On the Complexity of Bisimulation Problems for Pushdown Automata

Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France. E-mail: mayr@liafa.jussieu.fr
Phone: +33 1 44 27 28 40, Fax: +33 1 44 27 68 49

Abstract. All bisimulation problems for pushdown automata are at least *PSPACE*-hard. In particular, we show that (1) Weak bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, even for a small fixed finite automaton, (2) Strong bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, but polynomial for every fixed finite automaton, (3) Regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity is *PSPACE*-hard.

Keywords: Pushdown automata, bisimulation, verification, complexity

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [21]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [22] for a survey). While many algorithms for bisimulation problems have a very high complexity, only few lower bounds are known. Jančar [12, 13] showed that strong bisimilarity of two Petri nets [25] and weak bisimilarity of a Petri net and a finite automaton is undecidable. Stříbrná [28] showed that weak bisimilarity for Basic Parallel Processes (BPP) is $\mathcal{NP}$-hard and weak bisimilarity for context-free processes (BPA) is *PSPACE*-hard. (BPA are a proper subclass of pushdown automata.) However, it is still an open question whether these two problems are decidable. So far, the only known lower bound for a decidable bisimulation problem was an *EXPSPACE*-lower bound for strong bisimilarity of Petri nets and finite automata [15], that follows from the hardness of the Petri net reachability problem [18].

For bisimulation problems where one compares an infinite-state system with a finite-state one, much more is known about the decidability and complexity than in the general case of two infinite-state systems [14]. Also the complexity can be much lower. In particular, weak (and strong) bisimilarity of a BPA-process and a finite automaton is decidable in polynomial time [17], while weak bisimilarity of two BPA-processes is *PSPACE*-hard [28].

However, this surprising result does not carry over to general pushdown automata. We show that strong and weak bisimilarity of a pushdown automaton

and a finite automaton is *PSPACE*-hard. (These problems were already known to be in *EXPTIME* [14].) For weak bisimilarity this hardness result holds even for a small fixed finite automaton, while the same problem for strong bisimilarity is polynomial in the size of the pushdown automaton for every fixed finite automaton. These results also yield a *PSPACE* lower bound for strong bisimilarity of two pushdown automata, a problem that has recently been shown to be decidable by Sénizergues [27] (the proof in [27] uses a combination of two semidecision procedures and does not yield any complexity measure).

The problem of bisimilarity is also related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata [26]. See Section 5 for details.

Furthermore, we prove a *PSPACE* lower bound for the problem of regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity.

Thus no bisimulation problem for pushdown automata is polynomial (unless *PSPACE* is $\mathcal{P}$). This shows that there is a great difference between pushdown automata and BPA, although they describe exactly the same class of languages (Chomsky-2).

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E ::= \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '.' is a sequential composition and '$\|$' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '$\|$' are associative, '$\|$' is commutative, and 'ϵ' is a unit for '.' and '$\|$'.

A *process rewrite system* (PRS) [20] is specified by a finite set Δ of *rules* which have the form $E \stackrel{a}{\rightarrow} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\rightarrow} F) \in \Delta}{E \stackrel{a}{\rightarrow} F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E.F \stackrel{a}{\rightarrow} E'.F} \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E \| F \stackrel{a}{\rightarrow} E' \| F}$$

We extend the notation $E \stackrel{a}{\rightarrow} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \stackrel{w}{\rightarrow} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. We also use '1' to denote the set of process constants.

The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA (Basic Process Algebra, also called context-free processes), BPP (Basic Parallel Processes), and PA-processes are well-known [1], PDA correspond to pushdown automata (as proved by Caucal in [6]), PN correspond to Petri nets, PRS stands for 'Process Rewrite Systems', PAD and PAN are artificial names made by combining existing ones (PAD = PA+PDA, PAN = PA+PN).

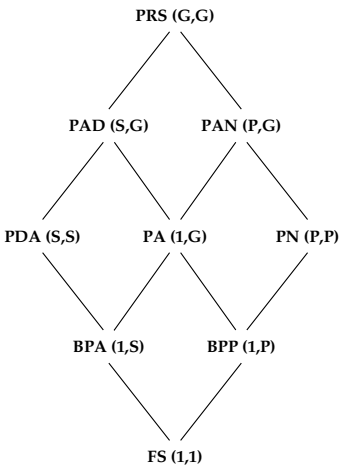

Fig. 1. A hierarchy of PRS

We consider the semantical equivalences *weak bisimilarity* and *strong bisimilarity* [21] over transition systems generated by PRS. In what follows we consider process expressions over $Const(\Delta)$ where Δ is some fixed process rewrite system.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over process expressions is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them.*

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

Note that context-free processes (BPA) correspond to the subclass of pushdown automata (PDA) where the finite control has size 1. Although BPA and PDA describe the same class of languages (Chomsky-2), BPA is strictly less expressive w.r.t. bisimulation.

