On the Expressive Power of Extended Process Rewrite Systems

Mojmír Křetínský
Vojtěch Řehák
Jan Strejček

BRICS Report Series RS-04-7

ISSN 0909-0878 April 2004

On the Expressive Power of Extended Process Rewrite Systems[*]

M. Křetínský, V. Řehák[†] and J. Strejček

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
{kretinsky,rehak,strejcek}@fi.muni.cz

April, 2004

Abstract

We provide a unified view on three extensions of Process rewrite systems (PRS) and compare their and PRS's expressive power. We show that the class of Petri Nets is less expressible up to bisimulation than the class of Process Algebra extended with finite state control unit. Further we show our main result that the reachability problem for PRS extended with a so called weak finite state unit is decidable.

1 Introduction

An automatic verification of current software systems often needs to model them as infinite-state systems, i.e. systems with an evolving structure and operating on unbounded data types: a network of mobile phones is a concurrent system with evolving structure which dynamically changes its size (and can become very large). Robustness of the network requires that underlying protocols should work for an arbitrarily large

[*]This work has been supported by GAČR, grant No. 201/03/1161.

[†]The co-author is supported by Marie Curie Fellowship of the European Community Programme Improving the Human Research Potential and the Socio-economic Knowledge Base under contract number HPMT-CT-2000-00093.

(i.e. potentially infinite) number of client processes. A JAVA applet dynamically downloads classes over the network and executes their methods, the stack of activation records should be seen as potentially infinite.

Infinite-state systems can be specified in a number of ways with their respective advantages and limitations. Petri nets, pushdown automata, and process algebras like BPA, BPP, or PA all serve to exemplify this. Here we employ the classes of infinite-state systems defined by term rewrite systems and called PRS (*Process Rewrite Systems*) as introduced by Mayr [12]. PRS subsume a variety of the formalisms studied in the context of formal verification (e.g. all the models mentioned above).

A Process Rewrite System is a finite set of rules $t \xrightarrow{a} t'$ where a is an action under which a subterm t can be reduced onto a subterm t'. Terms are build up from an empty process ε and a set of process constants using (associative) sequential "." and (associative and commutative) parallel "$\|$" operators. The semantics of PRS can be defined by labelled transition systems (LTS) – labelled directed graphs whose nodes (states of the system) correspond to terms modulo properties of "." and "$\|$" and edges correspond to individual actions (computational steps) which can be performed in a given state. The relevance of various subclasses of PRS for modelling and analysing programs is shown e.g. in [7], for automatic verification see for example surveys [5, 18].

Mayr [12] has also shown that the reachability problem (i.e. given terms t, t': is t reducible to t'?) for PRSs is decidable. This property is important to automatic verification as many verification problems, e.g. verification of safety properties, reduce to the reachability problem. Most research (with some recent exceptions, e.g. [3, 7]) has been devoted to the PRS classes from the lower part of the PRS hierarchy, especially to pushdown automata (PDA), Petri nets (PN) and their respective subclasses. We mention the successes of PDA in modeling recursive programs (without process creation), PN modeling dynamic creation of concurrent processes (without recursive calls), and CPDS (communicating pushdown systems [2]) modeling both features. All of these formalisms subsume a notion of a finite state unit (FSU) keeping some kind of global information which is accessible by the ready to be reduced components of a PRS term – hence a FSU can regulate rewriting. On the other hand, using a FSU to extend the PRS rewriting mechanism is very powerful since the state-extended version of PA (sePA) processes has a full Turing-power [1] – the decidability of reachability is lost for sePA, all its superclasses (see Fig. 1), and CPDS as well.

In brief, the purpose of this paper is to present suitable models for some real-life patterns of software systems such that reachability remains decidable. We have proposed two PRS extensions, namely fcPRS ([19], inspired by concurrent constraint programming [17]) and wPRS ([9] for PRS equipped with weak FSU inspired by weak automata [16]). It is shown that they increase the expressive power of those PRS subclasses which do not subsume the notion of finite control. By our opinion (sub)classes of wPRS are suitable for modeling some software systems which can be found in the areas of real-time control programs and communication and cryptographic protocols. In wPRS rewriting, FSU can cycle in any control state, but it can change its state only finitely many times. Hence an LTS generated by wPRS models the consecutive execution of the respective (and differently working) phases of the mentioned software systems.

