Undecidability of Bisimilarity by Defender's Forcing

PETR JANČAR
Technical University of Ostrava
and
JIŘÍ SRBA
Aalborg University

Stirling (1996, 1998) proved the decidability of bisimilarity on so called normed pushdown processes. This result was substantially extended by Sénizergues (1998, 2005) who showed the decidability of bisimilarity for regular (or equational) graphs of finite out-degree; this essentially coincides with weak bisimilarity of processes generated by (unnormed) pushdown automata where the ε-transitions can only deterministically pop the stack. The question of decidability of bisimilarity for the more general class of so called Type -1 systems, which is equivalent to weak bisimilarity on unrestricted ε-popping pushdown processes, was left open. This was repeatedly indicated by both Stirling and Sénizergues. Here we answer the question negatively, i.e., we show the undecidability of bisimilarity on Type -1 systems, even in the normed case.

We achieve the result by applying a technique we call Defender's Forcing, referring to the bisimulation games. The idea is simple, yet powerful. We demonstrate its versatility by deriving further results in a uniform way. Firstly, we classify several versions of the undecidable problems for prefix rewrite systems (or pushdown automata) as Π_1^0-complete or Σ_1^1-complete. Secondly, we solve the decidability question for weak bisimilarity on PA (Process Algebra) processes, showing that the problem is undecidable and even Σ_1^1-complete. Thirdly, we show Σ_1^1-completeness of weak bisimilarity for so called parallel pushdown (or multiset) automata, a subclass of (labelled, place/transition) Petri nets.

Categories and Subject Descriptors: F.3.2 [**Semantics of Programming Languages**]: Process models; F.4.2 [**Grammars and Other Rewriting Systems**]: Decision problems

General Terms: Theory, Verification

Additional Key Words and Phrases: bisimilarity, pushdown automata, process algebra, undecidability

1. INTRODUCTION

Bisimilarity, or bisimulation equivalence [Park 1981; Milner 1989], has been recognized as a fundamental notion in concurrency theory, in verification of behaviour of (reactive) systems, and in other areas. This has initiated several research directions; one research line explores the (un)decidability and complexity questions for bisimilarity, where the results have turned out to be different than those for

Author's address: P. Jančar, Center of Applied Cybernetics, Faculty of Electrical Engineering and Informatics, Technical University of Ostrava, 17. listopadu 15, CZ – 708 33 Ostrava - Poruba, Czech Republic. J. Srba, Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, DK – 9220 Aalborg East, Denmark. The first author is supported by Grant No. 1M0567 and the second author by Project No. MSM0021622419 of the Ministry of Education of the Czech Republic.

classical language equivalence.

We can recall that (the graph of) a nondeterministic finite automaton (NFA) can be viewed as a finite state labelled transition system (LTS), where accepting states play no role. Informally speaking, two states r, s of an LTS are bisimilar if for any transition $r \stackrel{a}{\longrightarrow} r'$ ($s \stackrel{a}{\longrightarrow} s'$) there is a transition $s \stackrel{a}{\longrightarrow} s'$ ($r \stackrel{a}{\longrightarrow} r'$) such that r', s' are bisimilar. While language equivalence is well known to be PSPACE-complete for NFAs, bisimulation equivalence is solvable by fast polynomial algorithms [Paige and Tarjan 1987; Kanellakis and Smolka 1990]. It also makes good sense to compare states in infinite state LTSs generated by finite descriptions. E.g., a context-free grammar in Greibach normal form naturally generates an infinite state LTS where the states are finite sequences of nonterminals; a rule $X \longrightarrow aY_1Y_2 \dots Y_n$ (here more appropriately written as $X \stackrel{a}{\longrightarrow} Y_1Y_2 \dots Y_n$) induces transitions $X\,u \stackrel{a}{\longrightarrow} Y_1Y_2 \dots Y_n\,u$ for all sequences u. Bisimilarity between such states, called BPA (Basic Process Algebra) processes, turned out to be decidable (in [Baeten et al. 1993] for the normed subclass with no redundant nonterminals, and in [Christensen et al. 1995] for the full class without any restrictions) while language equivalence is well-known to be undecidable already for the normed subclass. This started the research topic of exploring bisimilarity questions on infinite state systems, the history of which is reflected in survey papers like [Moller 1996; Burkart and Esparza 1997; Burkart et al. 2001; Srba 2004; Kučera and Jančar 2006].

The above mentioned BPA systems are a special case of LTSs generated by finite sets of prefix-rewrite rules. PDA systems, generated by classical pushdown automata rules $pX \stackrel{a}{\longrightarrow} qw$, are more general w.r.t. bisimilarity (though equivalent with BPA in the classical language sense). Therefore the decidability results for BPA could not be automatically extended to PDA, and the decidability question for PDA processes (i.e., states in PDA systems) remained a difficult open problem, explicitly formulated, e.g., in [Caucal 1995].

Stirling [Stirling 1996] showed the decidability of bisimilarity for restricted, so called normed, PDA processes. (A PDA process is normed if the stack can be emptied from every reachable configuration.) Decidability for the whole class of PDA processes was later shown by Sénizergues [Sénizergues 1998]. The involved proof extended the technique he used for his famous result showing the decidability of language equivalence for deterministic pushdown automata [Sénizergues 2001] (see also [Stirling 2001]); a complete journal version of the bisimilarity result appeared in [Senizergues 2005]. In fact, the result is more general; we present it in terms of weak bisimilarity, which is a standard generalization of bisimilarity abstracting away the silent (internal) actions, i.e. the ε-transitions in our case. Sénizergues showed the decidability of weak bisimilarity for PDA systems which are 'disjoint' (in the terminology of [Stirling 2001], meaning that no configuration admits both visible and ε-transitions) and where ε-transitions can only deterministically pop the stack. A natural question asks how far this result can be extended. Stirling [Stirling 1996] formulated the bisimilarity question for so called Type -1 and Type -2 systems. For us it suffices to say that this corresponds to the weak bisimilarity problem on (nondeterministic) disjoint ε-popping PDAs and on general disjoint PDAs, respectively. There was a hope that the developed techniques can be extended to

show the decidability for these more general classes, at least in the normed case.

Such a hope was also strengthened by the fact that several nontrivial results achieved for pushdown graphs turned out to be extendable to a more general class of prefix-recognizable graphs, also called REC_{RAT} in [Caucal 1996]; this includes the decidability of monadic second order logic [Caucal 1996] and the existence of uniform winning strategies for parity games [Cachat 2002].

We note that due to a terminology mismatch, it was incorrectly indicated in [Sénizergues 1998] that the positive decidability result applies to Type -1 systems as well. This was later corrected in [Senizergues 2005], where the author made clear that the problem for Type -1 systems, i.e. weak bisimilarity on disjoint ε-popping PDAs, remains open.

The main result of our paper demonstrates that weak bisimilarity on (disjoint) ε-popping PDAs is undecidable, even in the normed case. To provide a deeper insight, we accompany this result by a more detailed analysis of the undecidability degrees for several combinations of types of ε-transitions and (un)normedness. This analysis contributes to the general experience that the undecidable problems naturally arising in computer science are either 'lowly' undecidable, i.e., residing at the first levels of arithmetical hierarchy — typically equivalent to the halting problem or to its complement, and thus Σ_1^0-complete or Π_1^0-complete — or 'highly' undecidable — at the first levels of analytical hierarchy, typically Σ_1^1-complete or Π_1^1-complete. (Later we refer to [Harel 1986], where this phenomenon is also discussed.)

Besides the concrete results, we also consider the technique of their proofs as an important part of our contribution. Bisimilarity, as well as weak bisimilarity, can be naturally presented in terms of two-player games (see also [Thomas 1993; Stirling 1995]). In a current 'position', i.e. in a pair of states in an LTS, Attacker performs a transition from one of the states and Defender must respond by performing a transition with the same action-label from the other state; a new current position thus arises. If one player is stuck then the other player wins, and an infinite play is a win of Defender. Thus two states r, s are bisimilar iff Defender has a winning strategy when starting from (r, s).

Hardness results for (weak) bisimilarity are naturally obtained by constructing suitable bisimulation games corresponding to instances of a hard problem. Due to the nature of the game, it is relatively straightforward to implement situations where Attacker is supposed to determine the following part of a play; if it is Defender who should determine this, implementations are less obvious (if possible at all). In our cases, we have succeeded in implementing this 'Defender's Forcing' ('Defender's Choice' can be another suitable term).

The idea behind this technique is simple, yet powerful. Here we show its versatility by deriving further results in a uniform way. These results deal with the 'middle row' of so called process rewrite systems hierarchy, which is depicted in Figure 1 from [Mayr 2000] and contains the already discussed class PDA and the classes PA, PN. Roughly speaking, a PDA system can be viewed as a finite control unit operating on sequentially composed stack symbols. PA (Process Algebra) systems were introduced in [Baeten and Weijland 1990]; they combine sequential and parallel composition but lack the control unit. PN stands for Petri nets, a well-known model the study of which was initiated in [Petri 1962]. For us it is sufficient to

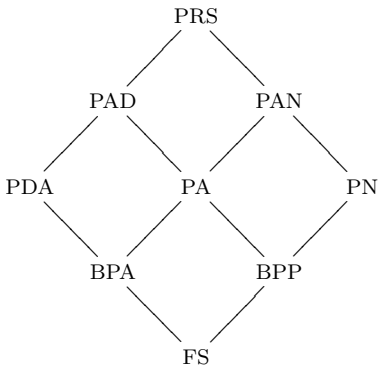

Fig. 1. Process rewrite systems hierarchy

consider the subclass PPDA of PN; a PPDA (parallel PDA) system can be viewed as a finite control unit operating on parallel composition of stack symbols.

By applying the method of Defender's Forcing we derive the undecidability of weak bisimilarity for PA processes; in fact, the problem turns out to be Σ_1^1-complete (and thus highly undecidable). Another application demonstrates Σ_1^1-completeness of weak bisimilarity for PPDAs (and thus also for PNs).

