

On Extensions of Process Rewrite Systems: Rewrite Systems with Weak Finite-State Unit¹

Mojmír Křetínský² Vojtěch Řehák² Jan Strejček²

*Faculty of Informatics
Masaryk University
Brno, Czech Republic*

Abstract

Various classes of infinite-state processes are often specified by rewrite systems. We extend Mayr's Process Rewrite Systems (PRS) [13] by finite-state unit whose transition function satisfies some restrictions inspired by weak finite automata. We classify these models by their expressiveness and show how the hierarchy of new classes (w.r.t. bisimilarity) is related to both PRS hierarchy of Mayr and two other hierarchies of PRS extensions introduced in [9,22].

Key words: process rewrite systems, state extension, infinite state

1 Introduction

As the state-space infiniteness of concurrent systems has a various, real-life sources (e.g. data manipulation, asynchronous communication, etc.), a motivation is to provide adequate representations of concurrent systems as well as to study the possibilities of their formal verification. Concurrent systems can be modelled in a number of ways (e.g. Process Algebras, Petri Nets, etc.), however a unifying view is to interpret them as labelled transition systems (LTS) with possibly infinite number of states. LTS families are often specified via a variety of rewrite systems and form hierarchies (w.r.t. bisimulation equivalence), see for example [5,3,15,13]. Here we employ the classes of infinite-state systems defined by Process Rewrite Systems (PRS, introduced by Mayr in [13]) as they contain a variety of the formalisms studied in the context of formal verification. For surveys of formal verification techniques and results see for example [15,4,2,10,20].

It is possible to extend rewriting mechanisms by a finite-state unit [15,9]. However this extension is very powerful as the state-extended version of PA

¹ This work has been partially supported by GAČR, grant No. 201/03/1161.

² Emails: {kretinsky,rehak,strejcek}@fi.muni.cz

processes (i.e. the state-extended (1,G)-PRS) has a full Turing-power [1], while unrestricted PRS is not Turing-powerful [13]. One of motivations for introducing state-extended PRS classes can be seen as follows: it is required to specify some process classes which are not explicitly present in PRS hierarchy. This can be exemplified by the class of so called Parallel Push-Down Automata (PPDA) introduced by Moller as a state-extended version of BPP in [15]. Please note BPP forms a proper subclass of PPDA which is properly contained in Petri nets [16]. We also note the problem of bisimulation equivalence on BPP is decidable [6] (the recent results [21,8] show it is PSPACE-complete), while the same problem on their state-extended version is undecidable [15].

In this paper we aim at weakening the strength of state-extension by putting some restrictions on (the transition function of) finite-state unit – we use weak finite automaton as introduced in [17], but used here as a non-deterministic (NFA) rather than alternating one. A NFA $A = (Q, \Sigma, \delta, q_0, F)$ is *weak* if its state space is partitioned into a disjoint union $Q = \bigcup Q_i$, and there is a partial order \geq on the collection of the Q_i . The transition function $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is such that if $q \in Q_i$ and $q' \in \delta(q, a)$ then $q' \in Q_j$, where $Q_i \geq Q_j$. The set F of final states will not play any role in this paper, however recall it is required that $Q_i \subseteq F$ or $Q_i \cap F = \emptyset$ for each Q_i . Due to their connections to (modal) logics weak (nondeterministic, alternating,...) automata have been used in several contexts and in fact their slightly modified variants have been employed. For example we refer to [12] where the common fragment of a linear time logic LTL [18] and a branching time logic ACTL (which is a fragment of CTL [7]) is given by exactly those LTL formulae the negation of which can be represented by 1-weak Büchi automaton (automaton is 1-weak if each partition block contains exactly one state). We mention the 1-weak variant only and skip the others as this will serve as a suitable abstraction used in our definition of PRS with weak unit.

In state-extended PRS a finite-state unit keeps a sort of global information accessible to all parallel (ready to be reduced) components of a PRS term. For example different (sub)sets of rewriting rules can be applied depending on the current state of unit. Elsewhere [22] one of the authors of this paper enriched (pure) PRS by 'monotonically evolving' unit and showed the introduced fcPRS classes ('fc' standing for 'finitely constrained' – see Section 3) are strictly more expressible than the respective classes of PRS provided the respective PRS class does not subsume some notion of 'control state' as e.g. PDA or Petri Nets do.

Some basic definitions and properties of PRS (taken from [13]) and fcPRS ([22]) are recalled in Sections 2 and 3. We introduce PRS with weak unit (wPRS) and mention state-extended PRS in Section 4. Some relationships between the respective new classes and the already existing ones as well as the (combined) hierarchy of PRS and (relevant classes of) extended PRSs are shown in Section 5.