The outline of the paper is a follows: after some preliminaries we introduce a uniform framework for specifying all extended PRS formalisms in Section 3 and compare their relative expressibility with respect to strong bisimulation in Section 4. Here we also solve (to the best of our knowledge) an open problem of the relationship between PN and sePA classes by showing that PN are less expressible (up to bisimulation) than sePA. In Section 5 we show that all classes of our fcPRS and wPRS extensions keep the reachability problem decidable. The last section summarises our results.

Related work: In the context of reachability analysis one can see at least two approaches: (i) abstraction (approximate) analysis techniques on stronger 'models' such as sePA and its superclasses with undecidable reachability, e.g. see a recent work [2], and (ii) precise techniques for 'weaker' models, e.g. PRS classes with decidable reachability, e.g. [10] and another recent work [3]. In the latter one, symbolic representations of set of reachable states are built with respect to various term structural equivalences. Among others it is shown that for the PAD class and the same equivalence as in this paper, when properties of sequential and parallel compositions are taken into account, one can construct nonregular representations based on counter tree automata.

2 Preliminaries

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*.

We 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 $\alpha_0 \xrightarrow{\sigma} \alpha$. Let $Const = \{X, \ldots\}$ be a countably infinite set of *process constants*. The set $\mathcal{T}$ of *process terms* (ranged over by $t, \ldots$) is defined by the abstract syntax $t = \varepsilon \mid X \mid t_1.t_2 \mid t_1\|t_2$, where ε is the empty term, $X \in Const$ is a process constant (used as an atomic process), '$\|$' 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 '$\|$' and modulo associativity of '.' We also define $\varepsilon.t = t = t.\varepsilon$ and $t\|\varepsilon = t$.

We distinguish four *classes of process terms*: '1' stands for terms consisting of a single process constant only (i.e. $\varepsilon \notin 1$), '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 $Act = \{a, b, \cdots\}$ 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) *Δ is a pair (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.

Given PRS Δ 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:

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1\|t_2 \xrightarrow{a} t_1'\|t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2}.$$

If no confusion arises, we sometimes speak about a "process rewrite system" meaning a "labelled transition system generated by process rewrite system".

Some classes of (α, β)-PRS correspond to widely known models as finite state systems (FS), basic process algebras (BPA), basic parallel processes (BPP), process algebras (PA), pushdown processes (PDA, see [6] for justification), and Petri nets (PN). The other classes were introduced (and named as PAD, PAN, and PRS) by Mayr [12]. The correspondence between (α, β)-PRS classes and acronyms just mentioned can be seen in Figure 1.

3 Extended PRS

In this section we recall the definitions of three different extensions of process rewrite systems, namely *state-extended PRS (sePRS)* [8], *PRS with a finite constraint system (fcPRS)* [19], and *PRS with a weak finite-state unit (wPRS)* [9]. In all cases, the PRS formalism is extended with a finite state unit of some kind.

sePRS State-extended PRS corresponds to PRS extended with s finite state unit without any other restrictions. The well-known example of this extension is the state-extended BPA class (also known as pushdown processes).

wPRS The notion of weakness employed in wPRS formalism corresponds to weak automaton [16] in automata theory. The behaviour of a weak state unit is acyclic, i.e. states of state unit are ordered and non-increasing during every sequence of actions. As the state unit is finite, its state can be changed only finitely many times during every sequence of actions.

fcPRS The extension of PRS with finite constraint systems is motivated by *concurrent constraint programming (CCP)* (see e.g. [17]). In CCP the processes work with a shared *store* (seen as a constraint on values that variables can represent) via two operations, *tell* and *ask*. The *tell* adds a constraint to the store provided the store remains *consistent*. The *ask* is a test on the store – it can be executed only if the current store implies a specified constraint.

Formally, values of store form a bounded lattice (called a *constraint system*) with the lub operation $\wedge$ (least upper bound), the least element *tt*, and the greatest element *ff*. The execution of *tell(n)* changes the value of the store from o to $o \wedge n$ (provided $o \wedge n \neq ff$ – consistency check). The *ask(m)* can be executed if the current value of the store o is greater than m.