Remark. The decidability question for (strong) bisimilarity on PA processes is still an open problem; in the case of normed PA processes, an involved proof of the decidability was shown in [Hirshfeld and Jerrum 1999]. For the (unrestricted) subclasses BPA, BPP the decidability of bisimilarity was known earlier (see, e.g., [Srba 2004] for references) but the decidability questions for weak bisimilarity on both BPA and BPP are still open. Bisimilarity on Petri nets was shown to be Π_1^0-complete in [Jančar 1995b]; the proof can be easily reformulated for PPDAs (as was done in [Moller 1996]). Our proof for weak bisimilarity strengthens and substantially simplifies the result in [Jančar 1995a].

Structure of the paper. Section 2 contains basic definitions and recalls and/or proves some useful results. Section 3 shows the Π_1^0-complete and Σ_1^1-complete problems we use for the hardness reductions, and Section 4 sketches the main idea of Defender's Forcing. Section 5 contains the results for prefix-rewrite systems and pushdown automata, Section 6 then deals with PA processes, and Section 7 with PPDA (and Petri nets). Section 8 summarizes the results.

2. PRELIMINARIES

2.1 Labelled transition systems, bisimulation equivalence, games

A *labelled transition system* (LTS) is a triple $(S, \mathcal{A}ct, \longrightarrow)$ where S is a set of *states* (or *processes*), $\mathcal{A}ct$ is a set of *labels* (or *actions*), and $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*; for each $a \in \mathcal{A}ct$, we view $\stackrel{a}{\longrightarrow}$ as a binary relation on S where $\alpha \stackrel{a}{\longrightarrow} \beta$ iff $(\alpha, a, \beta) \in \longrightarrow$. The notation can be naturally extended to $\alpha \stackrel{s}{\longrightarrow} \beta$ for finite sequences of actions s; and by $\alpha \longrightarrow^* \beta$ we mean that there is s such that $\alpha \stackrel{s}{\longrightarrow} \beta$.

Given $(S, \mathcal{A}ct, \longrightarrow)$, a binary relation $R \subseteq S \times S$ is a *simulation* iff for each

$(\alpha, \beta) \in R$, $a \in \mathcal{A}ct$, and α' such that $\alpha \xrightarrow{a} \alpha'$ there is β' such that $\beta \xrightarrow{a} \beta'$ and $(\alpha', \beta') \in R$. A *bisimulation* is a simulation which is symmetric. Processes α and β are *bisimilar*, denoted $\alpha \sim \beta$, if there is a bisimulation containing (α, β). We note that bisimilarity is an equivalence relation.

We use a standard game-theoretic characterization of bisimilarity [Thomas 1993; Stirling 1995]. A *bisimulation game* on a pair of processes (α_1, α_2) is a two-player game between *Attacker* and *Defender*; to make some further considerations easier, we view Attacker as "him" and Defender as "her". The game is played in *rounds*. In each round (consisting of two moves) the players change the *current pair of states* (β_1, β_2) (initially $\beta_1 = \alpha_1$ and $\beta_2 = \alpha_2$) according to the following rule:

(1) Attacker chooses $i \in \{1, 2\}$, $a \in \mathcal{A}ct$ and $\beta_i' \in S$ such that $\beta_i \xrightarrow{a} \beta_i'$. He thus creates an *intermediate pair* which is (β_1', β_2) in the case $i = 1$, and (β_1, β_2') in the case $i = 2$.
(2) Defender responds by choosing $\beta_{3-i}' \in S$ such that $\beta_{3-i} \xrightarrow{a} \beta_{3-i}'$. She thus completes one round of the game by playing under the same action a in the other process.
(3) The pair (β_1', β_2') becomes the (new) current pair of states.

Any *play* (of the bisimulation game) thus corresponds to a sequence of pairs of states such that Attacker is making a move from every odd position and Defender from every even one (under the same action which was used by Attacker in the previous move).

A play (and the corresponding sequence) is finite iff one of the players gets stuck (cannot make a move); the player who got stuck lost the play and the other player is the winner. (A play finishing in an intermediate pair on an even position is winning for Attacker and a play finishing on an odd position is winning for Defender.) If the play is infinite then Defender is the winner.

We note that all possible plays from a pair (α, β) can be naturally organized in a tree where each vertex is labelled with a pair of processes, the root being labelled with (α, β), and each edge, labelled by an action, corresponds to a move of Attacker or Defender under that action. Each vertex on an odd level (the root is assumed to be on level one) corresponds to a situation where Attacker chooses a move (i.e., an outgoing edge to an intermediate pair on the next level), and each vertex on an even level corresponds to a situation where Defender chooses a response (i.e., an outgoing edge with the same label as the label of the incoming edge). A *strategy of Attacker* from a pair (α, β) can be viewed as a tree arising from the above "all-plays tree" by removing all but one outgoing edge (with the corresponding subtrees) from each odd-level vertex with more than one (immediate) successor. Analogously, *a strategy of Defender* arises by removing all but one outgoing edge from each even-level vertex with more than one (immediate) successor. A strategy of a player is a *winning strategy (WS)* if every branch (corresponding to a play) is winning for the player.

We now recall the following standard fact.

PROPOSITION 2.1. *It holds that $\alpha_1 \sim \alpha_2$ iff Defender has a winning strategy in the bisimulation game starting with the pair (α_1, α_2); and $\alpha_1 \not\sim \alpha_2$ iff Attacker has a winning strategy.*

Sometimes we assume that the set $\mathcal{A}ct$ of actions contains a distinguished *silent action* τ. The *weak transition relation* $\Longrightarrow$ is defined by $\stackrel{\tau}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^*$ and $\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} (\stackrel{\tau}{\longrightarrow})^* \circ \stackrel{a}{\longrightarrow} \circ (\stackrel{\tau}{\longrightarrow})^*$ for $a \in \mathcal{A}ct \setminus \{\tau\}$.

Given an LTS $(S, \mathcal{A}ct, \longrightarrow)$, a binary relation $R \subseteq S \times S$ is a *weak simulation* iff for each $(\alpha, \beta) \in R$, $a \in \mathcal{A}ct$, and α' such that $\alpha \stackrel{a}{\longrightarrow} \alpha'$ there is β' such that $\beta \stackrel{a}{\Longrightarrow} \beta'$ and $(\alpha', \beta') \in R$. A *weak bisimulation* is a weak simulation which is symmetric. Processes α and β are *weakly bisimilar*, denoted $\alpha \approx \beta$, if there is a weak bisimulation containing (α, β). (Weak bisimilarity is also an equivalence relation.)

An analogue of Proposition 2.1 for weak bisimilarity and the *weak bisimulation game* can be also easily derived.

Remark 2.2. Nothing changes when we allow the 'long' moves $\alpha \stackrel{a}{\Longrightarrow} \alpha'$ to be played by Attacker as well (he can do them by a sequence of 'short' moves $\stackrel{a}{\longrightarrow}$, $\stackrel{\tau}{\longrightarrow}$ anyway). The weak bisimulation game can be then viewed as the ('strong') bisimulation game played on the modified LTS (in which $\Longrightarrow$ is the transition relation). Allowing only 'short' moves to Attacker can just technically ease analysis of some concrete cases.

2.2 Prefix rewrite systems, pushdown automata, normedness

We now define special classes of labelled transition systems (LTSs), namely those generated by systems of prefix rewrite rules.

The most general prefix rewrite systems we consider are *Type -2 systems* (in the terminology to be mentioned later). Such a system $\mathcal{S}$ can be viewed as a triple $\mathcal{S} = (\Gamma, \mathcal{A}ct, \Delta)$ where Γ is a finite set of *process symbols*, $\mathcal{A}ct$ is a finite set of *actions*, and Δ is a finite set of *rewrite rules*. Each rewrite rule is of the form $R_1 \stackrel{a}{\longrightarrow} R_2$ where $a \in \mathcal{A}ct$ and R_1 and R_2 are regular languages over Γ such that $\varepsilon \notin R_1$ (ε denotes the empty sequence); for concreteness, we can assume that R_1, R_2 are given by regular expressions.

System $\mathcal{S} = (\Gamma, \mathcal{A}ct, \Delta)$ represents the LTS $(\Gamma^*, \mathcal{A}ct, \longrightarrow)$ defined as follows. A process (a state in the LTS) is any finite sequence of process symbols, i.e., any element of Γ^*; we use $u, v, w, \ldots$ for denoting elements of Γ^*. The transition relation $\longrightarrow$ (i.e., the collection of relations $\stackrel{a}{\longrightarrow}$ for $a \in \mathcal{A}ct$) is defined by the following derivation rule:

$$\frac{(R_1 \stackrel{a}{\longrightarrow} R_2) \in \Delta, \quad w \in R_1, \quad w' \in R_2, \quad u \in \Gamma^*}{wu \stackrel{a}{\longrightarrow} w'u}$$

Thus any rule $(R_1 \stackrel{a}{\longrightarrow} R_2) \in \Delta$ represents possibly infinitely many rewrite rules $w \stackrel{a}{\longrightarrow} w'$ where $w \in R_1$ and $w' \in R_2$.

We also use the notion of normedness. We say that a process $w \in \Gamma^*$ is *normed* if for any w' such that $w \longrightarrow^* w'$ we have $w' \longrightarrow^* \varepsilon$. In other words, a process w is normed iff any finite path from w in the respective LTS can be prolonged to finish in ε. A *norm* of a normed process w, denoted by $norm(w)$, is the length of the shortest action sequence s such that $w \stackrel{s}{\longrightarrow} \varepsilon$.

We note the following two propositions regarding normedness.

PROPOSITION 2.3. *If two normed processes are bisimilar then they have the same norm.*

PROOF. Assume normed u, v with $norm(u) < norm(v)$. For the shortest sequence s such that $u \xrightarrow{s} \varepsilon$ we have: if $v \xrightarrow{s} v'$ then v' is normed and $v' \neq \varepsilon$ (thus v' can perform an action). This implies that u and v are not bisimilar. □

PROPOSITION 2.4. *There is an algorithm which decides whether a given Type -2 process v is normed, and computes its norm in the positive case.*

PROOF. We can base the algorithm on the well-known fact regarding (classical) pushdown automata: given a pushdown automaton and an initial (state × stack) configuration, the set of all (state × stack) configurations reachable from the initial one is regular, and its representation can be effectively constructed [Büchi 1964] (further useful references are [Bouajjani et al. 1997; Esparza et al. 2000]).