2 Process rewrite systems (PRS)

A *labelled transition system (LTS)* \mathcal{L} is a tuple $(S, Act, \longrightarrow, \alpha_0)$, where S is a set of *states* or *processes*, Act is a set of *atomic actions* or *labels*, $\longrightarrow \subseteq S \times Act \times S$ is a *transition relation* (written $\alpha \xrightarrow{a} \beta$ instead of $(\alpha, a, \beta) \in \longrightarrow$), $\alpha_0 \in S$ is a distinguished *initial state*. A state $\alpha \in S$ is *terminal* (or *deadlocked*, written $\alpha \not\rightarrow$) if there is no $a \in Act$ and $\beta \in S$ such that $\alpha \xrightarrow{a} \beta$. We also use the natural generalization $\alpha \xrightarrow{\sigma} \beta$ for finite sequences of actions $\sigma \in Act^*$. The state α is *reachable* if there is $\sigma \in Act^*$ such that $\alpha_0 \xrightarrow{\sigma} \alpha$.

A binary relation R on set of states S is a *bisimulation* [14] iff for each $(\alpha, \beta) \in R$ the following conditions hold.

- $\forall \alpha' \in S, a \in Act : \alpha \xrightarrow{a} \alpha' \implies (\exists \beta' \in S : \beta \xrightarrow{a} \beta' \wedge (\alpha', \beta') \in R)$
- $\forall \beta' \in S, a \in Act : \beta \xrightarrow{a} \beta' \implies (\exists \alpha' \in S : \alpha \xrightarrow{a} \alpha' \wedge (\alpha', \beta') \in R)$

Bisimulation equivalence on an assumed LTS is the maximal bisimulation (i.e. union of all bisimulations).

Let $Const = \{X, \dots\}$ be a countably infinite set of *process constants*. The set \mathcal{T} of *process terms* (ranged over by t, \dots) is defined by the abstract syntax

$$t = \varepsilon \mid X \mid t_1.t_2 \mid t_1 \parallel t_2,$$

where ε is the empty term, $X \in Const$ is a process constant (used as an atomic process), ' \parallel ' and '.' mean parallel and sequential compositions respectively. The set $Const(t)$ is the set of all constants occurring in a process term t . We always work with equivalence classes of terms modulo commutativity and associativity of ' \parallel ' and modulo associativity of '.' We also define $\varepsilon.t = t = t.\varepsilon$ and $t \parallel \varepsilon = t$.

We distinguish four *classes of process terms*: '1' stands for terms consisting of a single process constant only (i.e. $\varepsilon \notin \mathbf{1}$), 'S' are *sequential* terms – without parallel composition, 'P' are *parallel* terms - without sequential composition, 'G' are *general* terms with arbitrarily nested sequential and parallel compositions.

Definition 2.1 Let $Act = \{a, b, \dots\}$ be a countably infinite set of *atomic actions*, $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An (α, β) -PRS (*process rewrite system*) is a pair $\Delta = (R, t_0)$, where

- R is a finite set of *rewrite rules* of the form $t_1 \xrightarrow{a} t_2$, where $t_1 \in \alpha$, $t_1 \neq \varepsilon$, $t_2 \in \beta$ are process terms and $a \in Act$ is an atomic action,
- $t_0 \in \beta$ is an *initial state*.

Unless stated otherwise we assume a given Δ as in Definition 2.1. We define $Const(\Delta)$ as the set of all constants occurring in the rewrite rules of Δ or in its initial state, and $Act(\Delta)$ as the set of all actions occurring in the rewrite rules of Δ . We sometimes write $(t_1 \xrightarrow{a} t_2) \in \Delta$ instead of $(t_1 \xrightarrow{a} t_2) \in R$.

The semantics of Δ is given by the LTS $(S, Act(\Delta), \longrightarrow, t_0)$, where $S = \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$ and \longrightarrow is the least relation satisfying the

inference rules:

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2}, \quad \frac{t_1 \xrightarrow{a} t'_1}{t_1 \| t_2 \xrightarrow{a} t'_1 \| t_2}, \quad \frac{t_1 \xrightarrow{a} t'_1}{t_1.t_2 \xrightarrow{a} t'_1.t_2}.$$

If no confusion can arise, we sometimes speak about “process rewrite system” meaning “labelled transition system generated by process rewrite system”.

The *PRS-hierarchy* of (α, β) -PRS is depicted as a sub-hierarchy in Figure 1. Some classes included in the hierarchy correspond to widely known models as Finite State systems (FS), Basic Process Algebras (BPA), Basic Parallel Processes (BPP), Process Algebras (PA), Push-Down Processes (PDA, see [5] for justification), and Petri Nets (PN). The other three classes were introduced (and named) by Mayr [13]. The relationship between the class name and its (α, β) -PRS counter-part is given in Figure 1 as well.