The state unit of fcPRS has the same properties as the store in CCP. We add two constraints (m, n) to each rewrite rule. The application of a rule corresponds to the concurrent execution of *ask(m)*, *tell(n)*, and rewriting:

- a rule can be applied only if the actual store o satisfies $m \leq o$ and $o \wedge n \neq ff$,
- the application of the rule rewrites the process term and changes the store to $o \wedge n$.

At first we define the common syntax of extended PRS and then we specify the individual restrictions on state units.

Definition 3.1. *Let Act* $= \{a, b, \cdots\}$ *be a countably infinite set of* atomic actions*,* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$*. An* extended (α, β)-PRS Δ *is a tuple* $(M, \leq, R, m_0, t_0)$*, where*

- *M is a finite set of* states *of* state unit*,*
- $\leq$ *is a binary relation over* M*,*
- *R is a finite set of* rewrite rules *of the form* $(m, t_1) \xrightarrow{a} (n, t_2)$*, where* $t_1 \in \alpha$*,* $t_1 \neq \varepsilon$*,* $t_2 \in \beta$*,* $m, n \in M$*, and* $a \in Act$*,*
- *Pair* $(m_0, t_0) \in M \times \beta$ *forms a distinguished* initial state *of the system.*

The specific type of extended (α, β)-PRS is given by further requirements on $\leq$. An extended (α, β)-PRS is

- (α, β)-*sePRS* without any requirements.[1]
- (α, β)-*wPRS* iff $(M, \leq)$ is a partially ordered set.
- (α, β)-*fcPRS* iff $(M, \leq)$ is a bounded lattice. The lub operation (least upper bound) is denoted by $\wedge$, the least and the greatest elements are denoted by *tt* and *ff*, respectively. We also assume that $m_0 \neq ff$.

[1]In this case, the relation $\leq$ can be omitted from the definition.

To shorten our notation we prefer mt over (m,t). As in the PRS case, instead of $(mt_1 \xrightarrow{a} nt_2) \in R$ where $\Delta = (M, \leq, R, m_0, t_0)$, we usually write $(mt_1 \xrightarrow{a} nt_2) \in \Delta$. The meaning of $Const(\Delta)$ (process constants used in rewrite rules) and $Act(\Delta)$ (actions occurring in rewrite rules) for a given extended PRS Δ is also the same as in the PRS case.

The semantics of extended (α,β)-PRS Δ is given by the corresponding labelled transition system $(S, Act(\Delta), \longrightarrow, m_0t_0)$, where $S = M \times \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$[2] and the relation $\longrightarrow$ is defined as the least relation satisfying the inference rule corresponding to the application of rewrite rules (and dependent on the concrete formalism):

$$\text{sePRS} \qquad \frac{(mt_1 \xrightarrow{a} nt_2) \in \Delta}{mt_1 \xrightarrow{a} nt_2}$$

$$\text{wPRS} \qquad \frac{(mt_1 \xrightarrow{a} nt_2) \in \Delta}{mt_1 \xrightarrow{a} nt_2} \text{ if } n \leq m$$

$$\text{fcPRS} \qquad \frac{(mt_1 \xrightarrow{a} nt_2) \in \Delta}{ot_1 \xrightarrow{a} (o \wedge n)t_2} \text{ if } m \leq o \text{ and } o \wedge n \neq f\!f$$

and two common inference rules

$$\frac{mt_1 \xrightarrow{a} nt_1'}{mt_1 \| t_2 \xrightarrow{a} nt_1' \| t_2}, \qquad \frac{mt_1 \xrightarrow{a} nt_1'}{mt_1.t_2 \xrightarrow{a} nt_1'.t_2},$$

where $t_1, t_2, t_1' \in \mathcal{T}$ and $m, n, o \in M$.

Instead of $(1,S)$-sePRS, $(1,S)$-wPRS, $(1,S)$-fcPRS, ... we use a more natural notation seBPA, wBPA, fcBPA, etc. The class seBPP is also known as *parallel pushdown automata (PPDA)* or *multiset automata (MSA)*, see [14].