We observe (see also [Stirling 2000]) that applying a rule $R_1 \xrightarrow{a} R_2$ to v, i.e., replacing a prefix $w \in R_1$ of v by $w' \in R_2$, can be implemented by a series of ε-moves of a pushdown automaton (whose control unit includes finite automata for R_1, R_2).

In this way we can easily derive that, given a Type -2 system and a process v, the set $post^*(v)$ — consisting of all processes reachable from v — is an effectively constructible regular set. Similarly, the set $pre^*(\varepsilon)$ — consisting of all processes from which ε is reachable — is an effectively constructible regular set. Checking normedness of v now amounts to verifying whether $post^*(v) \subseteq pre^*(\varepsilon)$.

Computing $norm(v)$ for a normed v can be accomplished by stepwise constructing $pre(\varepsilon)$, $pre(\, pre(\varepsilon)\,)$, $pre(\, pre(\, pre(\varepsilon)\,)\,)$, ... until v is included; here $pre(R)$ denotes the set of processes from which some $u \in R$ is reachable in one step. Such a computation can be again easily reduced to computing the sets of reachable configurations of pushdown automata. □

The other systems we consider arise from the above defined Type -2 systems by restricting the form of rewrite rules. We use the terminology introduced by Stirling (see, e.g., [Stirling 2003]). In Figure 2, R_1, R_2 and R stand for regular sets over Γ; w, w' stand for elements of Γ^* (the respective regular languages are thus singletons); and X, Y, p, q stand for elements of Γ. We have added Type -1b to Stirling's table; his Type -1 coincides with our Type -1a.

Remark 2.5. The classes Type -1a and Type -1b are incomparable w.r.t. bisimilarity and strictly above Type 0 and below Type -2 systems. (We show this in the report [Jančar and Srba 2006].)

We can note that Type $1\frac{1}{2}$ rules are classical pushdown rules (p, q are 'highlighted' as elements of the set of control states which is disjoint with the stack alphabet). In this paper we thus take the term *pushdown automaton (PDA)* as a synonym of a Type $1\frac{1}{2}$ system; the class of LTSs defined by PDAs was shown to coincide (up to isomorphism) with the class of Type 0 systems [Caucal 1992]. For technical convenience, we also view $p \xrightarrow{a} qw$, $pX \xrightarrow{a} \varepsilon$, $p \xrightarrow{a} \varepsilon$ as PDA-rules.

We say that a PDA is *ε-popping* if the τ-rules are only of the form $pX \xrightarrow{\tau} q$. A PDA is *ε-pushing* if the τ-rules are only of the form $pX \xrightarrow{\tau} qXw$ (or $p \xrightarrow{\tau} qw$).

Type	Form of Rewrite Rules
Type -2	$R_1 \xrightarrow{a} R_2$
Type -1a/-1b	$R \xrightarrow{a} w \;/\; w \xrightarrow{a} R$
Type 0	$w \xrightarrow{a} w'$
Type $1\frac{1}{2}$	$pX \xrightarrow{a} qw$
Type 2	$X \xrightarrow{a} w$
Type 3	$X \xrightarrow{a} Y,\ X \xrightarrow{a} \varepsilon$

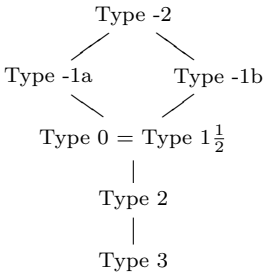

Fig. 2. Hierarchy of prefix rewrite systems

(We use the term 'ε-popping' rather than 'τ-popping' to recall the standard notion of ε-transitions of a pushdown automaton.)

For completeness, we add that Type 2 systems are also called BPA (Basic Process Algebra) systems, and Type 3 systems correspond to finite labelled transition systems.

Remark 2.6. The normedness of a PDA-process is usually defined as the ability to empty the stack from every reachable configuration. Such a normed PDA can be easily transformed to match our definition of normedness (of a Type -2 system) by using a special bottom-of-the-stack symbol $\perp$ and by adding the rules $q\perp \xrightarrow{e} \varepsilon$, where e is a special 'end'-action.

2.3 PA processes

The term PA comes from 'Process Algebra'. It is a formalism including both sequential and parallel composition and it was introduced by Baeten and Weijland [Baeten and Weijland 1990]. A *PA process rewrite system* (i.e., a $(1, G)$-PRS in the terminology of [Mayr 2000]) can be viewed as a triple $(\Gamma, \mathcal{A}ct, \Delta)$, similarly as a prefix rewrite system. However, the *(PA-)processes*, i.e., the states in the respective LTS, are not just sequences of process symbols from Γ; they can combine sequential and parallel composition, being defined by

$$P ::= \varepsilon \mid X \mid P.P \mid P \| P \quad \text{where } X \text{ ranges over } \Gamma.$$

The rewrite rules in Δ are now (only) of the form $X \xrightarrow{a} P$, where $X \in \Gamma$, $a \in \mathcal{A}ct$ and P is a (PA-)process. The transition relation is given by the following derivation rules.

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

As expected, we can easily derive that the parallel composition can be viewed as commutative and associative (while sequential composition is only associative).

Normedness of PA processes has the same meaning as in prefix rewrite systems: a process P is *normed* if every finite path starting from P (in the respective LTS) can be prolonged to reach ε.

2.4 Parallel pushdown automata, Petri nets

In our framework, a *Petri net* can be viewed as a triple $(\Gamma, \mathcal{A}ct, \Delta)$ where the processes (i.e., Petri net markings) are just parallel compositions of process symbols. A process α can then be viewed just as a mapping (or a multiset) $\alpha : \Gamma \to \mathbb{N}$. The rules have the form $\beta \stackrel{a}{\longrightarrow} \gamma$ where β, γ are processes (markings). The derivation rule for transitions can be presented as follows.

$$\frac{\beta \stackrel{a}{\longrightarrow} \gamma}{\beta \| \alpha \stackrel{a}{\longrightarrow} \gamma \| \alpha}$$

Parallel pushdown automata (PPDA) constitute a strict subclass of Petri nets. A PPDA arises from a (usual) PDA when we view the 'stack' as parallel composition of symbols (which is associative and commutative). We can thus use the same notation for PPDAs (and their rules) as for PDAs. But the rule $pX \stackrel{a}{\longrightarrow} qw$ is now performable whenever the current control state is p and X occurs somewhere in the stack (which is a multiset of stack-symbols).

2.5 (Weak) bisimilarity problems in Π_1^0 and Σ_1^1

The results of this paper establish Π_1^0-completeness and Σ_1^1-completeness of the (weak) bisimilarity problems for several classes of LTSs (introduced above). It is the hardness part which is crucial; the membership in Π_1^0 or Σ_1^1 can be demonstrated easily, and we thus handle this here.

Let us recall that the class Π_1^0 (in the arithmetical hierarchy) consists of problems whose complements are semidecidable, i.e., of those problems which have algorithms halting just on the negative instances.

Recall also that to demonstrate that α, β are *not* bisimilar, it is sufficient to present a winning strategy (WS) for Attacker, from the pair (α, β). Such a strategy can be viewed as a tree which was described before Proposition 2.1. We note that each branch of the tree corresponds to a possible play when Attacker plays according to the assumed winning strategy; each branch is thus finite (finishing by Defender's getting stuck).

In *image finite systems* (for each α, a, the set $\{\beta \mid \alpha \stackrel{a}{\longrightarrow} \beta\}$ is finite), this WS is a finite tree, and all 'candidate' trees can thus be systematically generated by an algorithm. This well-known idea (see, e.g.,[Hennessy and Milner 1985]) immediately yields the following proposition.

PROPOSITION 2.7. *Bisimilarity is in Π_1^0 for Type -1a systems, for PA systems, and for Petri nets. Weak bisimilarity is in Π_1^0 for ε-popping PDAs.*

Bisimilarity (i.e., the problem if two given processes in a given system are bisimilar) is in Π_1^0 also for normed Type -1b systems, though they are not image finite. The idea is that a WS (winning strategy) for Attacker does not need to count with 'too long' responses of Defender. We derive this as a consequence of the next proposition which is formulated more generally.

We say that an LTS $(S, \mathcal{A}ct, \longrightarrow)$ is *effective* iff both S and $\mathcal{A}ct$ are decidable subsets of the set of all finite strings in a given finite alphabet and the relation $\longrightarrow$ is decidable. An LTS $(S, \mathcal{A}ct, \longrightarrow)$ is called *finitely over-approximable* (w.r.t. bisimilarity) iff for any $\alpha, \beta \in S$ and $a \in \mathcal{A}ct$, a finite set $E_{(\alpha,\beta,a)} \subseteq S$ can be

effectively constructed so that whenever $\beta \xrightarrow{a} \beta'$ and $\beta' \sim \alpha$ then $\beta' \in E_{(\alpha,\beta,a)}$. Thus the (finite) set $E_{(\alpha,\beta,a)}$ over-approximates the set of a-successors of β which are bisimilar with α.

PROPOSITION 2.8. *Bisimilarity on effective and finitely over-approximable labelled transition systems is in* Π_1^0.

PROOF. The above mentioned tree demonstrating a WS for Attacker can be assumed finite since at each node (α, β) where Defender is obliged to response by a move $\beta \xrightarrow{a} \beta'$ it suffices to consider only (the finitely many) $\beta' \in E_{(\alpha,\beta,a)}$. It is thus sufficient to systematically generate all finite trees and check for each of them if it happens to represent a WS for Attacker; the checking can be done algorithmically due to our effectiveness assumptions. □

COROLLARY 2.9. *Bisimilarity is in* Π_1^0 *for* normed *Type -1b systems. Weak bisimilarity is in* Π_1^0 *for normed* ε*-pushing PDAs.*

PROOF. Type -1b systems are obviously effective; it is thus sufficient to show that *normed* Type -1b systems are finitely over-approximable. We recall that normed bisimilar processes must have equal norms (Proposition 2.3), and we note that $norm(u) \geq |u|/k$ where k is the length of the longest left-hand side in the rules $w \xrightarrow{a} R$ of the respective normed Type -1b system (and $|u|$ is the length of u). Since $norm(u)$ is computable by Proposition 2.4, the required (finite) set $E_{(u,v,a)}$ for given processes u, v and an action a can be defined as $\{\, v' \mid |v'| \leq k \cdot norm(u) \,\}$. The argument for normed ε-pushing PDAs is the same. □

Membership in the class Σ_1^1 (of the analytical hierarchy) is obvious for all (weak) bisimilarity problems considered in this paper: processes α and β are (weakly) bisimilar iff *there exists a set* of pairs which contains (α, β) and satisfies the (1st order arithmetic definable) conditions required by the definition of (weak) bisimulation.