PRS-hierarchy is not strict w.r.t. language equivalence (e.g. both BPA and PDA define the class of ε -free context-free languages). The strictness of the hierarchy w.r.t. bisimilarity follows from the results presented (or cited) in [3,15] and from the following two examples [13].

Example 2.2 A PDA system with the initial state $U.X$ which is not bisimilar to any PAN system (for proof see [13]).

$$\begin{array}{lll} U.X \xrightarrow{a} U.A.X & U.A \xrightarrow{a} U.A.A & U.B \xrightarrow{a} U.A.B \\ U.X \xrightarrow{b} U.B.X & U.A \xrightarrow{b} U.B.A & U.B \xrightarrow{b} U.B.B \\ U.X \xrightarrow{c} V.X & U.A \xrightarrow{c} V.A & U.B \xrightarrow{c} V.B \\ U.X \xrightarrow{d} W.X & U.A \xrightarrow{d} W.A & U.B \xrightarrow{d} W.B \\ V.X \xrightarrow{e} V & V.A \xrightarrow{a} V & V.B \xrightarrow{b} V \\ W.X \xrightarrow{f} W & W.A \xrightarrow{a} W & W.B \xrightarrow{b} W \end{array}$$

Example 2.3 A Petri net given as (P, P) -PRS with the initial state $X \| A \| B$ which is not bisimilar to any PAD process (for proof see [13]).

$$\begin{array}{llll} X \xrightarrow{g} X \| A \| B & Y \| A \xrightarrow{a} Y & X \| A \xrightarrow{d} Z & Y \| A \xrightarrow{d} Z \\ X \xrightarrow{c} Y & Y \| B \xrightarrow{b} Y & X \| B \xrightarrow{d} Z & Y \| B \xrightarrow{d} Z \end{array}$$

3 PRS with finite constraint systems (fcPRS)

In this section we recall the extension of process rewrite systems with finite constraint systems. This extension has been directly motivated by constraint systems used in *concurrent constraint programming* (CCP) – see e.g. [19]. A *constraint system* is a bounded lattice $(C, \vdash, \wedge, tt, ff)$, where C is the set of *constraints*, \vdash (called *entailment*) is an ordering on this set, \wedge is the lub operation, and tt (*true*), ff (*false*) are the least and the greatest elements of C respectively ($ff \vdash tt$ and $tt \neq ff$). The constraint system describes a state

space and possible evolution of a unit called *store*.

Definition 3.1 Let $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An (α, β) -*fcPRS* (*PRS with finite constraint system*) is a tuple $\Delta = (\mathcal{C}, R, t_0)$, where

- $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$ is a finite constraint system describing the *store*; the elements of C represent *values of the store*,
- R is a finite set of *rewrite rules* of the form $(t_1 \xrightarrow{a} t_2, m, n)$, where $t_1 \in \alpha$, $t_1 \neq \varepsilon$, $t_2 \in \beta$ are process terms, $a \in Act$ is an atomic action, and $m, n \in C$ are constraints,
- $t_0 \in \beta$ is a distinguished *initial process term*.

The semantics of an (α, β) -*fcPRS* system $\Delta = (\mathcal{C}, R, t_0)$ is given by the LTS $(S, Act(\Delta), \longrightarrow, (t_0, tt))$, where $S = \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\} \times (C \setminus \{ff\})$ and \longrightarrow is the least relation satisfying the inference rules:

$$\frac{(t_1 \xrightarrow{a} t_2, m, n) \in \Delta}{(t_1, o) \xrightarrow{a} (t_2, o \wedge n)} \quad \text{if } o \vdash m \text{ and } o \wedge n \neq ff,$$

$$\frac{(t_1, o) \xrightarrow{a} (t'_1, p)}{(t_1 \parallel t_2, o) \xrightarrow{a} (t'_1 \parallel t_2, p)}, \quad \frac{(t_1, o) \xrightarrow{a} (t'_1, p)}{(t_1.t_2, o) \xrightarrow{a} (t'_1.t_2, p)}.$$

The two side conditions in the first inference rule are very close to principles used in CCP. The first one ($o \vdash m$) ensures the rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ can be used only if the current value of the store o *entails* m (it is similar to $ask(m)$ in CCP). The second condition ($o \wedge n \neq ff$) guarantees that the store stays *consistent* after application of the rule (analogous to the consistency requirement when processing $tell(n)$ in CCP).