3 Hardness of Weak Bisimulation Problems

In this section we show lower bounds for problems about weak bisimulation. We consider the following two problems:

Weak bisimilarity of pushdown automata and finite automata

Instance: A pushdown automaton P and a finite automaton F.
Question: $P \approx F$?

Weak Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \approx F$?

We show that both these problems are *PSPACE*-hard. The proof is done by a reduction from the *PSPACE*-complete problem 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 automaton.
- If M doesn't accept w then P is weakly bisimilar to the finite automaton F of Figure 2.

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 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.

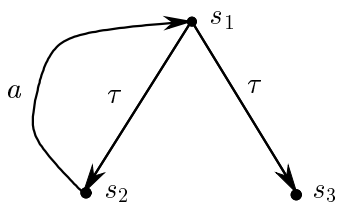

Fig. 2. The finite automaton F with initial state s_1.

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) $p \xrightarrow{\tau} p_0 A$ 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) $\mathit{fail} \xrightarrow{\tau} p_2$. 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 automaton F of Figure 2. (2) $\mathit{fail} \xrightarrow{\tau} p_3$. 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 automaton F of Figure 2. 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 automaton.*

Proof. We assume the contrary and derive a contradiction. Assume that there is finite automaton F' with k 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 $i \in \{1, \ldots, m-1\}$ $c_i \to c_{i+1}$ is a correct computation step of M.
P can (by a sequence of τ-steps) reach the configuration $\alpha := p_{verify}\,(cA)^{k+1}c$. Since c is an accepting computation sequence of M, none of the checks can fail. Thus α can only do the following sequence of actions: $\tau^{mn+m-1}(a\tau^{mn+m-1})^{k+1}$. We assumed that $P \approx F'$. Thus there must be some state f of F' s.t. $\alpha \approx f$. Since F' has only k 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 automaton from Figure 2. 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 $s_1 \stackrel{\tau}{\rightarrow} s_1/s_2$. These are weakly bisimilar to p_2/p_3 by definition. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\rightarrow} \alpha_1'$ with $\alpha_1, \alpha_1' \in C_1$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\rightarrow} \alpha_1'$ 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 $s_1 \stackrel{\tau}{\rightarrow} s_2$. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\rightarrow} \alpha_1'$ 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 $s_1 \stackrel{\tau}{\rightarrow} s_3$.
   - If the attacker makes a move $s_1 \stackrel{\tau}{\rightarrow} s_2/s_3$ 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 $\alpha_2 \stackrel{\tau}{\rightarrow} \alpha_2'$ with $\alpha_2, \alpha_2' \in C_2$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_2 \stackrel{a}{\rightarrow} \alpha_2'$ (this is only possible if the symbol A is at the top of the stack) then the control-state of α_2' is q_{verify} and $\alpha_2' \in C_1$. Thus the defender can respond by $s_2 \stackrel{a}{\rightarrow} s_1$.
   - If the attacker makes a move $s_2 \stackrel{a}{\rightarrow} s_1$ then the defender responds as follows: First he makes a sequence of τ-moves $\alpha_2 \stackrel{\tau^*}{\rightarrow} \alpha_2'$ that pops symbols from the stack without doing any 'check' until the special symbol A is at the top. Then he makes a move $\alpha_2' \stackrel{a}{\rightarrow} \alpha_2''$. By definition the control-state of α_2'' is q_{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 4. *Weak bisimilarity of pushdown automata and finite automata is PSPACE-hard, even for the fixed finite automaton F of Figure 2.*

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 automaton and thus $P \not\approx F$. If M doesn't accept w then by Lemma 3 $P \approx F$. □

Theorem 5. *Weak finiteness of pushdown automata is PSPACE-hard.*

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 automaton and thus not weakly finite. If M doesn't accept w then by Lemma 3 $P \approx F$ and thus P is weakly finite. □

4 Hardness of Strong Bisimulation Problems

STRONG BISIMILARITY OF PUSHDOWN AUTOMATA AND FINITE AUTOMATA
Instance: A pushdown automaton P and a finite automaton F.
Question: $P \sim F$?