PROPOSITION 2.10. *(Weak) bisimilarity is in* Σ_1^1 *for Type -2 systems, for PA systems, and for Petri nets.*

3. PROBLEMS FOR HARDNESS REDUCTIONS

In the first subsection we define some variants of Post's Correspondence Problem (PCP) which are suitable for deriving hardness results in the case of prefix-rewrite systems and PA-systems (since these systems naturally model sequences).

The second subsection recalls some problems for (Minsky) counter machines; these machines constitute a universal computational model which is particularly suitable for deriving hardness results for PPDA (or Petri nets).

3.1 Variants of Post's Correspondence Problem

For our aims, a *PCP-instance* INST is defined as a nonempty sequence $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ of pairs of nonempty words over the alphabet $\{A, B\}$ where $|u_i| \leq |v_i|$ for all $i \in \{1, 2, \ldots, n\}$ ($|u|$ denoting the length of u).

An *infinite initial solution* of a given PCP instance is an infinite sequence of indices $i_1, i_2, i_3, \ldots$ from the set $\{1, 2, \ldots, n\}$ such that i_1=1 and the infinite words

$u_{i_1}u_{i_2}u_{i_3}\cdots$ and $v_{i_1}v_{i_2}v_{i_3}\cdots$ are equal. A *recurrent solution* is an infinite initial solution in which index 1 appears infinitely often.

By *inf-PCP* we denote the problem to decide whether a given PCP instance has an infinite initial solution; *rec-PCP* denotes the problem to decide whether a given PCP instance has a recurrent solution.

PROPOSITION 3.1. *(1) Problem inf-PCP is Π_1^0-complete. (2) Problem rec-PCP is Σ_1^1-complete.*

These facts can be easily established from well-known results; we refer, e.g., to [Ruohonen 1985] for the (low) undecidability and to [Harel 1986] for the high undecidability.

Remark 3.2. Our (additional) requirement $|u_i| \leq |v_i|$ is non-standard but it can be easily checked to be harmless for the validity of Proposition 3.1; we use it for its technical convenience. This follows directly from the standard textbook reduction of the halting problem to PCP. For example, the reduction provided in [Sipser 2005, Chapter 5.2] consists of seven parts. Parts 1. to 5. deal with the simulation of a Turing machine computation and produce an instance of PCP which satisfies our requirement $|u_i| \leq |v_i|$. The last two categories of pairs of strings in Part 6. and Part 7. are used to equalize the lengths of the two generated words in case that an accepting configuration is reached. Since our question is about the existence of an infinite computation, we can safely omit the pairs from the last two categories.

It is also useful to note the following obvious fact, which will be implicitly used in later reasoning. By a *partial solution* of a PCP-instance $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ we mean a finite sequence $i_1, i_2, i_3, \ldots, i_\ell$ such that $u_{i_1}u_{i_2}\ldots u_{i_\ell}$ is a prefix of $v_{i_1}v_{i_2}\ldots v_{i_\ell}$.

PROPOSITION 3.3. *Given a PCP-instance and a sequence $i_1, i_2, i_3, \ldots$ of indices where $i_1 = 1$, the following three conditions are equivalent:*

—$i_1, i_2, i_3, \ldots$ is an infinite initial solution,

—for each ℓ, the sequence $i_1, i_2, i_3, \ldots, i_\ell$ is a partial solution,

—for infinitely many ℓ, the sequence $i_1, i_2, i_3, \ldots, i_\ell$ is a partial solution.

3.2 Minsky counter machines

A *(Minsky) counter machine (MCM) M*, with *nonnegative* counters $c_1, c_2, \ldots, c_m$, is a sequence of (labelled) instructions:

$$1 : instr_1;\ 2 : instr_2;\ \ldots\ n : instr_n$$

where $instr_n$ = HALT and $instr_i$, for $i \in \{1, 2, \ldots, n-1\}$, are of the following two types (assuming $j, k \in \{1, 2, \ldots, n\}$, $r \in \{1, 2, \ldots, m\}$):

Type (1) $c_r := c_r + 1$; goto j

Type (2) if $c_r = 0$ then goto j else ($c_r := c_r - 1$; goto k)

The instructions of type (1) are called *increment instructions*, the instruction of type (2) are *zero-test (and decrement) instructions*.

The *computation* of M on the input $(i_1, i_2, \ldots, i_m)$ is the sequence of configurations $(i, n_1, n_2, \ldots, n_m)$, starting with $(1, i_1, i_2, \ldots, i_m)$, where $i \in \{1, 2, \ldots, n\}$ is

the label of the instruction to be performed, and $n_1, n_2, \ldots, n_m \in \mathbb{N}$ are the (current) counter values; the sequence is determined by the instructions in the obvious way. The computation is either finite, i.e. halting by reaching the instruction n : HALT, or infinite.

We define *inf-MCM* as the problem to decide whether the computation of a given 2-counter MCM on $(0,0)$ is infinite, and we recall the following well-known fact.

PROPOSITION 3.4. *Problem inf-MCM is* Π^0_1*-complete.*

A *nondeterministic* Minsky counter machine is defined as MCM above but with an additional type of instructions, the *nondeterministic choice*:

Type (3) goto j or goto k

We define *rec-NMCM* as the problem to decide whether a given 2-counter nondeterministic MCM has an infinite computation on $(0,0)$ which uses instruction 1 infinitely often. We can refer to [Harel 1986] regarding the following fact.

PROPOSITION 3.5. *Problem rec-NMCM is* Σ^1_1*-complete.*

4. DEFENDER'S FORCING

An important ingredient in the constructions of our reductions is a method which we call "Defender's Forcing" (DF). Here we just sketch the main idea informally and abstractly; later sections provide examples of concrete applications.

When deriving a hardness result, it is often useful to model a (nondeterministic) 'computation', i.e. a sequence of configurations, of a system by means of the (weak) bisimulation game on a suitably created LTS. A play then gives rise to two (slightly differing) representants of a concrete computation. E.g., let us assume that the pair (α, β) represents a current configuration C and (α_1, β_1), (α_2, β_2), ..., (α_n, β_n) are the pairs representing the configurations $C_1, C_2, \ldots, C_n$, respectively, which are the possible outcomes of one computational step from C. If we want that it is Attacker who chooses the next configuration, an obvious implementation in the bisimulation game is the following: we use actions $a_1, a_2, \ldots, a_n$ and put $\alpha \xrightarrow{a_i} \alpha_i$, $\beta \xrightarrow{a_i} \beta_i$ for all $i \in \{1, 2, \ldots, n\}$. Attacker can choose i and play $(\alpha, \beta) \xrightarrow{a_i} (\alpha_i, \beta)$. Defender can only answer from the intermediate pair by $(\alpha_i, \beta) \xrightarrow{a_i} (\alpha_i, \beta_i)$. Hence Attacker forces the next configuration (α_i, β_i).

The situation where we need that it is Defender who chooses the next configuration is more complicated. An (abstract) solution can look like this: we introduce auxiliary states α' and $\beta'_1, \beta'_2, \ldots, \beta'_n$ and create the following transitions, also called *rules* (to anticipate the later use).

$$\alpha \xrightarrow{a} \alpha' \qquad \boxed{\alpha \xrightarrow{a} \beta'_i} \qquad\qquad \beta \xrightarrow{a} \beta'_i$$

$$\alpha' \xrightarrow{a_i} \alpha_i \qquad\qquad \beta'_i \xrightarrow{a_i} \beta_i \qquad \boxed{\beta'_i \xrightarrow{a_j} \alpha_j} \text{ for } i \neq j$$

Here subscripts i, j range over $\{1, 2, \ldots, n\}$; thus the rule $\alpha \xrightarrow{a} \beta'_i$ stands for the n rules $\alpha \xrightarrow{a} \beta'_1$, $\alpha \xrightarrow{a} \beta'_2$, ..., $\alpha \xrightarrow{a} \beta'_n$, the rule $\beta'_i \xrightarrow{a_j} \alpha_j$, $i \neq j$, stands for

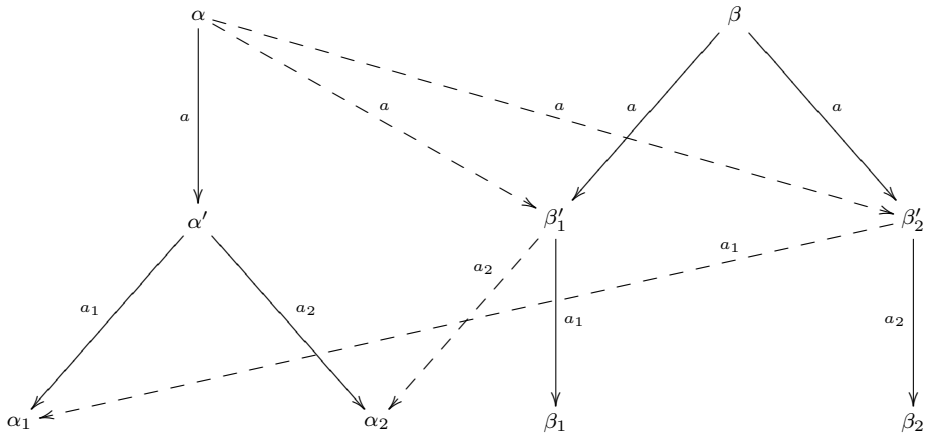

Fig. 3. From (α, β) Defender chooses and forces (α_1, β_1) or (α_2, β_2)

$n(n-1)$ rules like $\beta'_1 \xrightarrow{a_2} \alpha_2$, $\beta'_8 \xrightarrow{a_5} \alpha_5$, etc. Figure 3 presents the rules graphically for the case $n = 2$; the framed rules are represented by the dashed arrows.

