times S) \times \{0\} \cup \{q_c, q_c''\} \cup \{x\}$.

The reason that the control-states of the PDA can have complex forms like $(S \times \Sigma) \times \{1, \ldots, n\}$ is that it may be necessary to remember in the control-state of the PDA which tape symbol $\in \Sigma$ is stored at the position of the read/write head of the simulated LBA. Furthermore, one also needs to know the current tape position $\in \{1, \ldots, n\}$ of the read/write head of the simulated LBA.

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. $(q, i) \xrightarrow{X} (q, i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \leq i \leq n-2$;
2. $(q, i) \xrightarrow{(q,Y)} ((q, Y), i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \leq i \leq n-1$;

3. $((q,Y),i) \xrightarrow{X} ((q,Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
4. $((q,Y),n) \xrightarrow{q_1} (q_1,0)$ if $\pi(q) = \exists$ and $q_1 = \mathit{first}(q,Y)/\mathit{second}(q,Y)$;
5. $(q',i) \xrightarrow{X} (q',i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
6. $(q',i) \xrightarrow{(q,Y)} ((q',Y),i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \le i \le n-1$;
7. $((q',Y),i) \xrightarrow{X} ((q',Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
8. $((q',Y),n) \xrightarrow{q_1} (q_1',0)$ if $\pi(q) = \exists$ and $q_1 = \mathit{first}(q,Y)/\mathit{second}(q,Y)$;
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 $\pi(q) = \mathit{acc}$;
21. $(q,i) \xrightarrow{X} (q'',i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
22. $(q,i) \xrightarrow{(q,Y)} ((q'',Y),i+1)Yq$ for all $q \in S$, $0 \le i \le n-1$;
23. $((q,Y),i) \xrightarrow{X} ((q'',Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
24. $(q',i) \xrightarrow{X} (q'',i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
25. $(q',i) \xrightarrow{(q,Y)} ((q'',Y),i+1)Yq$ for all $q \in S$, $0 \le i \le n-1$;
26. $((q',Y),i) \xrightarrow{X} ((q'',Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
27. $(q'',i) \xrightarrow{X} (q,i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
28. $(q'',i) \xrightarrow{(q,Y)} ((q,Y),i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \le i \le n-1$;
29. $((q'',Y),i) \xrightarrow{X} ((q,Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
30. $((q'',Y),n) \xrightarrow{q_1} (q_1,0)$ if $\pi(q) = \exists$ and $q_1 = \mathit{first}(q,Y)/\mathit{second}(q,Y)$;
31. $(q'',i) \xrightarrow{X} (q'',i+1)X$ for all $q \in S$, $X \in \Sigma$, $0 \le i \le n-2$;
32. $(q'',i) \xrightarrow{(q,Y)} ((q'',Y),i+1)Yq$ for all $q \in S$, $Y \in \Sigma$, $0 \le i \le n-1$;
33. $((q'',Y),i) \xrightarrow{X} ((q'',Y),i+1)X$ for all $q \in S$, $X,Y \in \Sigma$, $0 \le i \le n-1$;
34. $((q'',Y),n) \xrightarrow{q_1} (q_1'',0)$ if $\pi(q) = \exists$ and $q_1 = \mathit{first}(q,Y)/\mathit{second}(q,Y)$;
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 $q \in S$, $0 \le i \le n-1$;
39. $((q,Y),i) \xrightarrow{c} q_c$ for all $q \in S$, $0 \le i \le n$;

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'.

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 (rules 4 and 8).

However, the defender determines the successor-control-state in those cases where the control-state is labeled as universal (rules 9–19). The construction in the rules 9–19 is a classic application of the so-called 'defender's choice' technique in bisimulation games. The idea is that by threatening to make the two processes equal (and thus winning the game immediately), the defender can force the attacker to choose a particular action according to the defender's wishes. (This technique was pioneered in [27], but not made very explicit there. See [30, 31] for a detailed modern description of this technique). In our construction the game proceeds as follows. If the current control-state q is universal (i.e., $\pi(q) = \forall$) then the rules (9–11) apply to the q-side and the rules (12–13) apply to the q' side. However, the attacker must choose rule 11, because otherwise the defender can make the two processes equal in the next step. Then the defender has a choice between rule 12 and rule 13, and thus chooses between the first successor q_1 and the second successor q_2 of q. If the defender chose q_1 (i.e., rule 12) then the attacker is forced to chose rule 16, because otherwise the processes become equal (by rules 15/17). Then the defender responds by rule 14 and the game continues from the control-states q_1/q'_1 in the left/right process. If the defender chose q_2 (i.e., rule 13) then the attacker is forced to chose rule 19, because otherwise the processes become equal (by rules 14/18). Then the defender responds by rule 15 and the game continues from the control-states q_2/q'_2 in the left/right process. Thus it was entirely up to the defender whether the game

continues from q_1/q_1' or q_2/q_2', i.e., whether the first or second successor control-state of q was chosen.

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 (see detailed description above). 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 (see detailed description above). 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 $\alpha \sim \alpha'$. □