We show that this problem is *PSPACE*-hard in general, but polynomial in the size of P for every fixed finite automaton F. The *PSPACE* lower bound is shown by a reduction of the *PSPACE*-complete problem of quantified boolean formulae (QBF). Let $n \in \mathbb{N}$ and let $x_1, \ldots, x_n$ be boolean variables. W.r. 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. The quantified boolean formula Q is given by

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

where the Q_i are clauses. The problem is if Q is valid. We reduce this problem to the bisimulation problem by constructing a pushdown automaton P and a finite automaton F s.t. Q is valid iff $P \sim F$.

F is defined as follows: The initial state is s_0.

$$
\begin{array}{lll}
s_{2i} \xrightarrow{x_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
s_n \xrightarrow{a} u & \\
u \xrightarrow{c} u & \\
t_n \xrightarrow{a} u & \\
t_n \xrightarrow{a} w_n & \\
w_i \xrightarrow{c} w_{i-1} & \text{for } 1 \leq i \leq n
\end{array}
$$

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

Now we define the pushdown automaton P. Initially the stack is empty and the initial control-state is p_0. 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:

$$
\begin{array}{lll}
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)} X_{2i+2} X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)} \bar{X}_{2i+2} X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)} X_{2i+2} \bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)} \bar{X}_{2i+2} \bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_n \xrightarrow{a} q_j & \text{for } 0 \leq j \leq k \\
q_0 \xrightarrow{c} q_0 & \\
q_j X_l \xrightarrow{c} q_j X_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(X_l). \\
q_j X_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(X_l). \\
q_j \bar{X}_l \xrightarrow{c} q_j \bar{X}_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(\bar{X}_l). \\
q_j \bar{X}_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(\bar{X}_l).
\end{array}
$$

Additionally we define for $1 \leq i \leq n/2-1$ that in the control-state r_{2i} the stack is ignored and the systems behaves just like t_{2i} in the system F of Figure 3.

Lemma 6. *If Q is not valid then $P \not\sim F$.*

Proof. If Q is not valid then $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n (\neg Q_1 \vee \ldots \vee \neg Q_k)$ and the attacker has the following winning strategy: The attacker chooses the values for the variables with the odd indices by doing actions x_i or $\bar{x}_i$ in the finite automaton F and goes from s_0 to s_n. The defender can respond in two different ways: (1) If the defender goes into a control-state r_{2i} for some i then the attacker can easily

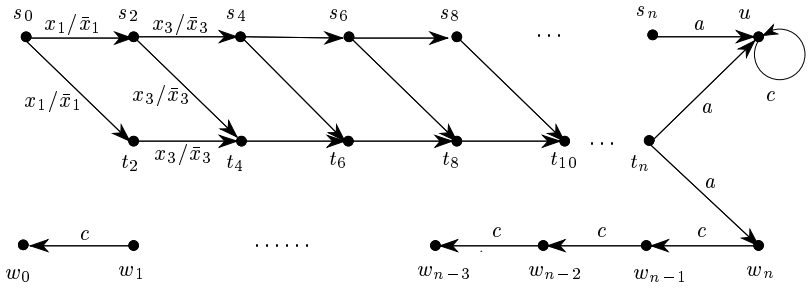

Fig. 3. Reducing QBF to strong bisimulation.

win, since r_{2i} behaves like t_{2i} and $s_{2i} \not\sim t_{2i}$ for every i. (2) If the defender stays in the 'p-domain' of control-states, he is forced to store the attacker's choices for the variables with odd indices on the stack. However, he can make his own choices for the variables with even indices and also stores them on the stack. Finally, the defender reaches the control-state p_n and the stack contains an assignment of values to all n variables. Since Q is not valid, there exists at least one Q_j with $1 \leq j \leq k$ that is not satisfied by this assignment. Now the attacker changes sides and makes the move $p_n \stackrel{a}{\rightarrow} q_j$ in the pushdown automaton P. The defender can only respond by making the move $s_n \stackrel{a}{\rightarrow} u$ in the system F. Now the pushdown automaton P can do the action 'c' only n times, while system F in state u can do it infinitely often. Thus the attacker can win. It follows that $P \not\sim F$. □

Lemma 7. *If Q is valid then $P \sim F$.*

Proof. Let C be a content of the stack and thus a (possibly incomplete) assignment of values to variables. Let $Q_i(C)$ be true iff C makes clause Q_i true. Let $Q(C) := \bigwedge_{1 \leq i \leq k} Q_i(C)$. Let $QX(C)$ be true iff C can be completed to a C' s.t. $Q(C')$. If Q is valid then the following relation is a strong bisimulation.