An important observation is that the value of a store can move in a lattice only upwards as $o \wedge n$ always entails o . Intuitively, partial information can only be added to the store, but never retracted (the store is *monotonic*).

Also note that an execution of a transition which starts in a state with o on the store and which is generated by a rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ implies that for every subsequent value of the store p the conditions $p \vdash m$ and $p \wedge n \neq ff$ are satisfied (and thus the use of the rule cannot be forbidden by a value of the store in future). The first condition $p \vdash m$ comes from the monotonic behaviour of the store. The second condition comes from the facts that the constraint n in the rule can change the store only in the first application of the rule and that for each subsequent state p of the store $p \wedge n = p$ holds.

4 PRS with weak finite-state unit (wPRS)

Definition 4.1 Let $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An (α, β) -*wPRS* (*PRS with weak finite-state unit*) is a tuple $\Delta = (Q, \geq, R, m_0, t_0)$, where

- (Q, \geq) is partially ordered finite set representing states of *weak finite-state unit*; the elements of Q are called *w-states*,

- R is a finite set of *rewrite rules* of the form $(mt_1) \xrightarrow{a} (nt_2)$ satisfying the condition $m \geq n$, where $m, n \in Q$, $t_1 \in \alpha$, $t_1 \neq \varepsilon$, $t_2 \in \beta$, and $a \in Act$,
- $m_0 \in Q$, $t_0 \in \beta$, and $m_0 t_0$ is the *initial state* of the system.

The semantics of an (α, β) -wPRS $\Delta = (Q, \geq, R, m_0, t_0)$ is given by the LTS $(S, Act(\Delta), \longrightarrow, m_0 t_0)$, where $S = \{mt \mid m \in Q, t \in \beta, Const(t) \subseteq Const(\Delta)\}$ and \longrightarrow is defined as the least relation satisfying the inference rules:

$$\frac{(mt_1 \xrightarrow{a} nt_2) \in \Delta}{mt_1 \xrightarrow{a} nt_2}, \quad \frac{mt_1 \xrightarrow{a} nt'_1}{m(t_1 \parallel t_2) \xrightarrow{a} n(t'_1 \parallel t_2)}, \quad \frac{mt_1 \xrightarrow{a} nt'_1}{m(t_1.t_2) \xrightarrow{a} n(t'_1.t_2)}.$$

The presented notion of weakness corresponds to 1-weakness condition in automata theory (mentioned already in Section 1; the general weak unit without considering final states would coincide with standard state-extension as all the states of Q could be included in one partition block). In any transition sequence the w-state components of visited states form a non-increasing sequence w.r.t. \geq (i.e. can only change finitely many times). However, in contrast to fcPRS, the weak unit can forbid the application of any rewrite rule.

State Extended PRS If we relax from the condition $m \geq n$ imposed on rewrite rules in Definition 4.1, we get the definition of state-extended (α, β) -PRS (denoted by prefix 'se'). Instead of $(1, S)$ -sePRS, $(1, P)$ -sePRS, ... we also use more traditional abbreviations seBPA, seBPP, ... and we also take up this notation for all the classes of both fcPRS and wPRS introduced earlier.

We remind a motivation of introducing sePRS given in Section 1. Of course, seBPA and seBPP coincides with PDA and PPDA respectively. Also recall that (S,S)-PRS and PDA are equivalent w.r.t. bisimilarity as shown by Caucal [5], while seBPP has no bisimulation equivalent counter-part within PRS hierarchy (it is strictly under (P,P)-PRS as shown by Moller in [16]).

5 Relations between classes, refining hierarchy

We start with some very obvious observations. Given process classes X, Y the notation $X \subseteq Y$ means that every LTS definable in class X can be defined (up to bisimulation equivalence) also in class Y. We say Y is at least as expressive as X .

An immediate observation is the classes FS, PDA and PN have the same expressiveness as the corresponding {fc, w, se} extended classes. We also note that

$$(\alpha, \beta)\text{-PRS} \subseteq (\alpha, \beta)\text{-fcPRS} \subseteq (\alpha, \beta)\text{-wPRS} \subseteq (\alpha, \beta)\text{-sePRS}$$

hold for every (α, β) -PRS class (even up to isomorphism). For the second inclusion take an arbitrary (α, β) -fcPRS Δ with an initial term t_0 and a constraint system $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$. The corresponding (α, β) -wPRS is $\Delta' = (C \setminus \{ff\}, \geq, R', tt, t_0)$, where $\geq = (\vdash)^{-1}$ and $R' = \{ot_1 \xrightarrow{a} (o \wedge n)t_2 \mid (t_1 \xrightarrow{a}$