4 Expressiveness

Figure 1 describes the hierarchy of PRS classes and their extended counterparts with respect to bisimulation equivalence. If any process in class X can be also defined (up to bisimulation) in class Y we write $X \subseteq Y$. If additionally $Y \not\subseteq X$ holds, we write $X \subsetneq Y$ and say X is less expressive than Y. This is depicted by the line(s) connecting X and Y with Y

[2]If Δ is an fcPRS, we eliminate the states with $f\!f$ as they are unreachable.

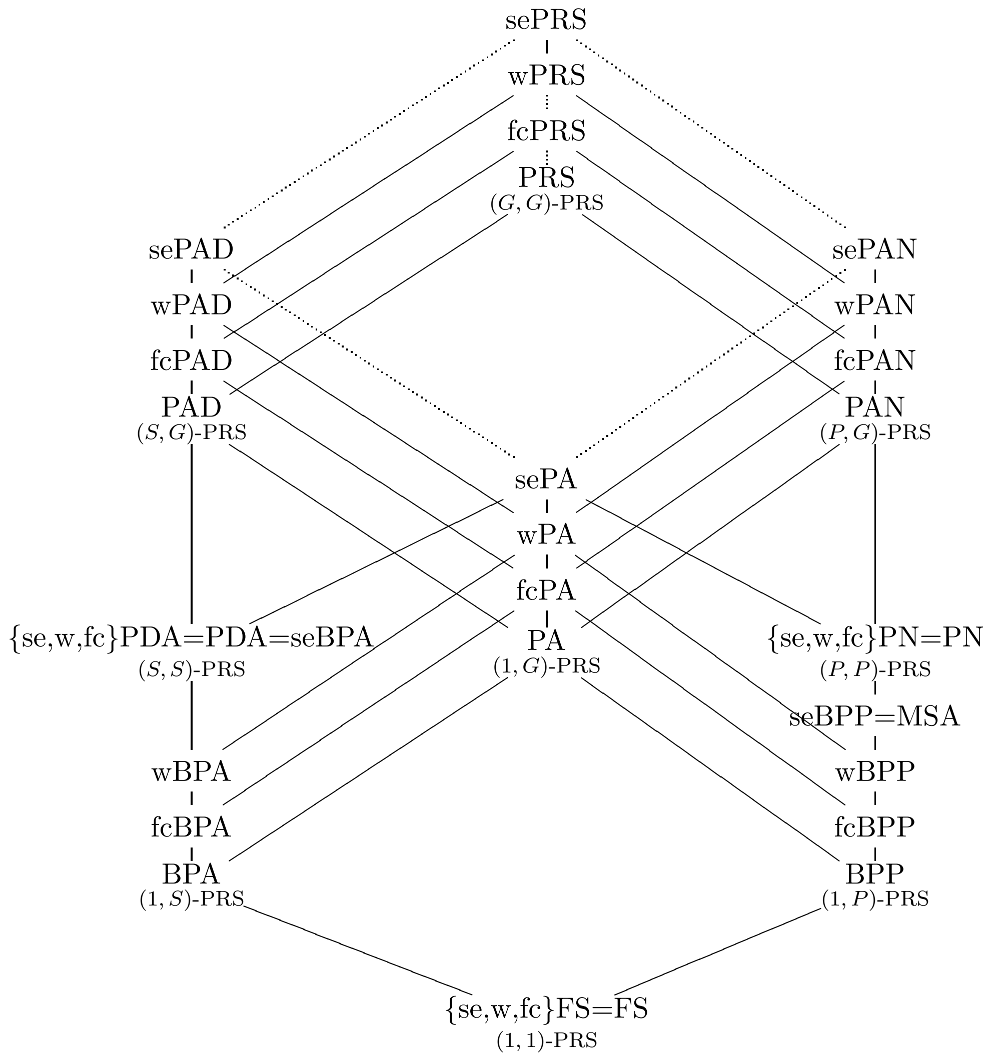

Figure 1: The hierarchy of classes defined by (extended) rewrite formalisms.

placed higher than X in Figure 1. The dotted lines represent the facts $X \subseteq Y$, where we just conjecture that $X \subsetneq Y$ hold.