$$\{(p_{2i}C, s_{2i}) \,|\, 0 \leq i \leq n/2 \wedge QX(C)\} \cup \{(p_{2i}C, t_{2i}) \,|\, 1 \leq i \leq n/2 \wedge \neg QX(C)\} \cup$$
$$\{(r_{2i}C, t_{2i}) \,|\, 1 \leq i \leq n/2\} \cup \{(q_jC, u) \,|\, 1 \leq j \leq k \wedge Q_j(C)\} \cup \{(q_0C, u)\} \cup$$
$$\{(q_jC, w_i) \,|\, 1 \leq j \leq k \wedge 0 \leq i \leq n \wedge \neg Q_j(C) \wedge \mathit{length}(C) = i\}$$

Since $(p_0\epsilon, s_0)$ is in this relation, we get $P \sim F$. □

Theorem 8. *Strong bisimilarity of pushdown automata and finite automata is PSPACE-hard.*

Proof. Directly from Lemma 6 and Lemma 7. □

Corollary 9. *Strong bisimilarity of pushdown automata is PSPACE-hard.*

Note that Theorem 4 is not a corollary of Theorem 8. For weak bisimilarity the hardness result holds even for the small fixed finite automaton of Figure 2. However, strong bisimilarity of a pushdown automaton P and a finite automaton F is polynomial in the size of P for *every* fixed F.

Theorem 10. *Let F be a fixed finite automaton. For every pushdown automaton P the problem if $P \sim F$ requires only polynomial time in the size of P.*

Proof. Using the construction from [14] one can reduce the problem $P \sim F$ to a model checking problem in the temporal logic EF (a fragment of CTL). One can effectively construct Hennessy-Milner Logic formulae Φ and Ψ that depend only on F s.t.

$$P \sim F \iff (P \models \Phi) \wedge (P \models \neg EF\,\Psi)$$

where the modal operator EF denotes reachability. Let n be the size of (the description of) P and m the maximum of the nesting-depth of Φ and Ψ. (The total size of Φ and Ψ can be $\mathcal{O}(2^m)$.) Let P' be a state that is reachable from P. It depends only on the control state of P and P' and on the first m stack symbols of P and P' if they satisfy Φ and Ψ, respectively. There are only n different possibilities for the control state and n^m different possibilities for the first m stack symbols. For each of these n^{m+1} configurations we check if it satisfies Φ or Ψ. Each of those checks can be done in $\mathcal{O}(n^m)$ time. Also for each α of these n^{m+1} configurations we check if P can reach a configuration $\alpha\beta$ for some β. (β represents the stack contents below the first m stack symbols. It does not matter for Φ and Ψ.) Each of those (generalized) reachability-checks can be done in $\mathcal{O}(n^3m^2)$ time [3]. Therefore the whole property above can be checked in $\mathcal{O}(n^{2m+1}m^2)$ time. Thus the problem is polynomial in n, the size of P, but exponential in m. (To be precise, m depends only on F and can be made linear in the number of states in F [14].) □

Now we consider the strong finiteness problem.

Strong Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \sim F$?

We show that this problem is *PSPACE*-hard by a reduction of QBF. Let Q, P and F be defined just as before in the hardness proof of strong bisimilarity. As shown before, Q is valid iff $P \sim F$. We now construct a pushdown automaton P' s.t. P' is finite w.r.t. strong bisimilarity iff $P \sim F$. The initial configuration of P' is $p'Z$. The transition rules are

$$\begin{aligned}
p' &\xrightarrow{a'} p'C \\
p' &\xrightarrow{a'} q' \\
q'C &\xrightarrow{b'} q' \\
q'C &\xrightarrow{c'} p_0 \\
q'Z &\xrightarrow{b'} q'Z \\
q'Z &\xrightarrow{c'} s_0
\end{aligned}$$

Note that if P' is in control-state p_0 or s_0 then it behaves like P and F, respectively.