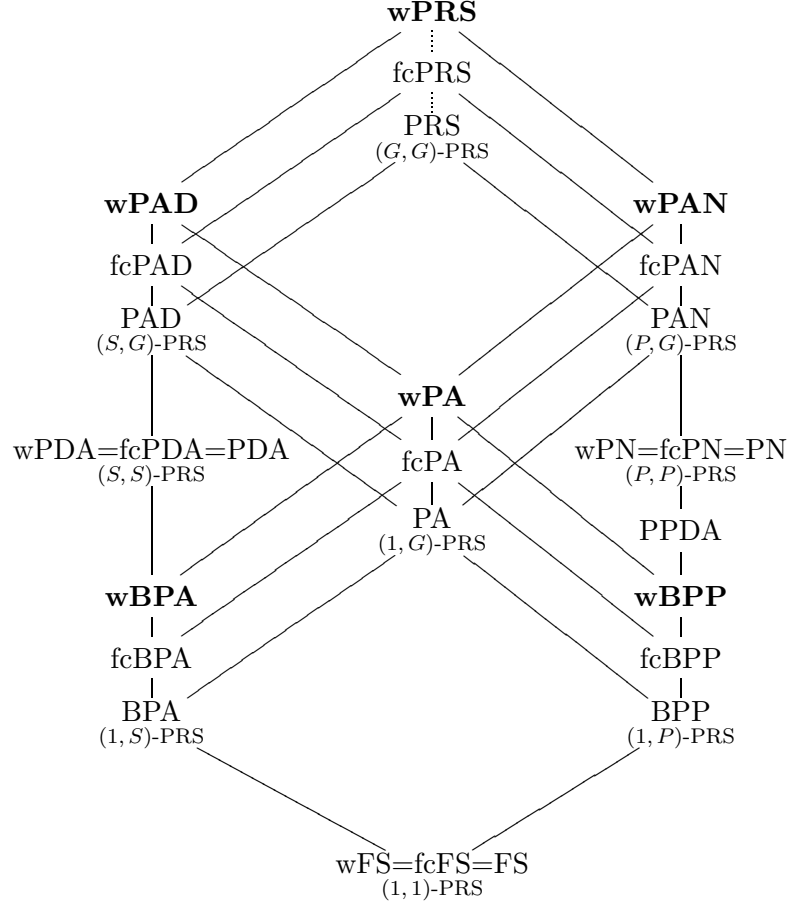


Fig. 1. The hierarchy of classes defined by (extended) rewrite formalisms

$t_2, m, n) \in \Delta$ and $o \vdash m$ and $o \wedge n \neq ff$. The first and third inclusions are obvious as well.

Relations between the classes of PRS-hierarchy, the corresponding classes extended with finite constraint systems or weak state unit, and the PPDA class (the only class of Moller's hierarchy not covered by previous formalisms) are depicted in Figure 1. The shape of the hierarchy follows from the definitions of included classes and from the relations between considered extensions. The strictness of the PRS-hierarchy (w.r.t. bisimulation equivalence) has been proved by Mayr [13]. The strictness of the hierarchy covering classes from PRS-hierarchy and corresponding classes with finite constraint systems has been proved in [22] (with one exception – the strictness between PRS and fcPRS is just a conjecture).

In the rest of this section we show that

- there is a PDA system which is not bisimilar to any wPAN system,
- there is a PPDA system which is not bisimilar to any wPAD system,
- there is a wBPP system which is not bisimilar to any fcPAD system (this proof formulates a property that is a sufficient condition for a PAD to be

bisimilar to some PDA),

- there is a wBPA system which is not bisimilar to any fcPAN system.

These proofs together with the fact that PPDA are strictly less expressive than PN [16] will finish the proof of the strictness of our hierarchy (with two exceptions – the strictness of the relations between PRS, fcPRS, and wPRS remains unproved, although we conjecture the existence of the separating gaps).

5.1 PDA non-bisimilar to wPAN

Example 5.1 Let us consider a PDA system of Example 2.2 but having $U.X.Y$ as the initial state and two more rewrite rules: $V.Y \xrightarrow{x} U.X.Y$, $W.Y \xrightarrow{x} U.X.Y$. This system, denoted by Δ_1 , behaves like that defined in Example 2.2, but whenever the original system terminates, the enhanced Δ_1 is restarted under the action x .