By using frames we have highlighted the use of DF; Attacker must make sure that the framed rules are never used (neither by him or her) since otherwise Defender can install a pair with equal components — an obvious win for her. Starting from (α, β), Attacker is thus forced to play $\alpha \xrightarrow{a} \alpha'$, reaching an intermediate pair (α', β), and it is now Defender who chooses i and plays $\beta \xrightarrow{a} \beta'_i$. In the resulting pair (α', β'_i) Attacker is forced to use the action a_i, and the only answer of Defender then installs the pair (α_i, β_i).

The idea can be generalized in several ways, e.g., to handle infinitely many possible outcomes (by using τ-rules). The main technical problem is if and how this idea of forcing can be realized in particular classes of LTSs.

5. PREFIX REWRITE SYSTEMS, PUSHDOWN AUTOMATA

5.1 Π^0_1-completeness

In this subsection we show how the Π^0_1-complete problem inf-PCP can be reduced to the bisimulation game on several subclasses of prefix rewrite systems and pushdown automata. We use the subclasses for which the (weak) bisimilarity problem is in Π^0_1, thus deriving Π^0_1-completeness results.

Let us consider a fixed instance INST of inf-PCP, i.e., a sequence of pairs $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ over the alphabet $\{A, B\}$; symbols $I_1, I_2, \ldots, I_n$ will represent the indices $1, 2, \ldots, n$. We can imagine the following abstract game: starting with the one-element sequence I_1, Attacker repeatedly asks Defender to prolong the current sequence $I_{i_1} I_{i_2} \ldots I_{i_\ell}$ by one I_i (of her choice), and eventually switches to check if the current sequence represents a partial solution (i.e., if $u_{i_1} u_{i_2} \ldots u_{i_\ell}$ is a prefix of $v_{i_1} v_{i_2} \ldots v_{i_\ell}$); the negative case is a win for Attacker, the positive case is a win for Defender. It is obvious that INST has an (infinite initial) solution iff Defender has a WS in the described game.

We now implement this abstract game as the bisimulation game starting with the

pair $(\, q_0 I_1 \bot,\ q'_0 I_1 \bot \,)$ of processes of the below described normed Type -1a system. Since we use prefix rewriting, it is convenient to represent the current sequences in the reversed order, as $I_{i_\ell} I_{i_{\ell-1}} \dots I_{i_1}$, and prolong them 'to the left'. The special symbol $\bot$ is used as an endmarker (the bottom-of-the-stack symbol) which here just helps to guarantee normedness. The final checking, i.e., the *verification phase*, can be naturally performed on two copies of the sequence of indices, where one sequence is interpreted over u_i's, the other over v_i's. These two copies arise in the *generating phase*, and they also serve for enabling DF (Defender's Forcing). We first give all the rules of the Type -1a system, and then explain them in detail.

Notation. We let I^* stand for the regular expression $(I_1 + I_2 + \cdots + I_n)^*$. By u^R we denote the reverse image of u. By $head(w)$ we denote the first symbol of w; $tail(w)$ is the rest of w. By $h(w)$ (head-action) we mean a if $head(w) = A$, and b if $head(w) = B$. Subscripts i, j range over $\{1, 2, \dots, n\}$; thus the rule $q_0 \xrightarrow{g} p_i$ stands for the n rules $q_0 \xrightarrow{g} p_1$, $q_0 \xrightarrow{g} p_2$, ..., $q_0 \xrightarrow{g} p_n$, the rule $p_i \xrightarrow{a_j} q_0 I_j$, $i \neq j$, stands for $n(n-1)$ rules like $p_1 \xrightarrow{a_2} q_0 I_2$, $p_8 \xrightarrow{a_5} q_0 I_5$, etc.

(G1) rules:
$$q_0 \xrightarrow{g} t \qquad \boxed{q_0 \xrightarrow{g} p_i} \qquad q'_0 \xrightarrow{g} p_i$$
$$t \xrightarrow{a_i} q_0 I_i \qquad p_i \xrightarrow{a_i} q'_0 I_i \qquad \boxed{p_i \xrightarrow{a_j} q_0 I_j} \quad \text{where } i \neq j$$

(S1) rules:
$$q_0 \xrightarrow{s} q_u \qquad \boxed{q_0 (I^*) I_i \xrightarrow{s} q_v w} \qquad q'_0 (I^*) I_i \xrightarrow{s} q_v w \quad \text{for all suffixes } w \text{ of } v_i^R$$

(V1) rules:
$$q_u I_i \xrightarrow{h(u_i^R)} q_u\ tail(u_i^R) \qquad q_v I_i \xrightarrow{h(v_i^R)} q_v\ tail(v_i^R)$$
$$q_u A \xrightarrow{a} q_u \qquad q_v A \xrightarrow{a} q_v$$
$$q_u B \xrightarrow{b} q_u \qquad q_v B \xrightarrow{b} q_v$$
$$q_u \bot \xrightarrow{e} \varepsilon \qquad q_v \bot \xrightarrow{e} \varepsilon$$

As introduced in Section 4, the frames highlight the use of DF; Attacker must make sure that the framed rules are never used since otherwise Defender can install a pair with (syntactically) equal components. E.g., if Attacker wants to use the action g in a pair $(q_0 w, q'_0 w)$ then he is forced to play $q_0 \xrightarrow{g} t$; in a pair $(tw, p_i w)$ Attacker is forced to use the action a_i, etc.

To show that $q_0 I_1 \bot \sim q'_0 I_1 \bot$ iff INST has an (infinite initial) solution, we first assume that there is such a (fixed) solution $i_1, i_2, i_3, \dots$ and describe a WS for Defender. The play starts with the pair $(q_0 I_1 \bot, q'_0 I_1 \bot)$, and as long as Attacker uses (G1)-rules, Defender forces that the play goes through longer and longer pairs

$$(\, q_0 I_{i_\ell} I_{i_{\ell-1}} \dots I_{i_1} \bot,\ q'_0 I_{i_\ell} I_{i_{\ell-1}} \dots I_{i_1} \bot \,) \quad (*)$$

where i_1=1 and $I_{i_1}, I_{i_2}, \dots, I_{i_\ell}$ represents a prefix of the assumed (fixed) solution $i_1, i_2, i_3, \dots$. We observe that Defender can guarantee this since it is her who chooses

$I_{i_2}, I_{i_3}, \ldots$.

Hence if Attacker wants to win, he has to *switch* (from generating to verification), i.e., to use (S1)-rules in some pair $(*)$; he is then forced to use $q_0 \xrightarrow{s} q_u$. Defender answers by shortening the 'right-hand side' sequence so that the resulting pair

$$(\, q_u I_{i_\ell} I_{i_{\ell-1}} \ldots I_{i_1} \bot \,,\ q_v w\, I_{i_m} I_{i_{m-1}} \ldots I_{i_1} \bot \,) \qquad (**)$$

satisfies

$$(u_{i_\ell})^R (u_{i_{\ell-1}})^R \ldots (u_{i_1})^R = w\,(v_{i_m})^R (v_{i_{m-1}})^R \ldots (v_{i_1})^R \ . \tag{1}$$

Finally the (deterministic) (V1)-rules clearly show that Defender wins.

If INST has no solution then there is an obvious WS for Attacker. He repeatedly uses (G1) until a pair $(*)$ which does not correspond to a partial solution appears. This will eventually happen. Then Attacker switches, using $q_0 \xrightarrow{s} q_u$, and after Defender's response we must get a pair $(**)$ where the condition (1) does not hold. Thus the following verification phase is clearly winning for Attacker.

The given Type -1a system (G1), (S1), (V1) is obviously normed. We note that it can be easily modified to yield an equivalent instance of the weak bisimilarity problem for (normed) PDA; we just replace (S1) with

(S1-τ) rules: $\quad q_0 \xrightarrow{s} q_u \qquad q_0' \xrightarrow{s} r'$

$$\boxed{q_0 \xrightarrow{s} r'} \qquad r' I_i \xrightarrow{\tau} r' \,|\, q_v w \qquad \text{for all suffixes } w \text{ of } v_i^R.$$

Notation. We have used here (and we will also use later) the convention that a collection of rules $\alpha \xrightarrow{a} \beta_1$, $\alpha \xrightarrow{a} \beta_2$, …, $\alpha \xrightarrow{a} \beta_m$ can be written more concisely as $\alpha \xrightarrow{a} \beta_1 \,|\, \beta_2 \,|\, \ldots \,|\, \beta_m$; the separation symbol "$|$" should not be confused with the symbol "$\|$" used for parallel composition.

We observe that (S1-τ) rules can be viewed as a faithful implementation of (S1); Defender cannot gain by finishing the τ-sequence before reaching q_v: Attacker could then perform any sequence of τ-moves on the right-hand side himself while the left-hand side (starting with q_u) is 'frozen' since no τ-moves are available there. We also note that the arising PDA (with rules (G1), (S1-τ), (V1)) can be easily made ε-popping — by viewing each $q_v w$ in the (S1-τ) rules as a special control state and adding the appropriate rules to (V1).

Another observation is that instead of using rules of the type $R \longrightarrow w$ (for shortening the v_i-side during the switching phase) we can use the rules $w \longrightarrow R$ for prolonging the u_i-side, preserving the property that INST has a solution iff Defender has a WS:

(S2) rules: $\quad q_0' \xrightarrow{s} q_v$

$$q_0 \xrightarrow{s} q_u(A+B)^* \qquad \boxed{q_0' \xrightarrow{s} q_u(A+B)^*}$$

(S2-τ) rules: $\quad q_0' \xrightarrow{s} q_v$

$$q_0 \xrightarrow{s} r \qquad \boxed{q_0' \xrightarrow{s} r}$$

$$r \xrightarrow{\tau} rA \,|\, rB \,|\, q_u$$

Recalling Propositions 3.1(1) and 2.7 and Corollary 2.9, we thus get the following theorems.