Lemma 11. *If $P \not\sim F$ then P' is infinite w.r.t. strong bisimilarity.*

Proof. There are infinitely many non-bisimilar reachable states $q'C^iZ$ for all $i \in \mathbb{N}$. It suffices to show that $q'C^iZ \not\sim q'C^jZ$ for $i > j$. The attacker has the following winning strategy: He does action b' exactly j times (the defender can respond in only one way) and the new state in the bisimulation game is $(q'C^{i-j}Z, q'Z)$. Then the attacker does action c' and after the defender's response the new state is $(p_0C^{i-j-1}Z, s_0)$. Since $P \not\sim F$, the attacker can win. □

Lemma 12. *If $P \sim F$ then P' is finite w.r.t. strong bisimilarity.*

Proof. Let the finite automaton F' with initial state s' be defined by

$$s' \xrightarrow{a'} s'$$
$$s' \xrightarrow{a'} t'$$
$$t' \xrightarrow{b'} t'$$
$$t' \xrightarrow{c'} s_0$$

where s_0 is the initial state of F. If $P \sim F$ then $p'C^iZ \sim s'$, $q'C^jZ \sim t'$, $p_0C^kZ \sim s_0$ and $s_0 \sim s_0$ and thus $P' \sim F'$. □

Theorem 13. *Strong finiteness of pushdown automata is PSPACE-hard.*

Proof. It follows from Lemmas 6,7, 11 and 12 that Q is satisfiable iff $P \sim F$ iff P' is finite w.r.t. strong bisimilarity. □

It might seem that Theorem 5 is a corollary of Theorem 13. However, a careful inspection reveals a slight difference. The proof of Theorem 5 shows that the question if, given a pushdown automaton P, "Is P weakly bisimilar to any finite automaton with at most 3 states ?" is *PSPACE*-hard. The same question for strong bisimilarity is polynomial, because of Theorem 10. (These results still hold if the number 3 in the question above is replaced by any other integer $k \geq 3$. For weak bisimilarity the question is *PSPACE*-hard in the size of P. For strong bisimilarity it is polynomial in the size of P and exponential in k.) So, while in general the finiteness problem for a pushdown automaton P is *PSPACE*-hard for both weak and strong bisimilarity, the modified question "Is P finite *and small* ?" is *PSPACE*-hard for weak bisimilarity, but polynomial for strong bisimilarity. To conclude, finiteness w.r.t. weak bisimilarity is hard in a slightly stronger sense.

5 Conclusion

We have shown that all bisimulation problems for pushdown automata are at least *PSPACE*-hard. Thus no bisimulation problem for pushdown automata is polynomial (unless $PSPACE = \mathcal{P}$). It is interesting to compare these results with the results for context-free processes (BPA), which describe exactly the

same class of languages (Chomsky-2). Strong and weak bisimilarity of BPA and finite automata can be decided in polynomial time [17]. This shows that there is a significant difference between pushdown automata and context-free processes (BPA) as far as 'branching-time equivalences' like strong and weak bisimulation are concerned. Intuitively, the reason for this is that, due to their finite control, pushdown automata have a limited power of self-test that context-free processes lack.

The problem of bisimulation equivalence is related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata (dPDA), which has been shown to be decidable in [26]. However, the relationship is more complex than it seems, because of the presence of ϵ-transitions in PDAs. 'Real-time' PDAs are PDAs without ϵ-transitions. We denote them by rPDA. We denote real-time deterministic PDAs as rdPDA. We can distinguish five problems.

1. For rdPDA, strong bisimilarity and trace-language equivalence coincide. (The problem of trace-language equivalence can easily be reduced to terminal-language equivalence on rdPDA.) This problem is also equivalent to strong bisimilarity of dPDA, because the ϵ-transitions don't matter for strong bisimilarity. Language equivalence on rdPDA has been shown to be decidable in [23]. Neither an upper complexity bound nor a lower complexity bound is known.
2. Strong bisimilarity for PDA and rPDA. These problems are equivalent, because the ϵ-transitions don't matter for strong bisimilarity. Decidability of strong bisimilarity for PDA has been shown in [27]. No upper complexity bound is known. Theorem 8 gives a *PSPACE* lower bound.
3. Language equivalence of dPDA. This is equivalent to weak bisimilarity of dPDA, if one renames the ϵ-transitions to τ-transitions. The problem is decidable by [26]. Neither an upper complexity bound nor a lower complexity bound is known.
4. Weak bisimilarity for PDA. It is an open question if this problem is decidable. A *PSPACE* lower bound has been shown in [28] (even for BPA). Theorem 4 shows that even the asymmetric problem of weak bisimilarity of a PDA and a (small fixed) finite automaton is *PSPACE*-hard.
5. Language equivalence for PDA and rPDA. These problems are inter-reducible and undecidable by [11].