Lemma 5.2 *There is no wPAN Δ bisimilar to the PDA Δ_1 of Example 5.1.*

Proof. To derive a contradiction assume a wPAN Δ bisimilar to the PDA Δ_1 . As the weak state unit of Δ is finite then there exists a reachable state mt of Δ such that every state reachable from mt has also m as its w-state component (the opposite would imply the infiniteness of the weak state unit). There exists a word $w \in \{a, b, c, d, e, f\}^*$ such that $mt \xrightarrow{w.x} mt'$, where mt' is bisimilar to the state $U.X.Y$ of the PDA process Δ_1 . If the rules labelled by the action x are removed from Δ and mt' is taken as the initial state, we obtain the system whose all reachable states have m as w-state component and which is bisimilar to the pushdown process of Example 2.2.

Now let Δ' be a PAN system with the initial state t' and with the set of rewrite rules consisting of the rules $l \xrightarrow{v} r$, where $(ml \xrightarrow{v} mr) \in \Delta$ and $v \in \{a, b, c, d, e, f\}$. It is obvious that this PAN system Δ' is bisimilar to the PDA system defined in Example 2.2 – a contradiction. \square

5.2 PPDA non-bisimilar to wPAD

Example 5.3 Let Δ be a PPDA process with the initial state $xA\|B\|C$ and the following rewrite rules:

$$\begin{array}{lll}
 xC \xrightarrow{g} xA\|B\|C & xA \xrightarrow{d} z\varepsilon & zA \xrightarrow{q} z\varepsilon \\
 xC \xrightarrow{c} yC & xB \xrightarrow{d} z\varepsilon & zB \xrightarrow{q} z\varepsilon \\
 yA \xrightarrow{a} y\varepsilon & yA \xrightarrow{d} z\varepsilon & zC \xrightarrow{r} xA\|B\|C \\
 yB \xrightarrow{b} y\varepsilon & yB \xrightarrow{d} z\varepsilon &
 \end{array}$$

The rules labelled by g, c, a, b, d correspond to the rules of the Petri net given in Example 2.3. Hence the PPDA Δ behaves as the mentioned Petri net, but when the Petri net terminates, the PPDA Δ can remove an arbitrary

number of A and B symbols from the parallel stack and then “restart” the system under action r .

Lemma 5.4 *There is no wPAD Δ' bisimilar to the PPDA Δ of Example 5.3.*

The proof is similar to the proof of Lemma 5.2 using the fact the Petri net of Example 2.3 is not bisimilar to any PAD system.

5.3 wBPP non-bisimilar to fcPAD

A rewriting system is *deadlockable* if for each reachable nonterminal state s (of its underlying LTS) there is a transition from s to a terminal state, i.e. $\exists a, t : s \xrightarrow{a} t \not\rightarrow$.

Definition 5.5 A sequential subterm t (i.e. $t \in S$) of term $g \in G$ is a *ready parallel component* iff t is a maximal subtree in the syntax tree of term g such that t is not in the right-hand side subtree of any sequential node (i.e. node corresponding to sequential operator). A ready parallel component t is *live* in a PAD system Δ if t is not deadlocked (i.e. there is a rule applicable to t).

Intuitively the ready parallel components are defined as the maximal sequential parts of a PAD process g such that g can perform an action a if and only if some of its ready parallel components can perform the same action a .

Lemma 5.6 *Every reachable state of an arbitrary deadlockable PAD system has at most one live ready parallel component.*

Proof. Observe that the application of a PAD rewrite rule can only modify one ready parallel component. Hence there is no way how to deadlock more than one live ready parallel component by one application of a PAD rewrite rule. \square

Lemma 5.7 *Every deadlockable PAD system is bisimilar to a PDA system.*

Proof. An idea is to transform PAD rewrite rules onto corresponding PDA rewrite rules (this is sufficient as for every PAD there is a bisimilar PAD system with a single process constant as the initial process term).

There is only one way to revive a deadlocked parallel component, namely to rewrite adjacent components onto ε . For example if $B.C$ is a deadlocked ready parallel component of $(A.C\|B.C).D$ and $(A.C\|B.C).D \xrightarrow{w} (\varepsilon\|B.C).D = B.C.D$ then the ready parallel component $B.C.D$ can be live.