THEOREM 5.1. *The bisimilarity problem is* Π_1^0*-complete*

—*for both normed and unrestricted Type -1a systems* ($R \longrightarrow w$)*; shown by (G1, S1, V1),*

—*for normed Type -1b systems* ($w \longrightarrow R$)*; shown by (G1, S2, V1).*

THEOREM 5.2. *The weak bisimilarity problem is* Π_1^0*-complete*

—*for both normed and unrestricted* ε*-popping PDA; shown by (G1, S1-*τ*, V1),*

—*for normed* ε*-pushing PDA; shown by (G1, S2-*τ*, V1).*

5.2 Σ_1^1-completeness

We again consider a fixed sequence $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ over the alphabet $\{A, B\}$, now as an instance INST of rec-PCP. We modify the previously described abstract game as follows: starting with the one-element sequence I_1, Attacker repeatedly asks Defender to prolong the current sequence $I_{i_1} I_{i_2} \ldots I_{i_\ell}$ by a (finite) *segment* $I_{i_{\ell+1}}, I_{i_{\ell+2}}, \ldots, I_{i_{\ell+m}}$ where $i_{\ell+m} = 1$, and eventually switches to check if the current sequence represents a partial solution. We can easily verify that INST has a (recurrent) solution iff Defender has a WS in the modified game.

For implementation we need to modify the generating rules, using now rules of the type $R_1 \longrightarrow R_2$ (or ε-popping and ε-pushing steps of PDAs) to this aim. The new generating rules are the following (we recall our convention that the subscript i ranges over $\{1, 2, \ldots, n\}$).

(G2) rules:

$$q_0 \xrightarrow{g} t$$

$$\boxed{q_0 \xrightarrow{g} t' I_1 I^*} \qquad q_0' \xrightarrow{g} t' I_1 I^*$$

$$t' \xrightarrow{g} q_0'$$

$$tI^* \xrightarrow{g} q_0 I^* \qquad \boxed{t' I^* \xrightarrow{g} q_0 I^*}$$

(G2-τ) rules:

$$q_0 \xrightarrow{g} t$$

$$\boxed{q_0 \xrightarrow{g} p'} \qquad q_0' \xrightarrow{g} p'$$

$$p' \xrightarrow{\tau} p' I_i \mid t' I_1$$

$$t' \xrightarrow{g} q_0'$$

$$t \xrightarrow{g} t_1 \qquad \boxed{t' \xrightarrow{g} t_1}$$

$$t_1 I_i \xrightarrow{\tau} t_1$$

$$t_1 \xrightarrow{\tau} t_2$$

$$t_2 \xrightarrow{\tau} t_2 I_i \mid q_0$$

We note that the rules $tI^* \xrightarrow{g} q_0 I^*$, $t' I^* \xrightarrow{g} q_0 I^*$ (type $R_1 \longrightarrow R_2$) guarantee that Defender can still reach a syntactic equality in the case when Attacker chooses to take 'Defender's role' and generate a segment on the left-hand side, possibly different than that generated by Defender on the right-hand side.

Attacker is thus forced to let Defender generate the segments on both the right-hand side and the left-hand side. To remove any possibility that Defender could

gain by adding a different segment on the left-hand side than she has added on the right-hand side, we add the following rules.

(S-ind) rules:	$q_0 \xrightarrow{s_d} q_d$	$q_0' \xrightarrow{s_d} q_d$
(V-ind) rules:	$q_d I_i \xrightarrow{d_i} q_d$	(for all $i \in \{1, 2, \ldots, n\}$)
	$q_d \bot \xrightarrow{e} \varepsilon$	

Following the previous arguments, it is straightforward to verify that Defender has a WS from $(q_0 I_1 \bot, q_0' I_1 \bot)$ in the normed Type -2 system (G2), (S1), (S-ind), (V1), (V-ind) iff there is a (recurrent) solution of INST. The same holds for the PDA system (G2-τ), (S1-τ), (S-ind), (V1), (V-ind) (in the weak bisimulation game).

We observe that we really needed the rules $w \longrightarrow R$ for generating the segments; these are used also in (S2). In fact, we can confine ourselves to using just such rules in the whole system, by adding a new role to the special symbol $\bot$: we can implement erasing a (so far generated) sequence by 'killing' it with $\bot$, which disables any access to its 'right-hand side'; this means that the suffix following the leftmost occurrence of $\bot$ can be deemed nonexistent. Thus we 'pay' by unnormedness for using only $w \longrightarrow R$ rules.

The respective (G3) rules arise from (G2) by replacing the 'second half'; similarly for (G3-τ)-rules.

(G3) rules:	$q_0 \xrightarrow{g} t$	
	$\boxed{q_0 \xrightarrow{g} t' I_1 I^*}$	$q_0' \xrightarrow{g} t' I_1 I^*$
		$t' \xrightarrow{g} q_0'$
	$t \xrightarrow{g} q_0 I^* \bot$	$\boxed{t' \xrightarrow{g} q_0 I^* \bot}$
(G3-τ) rules:	$q_0 \xrightarrow{g} t$	
	$\boxed{q_0 \xrightarrow{g} p'}$	$q_0' \xrightarrow{g} p'$
		$p' \xrightarrow{\tau} p' I_i \,\vert\, t' I_1$
		$t' \xrightarrow{g} q_0'$
	$t \xrightarrow{g} t_2 \bot$	$\boxed{t' \xrightarrow{g} t_2 \bot}$
	$t_2 \xrightarrow{\tau} t_2 I_i \,\vert\, q_0$	

Verifying that Defender has a WS from $(q_0 I_1 \bot, q_0' I_1 \bot)$ in the system (G3), (S2), (S-ind), (V1), (V-ind) iff there is a (recurrent) solution of INST is again straightforward (using the fact that any sequence $q_0 I_{i_\ell} I_{i_{\ell-1}} \ldots I_{i_1} \bot w$ can be viewed as $q_0 I_{i_\ell} I_{i_{\ell-1}} \ldots I_{i_1} \bot$). Similarly for (G3-$\tau$), (S2-$\tau$), (S-ind), (V1), (V-ind).

Recalling Propositions 3.1(2) and 2.10, we get the following theorems.

THEOREM 5.3. *The bisimilarity problem is Σ_1^1-complete*

—*for both normed and unrestricted Type -2 systems ($R_1 \longrightarrow R_2$); shown by (G2, S1, S-ind, V1, V-ind),*

—*for unrestricted Type -1b systems ($w \longrightarrow R$); shown by (G3, S2, S-ind, V1, V-ind).*

THEOREM 5.4. *The weak bisimilarity problem is Σ_1^1-complete*

—*for both normed and unrestricted PDA (with ε-pushing and ε-popping); shown by (G2-τ, S1-τ, S-ind, V1, V-ind),*

—*for (unnormed) ε-pushing PDA; shown by (G3-τ, S2-τ, S-ind, V1, V-ind).*

6. PA PROCESSES

The class of PA processes is especially interesting from the point of view of implementing the rec-PCP abstract game. We can again naturally represent sequences but we can not use an (explicit) control unit to interpret them. Nevertheless, the required control unit is very restricted, and it can be modelled by a (modest) use of parallelism.

We again assume a fixed INST of the rec-PCP problem, a sequence $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ over the alphabet $\{A, B\}$, and construct an appropriate weak bisimulation game. The players will start with the pair

$$(q_0 V_1 \bot \| C,\ q_0' V_1 \bot \| C)$$

which is $((q_0 V_1 \bot) \| C,\ (q_0' V_1 \bot) \| C)$ since we use a convention that sequential composition binds more tightly than parallel. The (weak bisimulation) game is played according to the following PA-rules. Here i again ranges over $\{1, 2, \ldots, n\}$; and we use parentheses in $C \xrightarrow{z} (C \,\|\, q_w)$ to highlight the (only one) use of '$\|$' (not to be confused with the separation sign '$\,|\,$'). An explanation of the rules follows after their listing.

(G-PA) rules:

$$q_0 \xrightarrow{g} t \qquad \boxed{q_0 \xrightarrow{g} p'} \qquad\qquad q_0' \xrightarrow{g} p' \qquad p' \xrightarrow{\tau} p' V_i \,|\, t' V_1$$

$$t' \xrightarrow{g} q_0' \qquad \boxed{t' \xrightarrow{g} t_2 \bot}$$

$$t \xrightarrow{g} t_2 \bot \qquad t_2 \xrightarrow{\tau} t_2 V_i \,|\, q_0$$

(S-PA-I) rules:

$$q_0' \xrightarrow{s} q_c \qquad \boxed{q_0' \xrightarrow{s} r_1 \bot}$$

$$q_0 \xrightarrow{s} r_1 \bot \qquad r_1 \xrightarrow{\tau} r_1 U_i \,|\, r_2 \qquad r_2 \xrightarrow{\tau} r_2 A \,|\, r_2 B \,|\, q_c$$

(S-PA-II) rules:

$$q_c \xrightarrow{z} \varepsilon \qquad q_c \xrightarrow{\tau} \bot$$

$$C \xrightarrow{c_1} M_{a,b} \qquad C \xrightarrow{c_2} M_d \qquad C \xrightarrow{z} (C \,\|\, q_w)$$

$$q_w \xrightarrow{\tau} q_w U_i \,|\, q_w V_i \,|\, q_w A \,|\, q_w B \,|\, \varepsilon$$

(V-PA) rules:
$$U_i \xrightarrow{d_i} \varepsilon \qquad V_i \xrightarrow{d_i} \varepsilon$$
$$U_i \xrightarrow{\tau} (u_i)^R \qquad V_i \xrightarrow{\tau} (v_i)^R$$
$$A \xrightarrow{a} \varepsilon \quad A \xrightarrow{\tau} \varepsilon$$
$$B \xrightarrow{b} \varepsilon \quad B \xrightarrow{\tau} \varepsilon$$
$$M_d \xrightarrow{d_i} M_d$$
$$M_{a,b} \xrightarrow{a} M_{a,b} \quad M_{a,b} \xrightarrow{b} M_{a,b}$$

Remark 6.1. The processes $q_0V_1\bot\|C,\ q_0'V_1\bot\|C$ are obviously unnormed. A minor point is that there is no rewrite rule with $\bot$ at the left-hand side, though the definitions of PA processes sometimes require at least one rewrite rule for every process symbol. We could add a harmless rule $\bot \xrightarrow{f} \bot$ if we needed to handle this.