Figure 4 shows the relationships between these five problems. The hardness results of this paper hold only for bisimilarity of nondeterministic PDA (i.e., problems number 2 and 4) and thus they don't yield a lower bound for the problem of language equivalence of dPDA (problem number 3). In particular, it is easy to see that language equivalence of a dPDA and a deterministic finite automaton is polynomial (unlike bisimilarity for nondeterministic systems; see Theorem 8). It still cannot be ruled out that a polynomial algorithm for language equivalence of dPDA might exist.

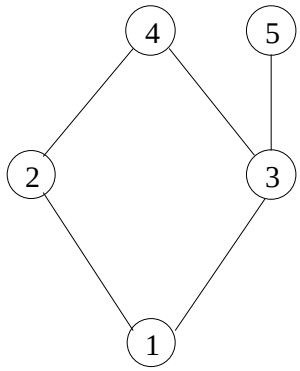

Fig. 4. Bisimulation vs. languages

Two lower bounds for bisimulation problems about Petri nets have not been mentioned explicitly in the literature so far. They concern the problems of strong bisimilarity of a Petri net and a finite automaton and finiteness of a Petri net w.r.t. strong bisimulation. It can easily be shown that these problems are *EXPSPACE*-hard by a reduction of the problem if a given place in a Petri net can ever become marked. (This problem is polynomially equivalent to the reachability problem for Petri nets [25] and thus *EXPSPACE*-hard [18].)

The following table summarizes known results about the complexity of bisimulation problems for several classes of infinite-state systems. The different columns show the results about the following problems: strong bisimilarity with finite automata, strong bisimilarity of two infinite-state systems, weak bisimilarity with finite automata and weak bisimilarity of two infinite-state systems. New results are in boldface.

	$\sim F$	$\sim$	$\approx F$	$\approx$
FS	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]	$\mathcal{P}$ [2, 24]
BPA	$\mathcal{P}$ [17]	$\in 2\text{-}EXPTIME$ [4]	$\mathcal{P}$ [17]	*PSPACE*-hard [28]
PDA	$\in EXPTIME$ [14] **PSPACE-hard**	decidable [27] **PSPACE-hard**	$\in EXPTIME$ [14] **PSPACE-hard**	*PSPACE*-hard [28]
BPP	$\in PSPACE$ [14]	decidable [7] co-$\mathcal{NP}$-hard [19]	$\in PSPACE$ [14]	$\mathcal{NP}$-hard [28] Π_2^p-hard [19]
PA	decidable [14]	co-$\mathcal{NP}$-hard [19]	decidable [14]	*PSPACE*-hard [28]
PAD	decidable [14] **PSPACE-hard**	**PSPACE-hard**	decidable [14] **PSPACE-hard**	*PSPACE*-hard [28]
PN	decidable [15, 14] *EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PAN	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PRS	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]

The following table summarizes results about the problems of strong and weak finiteness. New results are in boldface.

	strong finiteness	weak finiteness
BPA	$\in 2-EXPTIME$ [5, 4]	?
PDA	**PSPACE-hard**	**PSPACE-hard**
BPP	decidable [13] co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PA	co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PAD	**PSPACE-hard**	**PSPACE-hard**
PN	decidable [13] *EXPSPACE*-hard	undecidable [13]
PAN/PRS	*EXPSPACE*-hard	undecidable [13]

Some more results are known about the restricted subclasses of these systems that satisfy the 'normedness condition' (e.g. [10, 9, 8, 16]). Normedness means that from every reachable state there is a terminating computation. This condition makes many bisimulation problems much easier, e.g., strong bisimilarity of normed BPP is decidable in polynomial time [10], while it is at least co-$\mathcal{NP}$-hard in the general case [19]. Also for normed systems finiteness w.r.t. strong bisimilarity coincides with boundedness [16], while this doesn't hold in the general case.

Acknowledgment: Thanks to Colin Stirling for helpful discussions.

References

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

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

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

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

[5] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

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

[7] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[8] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.

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

[10] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.

[11] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

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

[13] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.

[14] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[15] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.

[16] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[17] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.

[18] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[19] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP'2000*, volume ? of *LNCS*. Springer Verlag, 2000.

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

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

[22] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[23] M. Oyamaguchi, N. Honda, and Y. Inagaki. The equivalence problem for real-time strict deterministic languages. *Information and Control*, 45:90–115, 1980.

[24] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[25] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[26] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 671–681. Springer Verlag, 1997.

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

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