Let Δ be a PAD system and $X \notin \text{Const}(\Delta)$ be a fresh process constant. Let us consider a rewrite rule of Δ with the right hand side containing a maximal subterm of the form $l.(t_1\|t_2).r$, where $t_1, t_2 \in S$ and l, r can be ε . In an arbitrary transition sequence the components t_1, t_2 generated by the application of the considered rewrite rule become ready at the same time. Thus at least one of them is deadlocked. Let t_2 be deadlocked. We replace the subterm $l.(t_1\|t_2).r$ of the rule by $l.X.t_1.X.t_2.r$ (or just $t_1.X.t_2.r$ whenever l is

ε). The process constant X eliminates any possible (unwanted) interaction of (the tail of) the term l and (the beginning of) the term t_1 (or the tail of t_1 and the beginning of t_2 respectively). Repeating this procedure eliminates all parallel operators from rewrite rules. The resulting PDA system Δ' enriched by rewrite rules of the form $X.s \xrightarrow{a} s'$ for every rule $s \xrightarrow{a} s' \in \Delta'$ is bisimilar to a given Δ . \square

Example 5.8 Let Δ_2 be the wBPP system with the initial state pX and the rules:

$$pX \xrightarrow{c} pX \| A \| B \quad pA \xrightarrow{a} p\varepsilon \quad pB \xrightarrow{b} p\varepsilon \quad pX \xrightarrow{d} q\varepsilon$$

Lemma 5.9 *There is no PAD system bisimilar to the wBPP system Δ_2 of Example 5.8.*

Proof. Δ_2 is deadlockable. Due to Lemma 5.7 it suffices to prove there is no PDA system bisimilar to Δ_2 . This directly follows from the fact that the language L generated by Δ_2 is not context-free ($L \cap c^*a^*b^*d = \{c^k a^l b^m d \mid 0 \leq l, m \leq k\}$ is not a context-free language). \square

Lemma 5.10 *There is no fcPAD system bisimilar to the wBPP system Δ_2 of Example 5.8.*

Proof. For the sake of a contradiction we assume a fcPAD Δ bisimilar to Δ_2 .

The finiteness of the constraint system used in Δ implies that there exists a reachable non-terminal state (t, m) of Δ such that every non-terminal state reachable from (t, m) has also m on the store (the contrary would mean the constraint system is infinite). As (t, m) is non-terminal there exists a word $w \in \{a, b\}^*$ such that $(t, m) \xrightarrow{w} (s, m)$ and (s, m) is bisimilar to the initial state pX of Δ_2 . The only transitions starting at states reachable from (s, m) and changing the value of the store can be the transitions leading to terminal states, i.e. the transitions labelled by d . Hence we can directly assume that all rewrite rules of Δ labelled with $x \in \{a, b, c\}$ have the form $(t_1 \xrightarrow{x} t_2, tt, tt)$.

Let Δ' be a PAD system with the set of rewrite rules as

$$\begin{aligned} & \{t_1 \xrightarrow{x} t_2 \mid (t_1 \xrightarrow{x} t_2, tt, tt) \in \Delta, x \neq d\} \cup \\ & \cup \{t_1 \xrightarrow{d} Z \mid (t_1 \xrightarrow{d} t_2, tt, n) \in \Delta, n \neq ff\}, \end{aligned}$$

where $Z \notin \text{Const}(\Delta)$ is a fresh process constant. If we restrict the systems Δ and Δ' to actions a, b, c then Δ and Δ' are bisimilar. Furthermore in every state q of Δ' reachable under $w \in \{a, b, c\}^*$ there is a transition labelled by d and starting at q . It suffices to show that this transition is leading to a terminal state.

The state (q, tt) (corresponding to the state q) has a ready parallel component able to perform an action c . This action cannot be disabled by any action performed by another ready parallel component. Hence there is just one ready parallel component able to perform both c and d . For the same reason this component is the only one which is able to perform actions a and b if they are

enabled in the state (q, tt) . The same holds for the state q of Δ' . Moreover the ready parallel component rewritten by the action d is deadlocked by the process constant Z . Thus the state reached under d is terminal and we get a PAD system Δ' bisimilar to the wBPP Δ_2 of Example 5.8 – a contradiction (see Lemma 5.9). \square

5.4 wBPA non-bisimilar to fcPAN

Example 5.11 Let us consider the following wBPA system with initial state pX .

$$\begin{array}{cccc} pX \xrightarrow{a} pAX & pX \xrightarrow{b} pBX & pA \xrightarrow{\bar{a}} p\varepsilon & pB \xrightarrow{\bar{b}} p\varepsilon \\ pA \xrightarrow{a} pAA & pA \xrightarrow{b} pBA & pA \xrightarrow{a'} q\varepsilon & pB \xrightarrow{b'} q\varepsilon \\ pB \xrightarrow{a} pAB & pB \xrightarrow{b} pBB & qA \xrightarrow{a'} q\varepsilon & qB \xrightarrow{b'} q\varepsilon \end{array}$$

Lemma 5.12 *There is no fcPAN system bisimilar to the wBPA system of Example 5.11.*

The proof employs the notion of ready sequential components (an analogue of ready parallel components introduced in Definition 5.5). As the proof is much more technically involved we skip it. The full version of the proof is published in [11].