Let us now assume that INST has a (fixed recurrent) solution $i_1, i_2, i_3, \ldots$; we show a WS of Defender from the pair $(q_0V_1\bot\|C, q_0'V_1\bot\|C)$. A part of this WS will be to mimic every move of Attacker in the (derivatives α of the) "C-components"; we can thus concentrate on the components $q_0V_1\bot, q_0'V_1\bot$. We first note that (G-PA)-rules are, in fact, (G3-τ)-rules (the unnormedness, i.e., 'killing' by $\bot$ is important) where we just use V_i instead of I_i.

Remark. Unlike the case of PDAs, here we cannot use 'control states' q_u, q_v for interpreting I_i; therefore we use symbols V_i which carry the information about the indices and about the fact that they will be interpreted as v_i's. The left-hand side sequence will at the end contain U_i's but we start with generating two copies of a V_i-sequence for enabling DF (Defender's Forcing).

As long as Attacker uses (G-PA)-rules, Defender prolongs the so far generated sequences $V_{i_\ell}, V_{i_{\ell-1}}, \ldots, V_{i_1}$ (where $i_\ell = i_1 = 1$) by segments $V_{i_{\ell+m}}, V_{i_{\ell+m-1}}, \ldots, V_{i_{\ell+1}}$ where $i_{\ell+m} = 1$, following the assumed solution $i_1, i_2, i_3, \ldots$ of INST.

If Attacker wants to win, he must use (S-PA-I)-rules at some moment, in fact the rule $q_0' \xrightarrow{s} q_c$, for switching to the verification phase. At this moment, we have the (intermediate) pair

$$(\, q_0\, V_{i_\ell} V_{i_{\ell-1}} \ldots V_{i_1}\bot\|\alpha\,,\ q_c\, V_{i_\ell} V_{i_{\ell-1}} \ldots V_{i_1}\bot\|\alpha\,)$$

(omitting the suffix after $\bot$) and Defender now responds by a (long) transition $\stackrel{s}{\Longrightarrow}$ on the left-hand side. There she kills the V_i-sequence by $q_0 \xrightarrow{s} r_1\bot$ and by τ-rules she generates a U_i-sequence supplemented by a sequence $w \in \{A, B\}^*$. So she can install the pair

$$(\, q_c\, w\, U_{i_\ell} U_{i_{\ell-1}} \ldots U_{i_1}\bot\|\alpha\,,\ q_c\, V_{i_\ell} V_{i_{\ell-1}} \ldots V_{i_1}\bot\|\alpha\,)$$

where $w\,(u_{i_\ell})^R(u_{i_{\ell-1}})^R \ldots (u_{i_1})^R = (v_{i_\ell})^R(v_{i_{\ell-1}})^R \ldots (v_{i_1})^R$. Now if $\alpha = M_{a,b}\|\alpha'$ or $\alpha = M_d\|\alpha'$ for some α' then by inspecting the (V-PA)-rules it is clear that the two processes are weakly bisimilar. The only remaining chance for Attacker is that C appears in α and he plays $q_c \xrightarrow{z} \varepsilon$ on some side. Then Defender performs the killing $q_c \xrightarrow{\tau} \bot$ on the other side and by using $C \xrightarrow{z} C\|q_w$ and the τ-rules from q_w she installs the syntactic equality (when the suffixes behind $\bot$ are ignored). Hence Defender has a winning strategy in this case.

On the other hand, if INST has no (recurrent) solution, Attacker can use the following WS. He makes no moves from C and repeatedly asks Defender to prolong

the generated V_i-sequences (not caring whether Defender generated the same segments on both sides) until the right-hand side sequence does not represent a partial solution. This must eventually happen. Then Attacker uses $q'_0 \stackrel{s}{\longrightarrow} q_c$, thus forcing Defender to make an $\stackrel{s}{\Longrightarrow}$ move and install a pair

$$(q\, w\, U_{i'_{\ell'}} U_{i'_{\ell'-1}} \dots U_{i'_1} \bot \parallel C,\ q_c V_{i_\ell} V_{i_{\ell-1}} \dots V_{i_1} \bot \parallel C)$$

where $q \in \{r_1, r_2, q_c, \bot\}$. Since (the nonempty sequence) $i_1, i_2, \dots, i_\ell$ is not a partial solution, necessarily either the condition

$$\text{(eq-ind):} \quad \ell' = \ell \ \text{ and } \ i'_j = i_j \ \text{ for } j = 1, 2, \dots, \ell$$

or the condition

$$\text{(eq-word):} \quad w\,(u_{i'_{\ell'}})^R (u_{i'_{\ell'-1}})^R \dots (u_{i'_1})^R = (v_{i_\ell})^R (v_{i_{\ell-1}})^R \dots (v_{i_1})^R$$

is not satisfied.

In the case $q = q_c$ Attacker wins as follows: if (eq-ind) is not satisfied then Attacker plays $C \stackrel{c_1}{\longrightarrow} M_{a,b}$ and if (eq-word) is not satisfied then he plays $C \stackrel{c_2}{\longrightarrow} M_d$; now Attacker's win is clear (from the (V-PA)-rules). If $q = \bot$ then Attacker obviously wins, e.g., by performing $C \stackrel{c_1}{\longrightarrow} M_{a,b}$ and then $q_c \stackrel{z}{\longrightarrow} \varepsilon$ on the right-hand side. If $q \in \{r_1, r_2\}$ then Attacker changes q by one or two τ-moves into q_c; Defender can either make empty moves or the move $q_c \stackrel{\tau}{\longrightarrow} \bot$ on the right-hand side, which results in a winning situation for Attacker, as handled in the previous cases.

Recalling Propositions 3.1(2) and 2.10, we have thus shown the following theorem.

THEOREM 6.2. *Weak bisimilarity is Σ^1_1-complete for PA processes.*

7. PARALLEL PUSHDOWN AUTOMATA, PETRI NETS

7.1 Π^0_1-completeness

Reducing the Π^0_1-complete problem inf-MCM to the bisimulation game on PPDA processes appeared implicitly in [Jančar 1995b], where it was presented on Petri nets; [Moller 1996] then described the used Petri nets explicitly as PPDAs. We recall the (short) construction here because it is used (and enhanced) in our Σ^1_1-completeness result for weak bisimilarity.

Given a 2-counter MCM M with n instructions (as in Subsection 3.2), we construct a PPDA system in which $q_1 \sim q'_1$ iff the computation of M on $(0,0)$ is infinite. We use the pair of symbols q_i, q'_i for (the label of) the ith instruction, $i \in \{1, 2, \dots, n\}$, and symbols C_1, C_2 for unary representations of the values of counters c_1, c_2. We define the following rules corresponding to instructions of M (where action a can be read as 'addition', i.e., increment, d as 'decrement', z as 'zero', and h as 'halt').

'$i : c_r := c_r + 1$; goto j':

$$q_i \stackrel{a}{\longrightarrow} q_j C_r \qquad\qquad q'_i \stackrel{a}{\longrightarrow} q'_j C_r$$

'i : if $c_r = 0$ then goto j else ($c_r := c_r - 1$; goto k)':

$$q_i C_r \xrightarrow{d} q_k \qquad q_i' C_r \xrightarrow{d} q_k'$$
$$q_i \xrightarrow{z} q_j \qquad q_i' \xrightarrow{z} q_j'$$
$$\boxed{q_i C_r \xrightarrow{z} q_j' C_r} \qquad \boxed{q_i' C_r \xrightarrow{z} q_j C_r}$$

'n : HALT':

$$q_n \xrightarrow{h} \varepsilon$$

If M halts on $(0,0)$ then Attacker can force the correct simulation of M in both components of the starting pair (q_1, q_1') and finally win by playing $q_n \xrightarrow{h} \varepsilon$ (because there is no rule for q_n'). If the computation of M on $(0,0)$ is infinite then Defender forces Attacker to perform the correct simulation of the computation anyway (threatening by syntactic equality), and she thus wins.

This (and image finiteness) implies the following result [Jančar 1995b; Moller 1996].

THEOREM 7.1. *Bisimilarity is* Π_1^0*-complete for PPDA.*

7.2 Σ_1^1-completeness

We now consider a *nondeterministic* MCM M with two counters c_1, c_2 as an instance of the Σ_1^1-complete problem rec-NMCM. We want to construct a PPDA system where $q_1 \approx q_1'$ iff there is an infinite computation of M on $(0,0)$ which uses instruction 1 infinitely often.

We first make our modelling task easier by introducing a new nondeterministic instruction

$$i\text{: raise } c_r\text{; goto } j\,.$$

Its performing means adding a (nondeterministically chosen) nonnegative integer to c_r. We transform M into M' with 3 counters c_1, c_2, c_3 as follows. Every instruction $i : instr_i$ of M is replaced (and simulated) by two instructions in M', with labels $2i{-}1$ and $2i$.

$1 : instr_1$ is replaced by

> 1 : raise c_3; goto 2
> $2 : instr_1$

$i : instr_i$, for $i \in \{2, 3, \ldots, n\}$, is replaced by

> $2i{-}1$: if $c_3 = 0$ then goto $2n$ (halt) else ($c_3 := c_3 - 1$; goto $2i$)
> $2i : instr_i$

Inside $instr_i$, $i \in \{1, 2, \ldots, n\}$, each 'goto j' is replaced by 'goto $(2j{-}1)$'.

To satisfy our goal, it is obviously sufficient to construct a PPDA system in which $q_1 \approx q'_1$ iff there is an infinite computation of M' on $(0,0,0)$ (it necessarily performs instruction 1 infinitely often). The increment and zero testing instructions of M' are realized by the PPDA rules as in the previous subsection; as before, the (asymmetric) halting rule $q_{2n} \xrightarrow{h} \varepsilon$ is added.

The nondeterministic choice is to be resolved by Defender, so it is realized by the following variant of (G1).

'i : goto j or goto k':

$$q_i \xrightarrow{c} r_i \qquad\qquad q'_i \xrightarrow{c} p_j \,|\, p_k$$
$$\boxed{q_i \xrightarrow{c} p_j \,|\, p_k}$$

$$r_i \xrightarrow{a_j} q_j \qquad\qquad p_j \xrightarrow{a_j} q'_j$$
$$r_i \xrightarrow{a_k} q_k \qquad\qquad p_k \xrightarrow{a_k} q'_k$$
$$\boxed{p_j \xrightarrow{a_k} q_k}$$
$$\boxed{p_k \xrightarrow{a_j} q_j}$$

It thus remains to enable that Defender chooses the number to be added to c_3 when instruction 1 (of M') is performed; this is handled by the following variant of (G2-τ).

'1 : raise c_3; goto 2':

$$q_1 \xrightarrow{g} r_1$$
$$\boxed{q_1 \xrightarrow{g} p'_1} \qquad\qquad q'_1 \xrightarrow{g} p'_1$$
$$p'_1 \xrightarrow{\tau} p'_1 C_3 \,|\, r'_1$$

$$r'_1 \xrightarrow{g} q'_2$$
$$\boxed{r'_1 \xrightarrow{g} p_1}$$

$$r_1 \xrightarrow{g} p_1$$
$$p_1 C_3 \xrightarrow{\tau} p_1$$
$$p_1 \xrightarrow{\tau} p_1 C_3 \,|\, q_2$$

We note that the rule $p_1 C_3 \xrightarrow{\tau} p_1$ allows to decrease the number of C_3's, which is again added to enable DF; if Attacker would try to generate the left-hand side number of C_3's by himself, he would be punished by a syntactic equality.

The rules could be obviously modified to implement a general instruction 'i : raise c_r; goto j', by adding some rules (similar to (S-ind), (V-ind)) which would enable that Attacker wins if Defender does not add the same number of C_r's on both sides. In our special case we do not need this since Defender cannot gain by such 'cheating'.

If M' has an infinite computation (on $(0,0,0)$) then Defender can force that such a computation is simulated, and she wins. If there is no infinite computation of M'

on $(0, 0, 0)$, Attacker can eventually force reaching the control state q_{2n} on one side and q'_{2n} on the other side, and he wins by the asymmetric rule $q_{2n} \xrightarrow{h} \varepsilon$.

We have thus proved the following theorem.

THEOREM 7.2. *Weak bisimilarity is Σ_1^1-complete for PPDAs.*

8. SUMMARY

The following tables summarize the known results for the prefix-rewrite hierarchy (recall Figure 2) and pushdown automata.

(Strong) bisimilarity on prefix rewrite systems		
	Normed Processes	**Unnormed Processes**
Type -2	Σ_1^1-complete	Σ_1^1-complete
Type -1b	Π_1^0-complete	Σ_1^1-complete
Type -1a	Π_1^0-complete	Π_1^0-complete
Type 0 and Type $1\frac{1}{2}$	decidable [Stirling 1998] EXPTIME-hard [Kučera and Mayr 2002]	decidable [Senizergues 2005] EXPTIME-hard [Kučera and Mayr 2002]
Type 2	in P [Hirshfeld et al. 1996] P-hard [Balcazar et al. 1992]	in 2-EXPTIME [Burkart et al. 1995] PSPACE-hard [Srba 2002]
Type 3	in P [Paige and Tarjan 1987] [Kanellakis and Smolka 1990] P-hard [Balcazar et al. 1992]	in P [Paige and Tarjan 1987] [Kanellakis and Smolka 1990] P-hard [Balcazar et al. 1992]

Weak bisimilarity on pushdown automata		
	Normed Processes	**Unnormed Processes**
unrestricted	Σ_1^1-complete	Σ_1^1-complete
ε-pushing only	Π_1^0-complete	Σ_1^1-complete
ε-popping only	Π_1^0-complete	Π_1^0-complete

All undecidability results (Π_1^0-completess and Σ_1^1-completeness) have been proved in this paper. We can add that Sénizergues' decidability result applies to so called

regular graphs of finite out-degree. These can be characterized by Type -1a rules $R \xrightarrow{a} w$ with the restriction that R is a *prefix-free* regular language (as discussed in the report [Jančar and Srba 2006]). Hence, informally speaking, the decidability boundary lies between the prefix-free and the unrestricted rules $R \xrightarrow{a} w$. In other words, between deterministic ε-popping and unrestricted ε-popping in the case of weak bisimilarity for pushdown automata.

Our further results showed Σ_1^1-completeness of weak bisimilarity on PA processes as well as Σ_1^1-completeness of weak bisimilarity on PPDA processes (and Petri nets).

Finally, we would like to note that the decidability questions of weak bisimilarity are still open for BPA and BPP. The techniques developed in this article are not applicable to these classes because they seem to lack the possibility to simulate the control-state unit.

8.1 Acknowledgments

We would like to thank the anonymous referees for their useful remarks and suggestions.

REFERENCES

Baeten, J., Bergstra, J., and Klop, J. 1993. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM 40,* 3, 653–682.

Baeten, J. and Weijland, W. 1990. *Process Algebra.* Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Balcazar, J., Gabarro, J., and Santha, M. 1992. Deciding bisimilarity is P-complete. *Formal Aspects of Computing 4,* 6A, 638–648.

Bouajjani, A., Esparza, J., and Maler, O. 1997. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97).* LNCS, vol. 1243. Springer-Verlag, 135–150.

Büchi, J. 1964. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung 6*, 91–111.

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

Burkart, O., Caucal, D., and Steffen, B. 1995. An elementary decision procedure for arbitrary context-free processes. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95).* LNCS, vol. 969. Springer-Verlag, 423–433.

Burkart, O. and Esparza, J. 1997. More infinite results. *Bulletin of the European Association for Theoretical Computer Science 62*, 138–159. Columns: Concurrency.

Cachat, T. 2002. Uniform solution of parity games on prefix-recognizable graphs. *Electronic Notes in Theoretical Computer Science 68,* 6, 71–84.

Caucal, D. 1992. On the regular structure of prefix rewriting. *Theoretical Computer Science 106,* 1, 61–86.

Caucal, D. 1995. Bisimulation of context-free grammars and of pushdown automata. In *Modal Logic and Process Algebra.* CSLI Lectures Notes, vol. 53. University of Chicago Press, 85–106.

Caucal, D. 1996. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23th International Colloquium on Automata, Languages and Programming (ICALP'96).* Vol. 1099. Springer-Verlag, 194–205.

Christensen, S., Hüttel, H., and Stirling, C. 1995. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation 121*, 143–148.

Esparza, J., Hansel, D., Rossmanith, P., and Schwoon, S. 2000. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00).* LNCS, vol. 1855. Springer-Verlag, 232–247.

Harel, D. 1986. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *Journal of the ACM (JACM) 33,* 1, 224–248.

Hennessy, M. and Milner, R. 1985. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery 32,* 1, 137–161.

Hirshfeld, Y. and Jerrum, M. 1999. Bisimulation equivalence is decidable for normed process algebra. In *Proceedings of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*. LNCS, vol. 1644. Springer-Verlag, 412–421.

Hirshfeld, Y., Jerrum, M., and Moller, F. 1996. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science 158,* 1–2, 143–159.

Jančar, P. 1995a. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*. LNCS, vol. 915. Springer-Verlag, 349–363.

Jančar, P. 1995b. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science 148,* 2, 281–301.

Jančar, P. and Srba, J. 2006. Undecidability results for bisimilarity on prefix rewrite systems. Tech. Rep. RS-06-7, BRICS Research Series.

Kanellakis, P. and Smolka, S. 1990. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation 86,* 1, 43–68.

Kučera, A. and Jančar, P. 2006. Equivalence-checking on infinite-state systems: Techniques and results. *Theory and Practice of Logic Programming 6(3)*, 227–264.

Kučera, A. and Mayr, R. 2002. On the complexity of semantic equivalences for pushdown automata and BPA. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS'02)*. LNCS, vol. 2420. Springer-Verlag, 433–445.

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

Milner, R. 1989. *Communication and Concurrency*. Prentice-Hall.

Moller, F. 1996. Infinite results. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*. LNCS, vol. 1119. Springer-Verlag, 195–216.

Paige, R. and Tarjan, R. 1987. Three partition refinement algorithms. *SIAM Journal of Computing 16,* 6, 973–989.

Park, D. 1981. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*. LNCS, vol. 104. Springer-Verlag, 167–183.

Petri, C. 1962. Kommunikation mit automaten. Ph.D. thesis, Darmstadt.

Ruohonen, K. 1985. Reversible machines and post's correspondence problem for biprefix morphisms. *Elektronische Informationsverarbeitung und Kybernetik 21,* 12, 579–595.

Sénizergues, G. 1998. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS'98)*. IEEE Computer Society, 120–129.

Sénizergues, G. 2001. L(A)=L(B)? Decidability results from complete formal systems. *Theoretical Computer Science 251,* 1–2, 1–166.

Senizergues, G. 2005. The bisimulation problem for equational graphs of finite out-degree. *SIAM Journal on Computing 34,* 5, 1025–1106.

Sipser, M. 2005. *Introduction to the Theory of Computation, Second Edition*. Course Technology.

Srba, J. 2002. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*. LNCS, vol. 2380. Springer-Verlag, 716–727.

Srba, J. 2004. *Roadmap of Infinite results*. Vol. 2: Formal Models and Semantics. World Scientific Publishing Co. Updated version can be downloaded from the author's home-page.

Stirling, C. 1995. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*. LNCS, vol. 962. Springer-Verlag, 1–11.

Stirling, C. 1996. Decidability of bisimulation equivalence for normed pushdown processes. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*. LNCS, vol. 1119. Springer-Verlag, 217–232.

Stirling, C. 1998. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science 195,* 2, 113–131.

Stirling, C. 2000. Decidability of bisimulation equivalence for pushdown processes. Research Report EDI-INF-RR-0005, School of Informatics, Edinburgh University. Jan. The latest version is downloadable from the author's home-page.

Stirling, C. 2001. Decidability of DPDA equivalence. *Theoretical Computer Science 255,* 1–2, 1–31.

Stirling, C. 2003. Bisimulation and language equivalence. In *Logic for Concurrency and Synchronisation.* Trends in Logic, vol. 18. Kluwer Academic Publishers, 269–284.

Thomas, W. 1993. On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract). In *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93).* LNCS, vol. 668. Springer-Verlag, 559–568.

Received Month Year; revised Month Year; accepted Month Year