6 Conclusion and future work

We have extended Process Rewrite Systems (PRS) [13] by 'weak' finite-state unit and have classified new classes by their expressiveness. We have shown the refined hierarchy (w.r.t. bisimilarity) containing new classes as well as those generated by both PRS and of two other PRS extensions introduced in [9,22].

We emphasize the results showing that BPP class and its three extensions form a strict (sub)hierarchy w.r.t. bisimulation,

$$\text{BPP} \subsetneq \text{fcBPP} \subsetneq \text{wBPP} \subsetneq \text{seBPP} \subsetneq \text{PN}$$

which is decidable (even PSPACE-complete) on the BPP class and undecidable on the class of state-extended BPP (i.e. PPDA). It remains open for other two classes (i.e. fcBPP and wBPP) and is a subject of our further research. We are motivated by the fact the strictness of two leftmost inclusions can be proved (but is not shown here) even for language equivalence. The strictness of inclusion between wBPP and seBPP on the language equivalence level is just our conjecture.

References

- [1] Bouajjani, A., R. Echahed and P. Habermehl, *On the verification problem of nonregular properties for nonregular processes*, in: *Proc. of LICS'95* (1995).
- [2] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: *Handbook of Process Algebra* (2001), pp. 545–623.
- [3] Burkart, O., D. Caucal and B. Steffen, *Bisimulation collapse and the process taxonomy*, in: *Proc. of CONCUR'96*, LNCS **1119** (1996), pp. 247–262.
- [4] Burkart, O. and J. Esparza, *More infinite results*, *Electronic Notes in Theoretical Computer Science* **5** (1997).
- [5] Caucal, D., *On the regular structure of prefix rewriting*, *Theoretical Computer Science* **106** (1992), pp. 61–86.
- [6] Christensen, S., Y. Hirshfeld and F. Moller, *Bisimulation is decidable for all basic parallel processes*, in: *Proceedings of CONCUR'93*, LNCS **715** (1993), pp. 143–157.
- [7] Clarke, E. M. and E. A. Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*, in: *Proc. IBM Workshop on Logic of Programs*, LNCS **131** (1981), pp. 52–71.
- [8] Jančar, P., *Strong bisimilarity on basic parallel processes is PSPACE-complete*, in: *Proc. of 18th IEEE Symposium on Logic in Computer Science (LICS'03)* (2003), pp. 218–227.
- [9] Jančar, P., A. Kučera and R. Mayr, *Deciding bisimulation-like equivalences with finite-state processes*, *Theoretical Computer Science* **258** (2001), pp. 409–433.
- [10] Kučera, A. and P. Jančar, *Equivalence-checking with infinite-state systems: Techniques and results*, in: *Proc. SOFSEM'2002*, LNCS **2540** (2002).
- [11] Křetínský, M., V. Řehák and J. Strejček, *Process Rewrite Systems with Weak Finite-State Unit*, Technical Report FIMU-RS-2003-05, Faculty of Informatics, Masaryk University Brno (2003), full version of this paper.
- [12] Maidl, M., *The common fragment of CTL and LTL*, in: *Proc. 41th Annual Symposium on Foundations of Computer Science*, 2000, pp. 643–652.
- [13] Mayr, R., *Process rewrite systems*, *Information and Computation* **156** (2000), pp. 264–286.
- [14] Milner, R., “Communication and Concurrency,” Prentice-Hall, 1989.
- [15] Moller, F., *Infinite results*, in: *Proc. of CONCUR'96*, LNCS **1119** (1996), pp. 195–216.
- [16] Moller, F., *Pushdown Automata, Multiset Automata and Petri Nets, MFCS Workshop on concurrency*, *Electronic Notes in Theoretical Computer Science* **18** (1998).

- [17] Muller, D., A. Saoudi and P. Schupp, *Alternating automata, the weak monadic theory of trees and its complexity*, Theoret. Computer Science **97** (1992), pp. 233–244.
- [18] Pnueli, A., *The temporal logic of programs*, in: *Proc. 18th IEEE Symposium on the Foundations of Computer Science*, 1977, pp. 46–57.
- [19] Saraswat, V. A. and M. Rinard, *Concurrent constraint programming*, in: *Proc. of 17th POPL* (1990), pp. 232–245.
- [20] Srba, J., *Roadmap of infinite results*, EATCS Bulletin (2002), pp. 163–175.
- [21] Srba, J., *Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard*, in: *Proc. STACS 2002*, LNCS **2285** (2002), pp. 535–546.
- [22] Strejček, J., *Rewrite systems with constraints*, *EXPRESS'01*, Electronic Notes in Theoretical Computer Science **52** (2002).