Some observations (even up to isomorphism) are immediate, for example (i) collapses of the classes FS, PDA and PN with their extended analogues, (ii) if e $\in \{se, w, fc\}$ and X $\subseteq$ Y then eX $\subseteq$ eY, and (iii) (α,β)-PRS $\subseteq$ (α,β)-fcPRS $\subseteq$ (α,β)-wPRS $\subseteq$ (α,β)-sePRS for every (α,β)-PRS class.

The strictness ('$\subsetneq$') of the PRS-hierarchy has been proved by Mayr [12], that of the corresponding classes of PRS and fcPRS has been proved in [19], and that of relating fcPRSs, wPRSs, and MSA is shown in [9]. Note the strictness relations wX $\subsetneq$ seX hold for all X = PA, PAD, PAN, PRS due to our reachability result for wPRS given in Sec. 5 and due to the full Turing-power of sePA [1].

These proofs together with Moller's result establishing MSA $\subsetneq$ PN [15] complete the strictness proof of Figure 1 – with one exception, namely the relation between PN and sePA classes. Looking at two lines leaving sePA down to the left and down to the right, we note the "left-part collapse" of (S, S)-PRS and PDA proved by Caucal [6] (up to isomorphism). The right-part counterpart is slightly different due to the previously mentioned MSA $\subsetneq$ PN. In the next subsection we prove PN $\subsetneq$ sePA (in fact it suffices to show PN $\subseteq$ sePA as the strictness is obvious).

4.1 $PN \subsetneq sePA$

We now show that Petri nets are less expressive (with respect to bisimulation) than state-extended Process Algebras. The proof is done by a construction of a sePA Δ' bisimilar to a given PN Δ. In this section, a Petri net is considered in traditional notation (via finite sets of labelled transitions and places). A state of a PN is a marking of the places $P_1, P_2, \ldots, P_k, k = |Const(\Delta)|$ and it is given as a k-tuple, where the i-th component stands for the number of tokens at place P_i.

Let L_i be the maximal number of arrows between any transition and place P_i. We put $M_i = k * L_i$.

Each state of sePA Δ' will consist of a term (a parallel composition of k counters for corresponding marking) and a state of a finite-state control unit (FSU). Each state of FSU is the product of three parts as:

$$\underbrace{\{1,\ldots,k\}}_{\text{update controller}} \times \underbrace{(\{-M_1,\ldots,2*M_1\}\times\ldots\times\{-M_k,\ldots,2*M_k\})}_{\text{modulo counter}} \times \underbrace{\{0,1\}^k}_{\text{empty info}}$$

The *update controller* goes around the range and refers to the counter being updated in the next step. The *modulo counter* is k-tuple of counters with values from $-M_i$ to $2 * M_i$. Each of them saves the number of tokens in one state counted modulo M_i. The *empty info* says which term counters are empty.

We define $2k$ process constants $B_i, X_i \in Const(\Delta')$, B_i representing the bottom of i-th counter and X_i representing M_i tokens at place P_i.

For a given initial marking $\alpha = (p_1, p_2, \ldots, p_k)$ of a PN Δ we construct the following initial state of the sePA Δ'

$$1(m_1, m_2, \ldots, m_k)(e_1, e_2, \ldots, e_k)t_1 \| t_2 \| \cdots \| t_k$$

where $m_i = p_i \mod M_i$, if $n = 0$ then $e_i = 1$ else $e_i = 0$, and $t_i = X_i^n B_i$, where $n = p_i$ div M_i. In other words we have $p_i = m_i + n * M_i$.

For each PN transition $((l_1, l_2, \ldots, l_k) \xrightarrow{a} (r_1, r_2, \ldots, r_k)) \in \Delta$ we construct the set of sePA rules

$$s(m_1, \ldots, m_k)(e_1, \ldots, e_k)t \xrightarrow{a} s'(m'_1, \ldots, m'_k)(e'_1, \ldots, e'_k)t'$$

such that they obey the following conditions:

- Update controller conditions: $s \in \{1, \ldots, k\}$ and $s' = (s \mod k) + 1$.
- The general conditions for modulo counters and empty infos ($1 \leq i \leq k$):
  - $m_i, m'_i \in \{-M_i, \ldots, 2 * M_i\}$, $e_i, e'_i \in \{0, 1\}$,
  - if $e_i = 1$ then $m_i \geq l_i$ (i.e. the transition can be performed),
  - if $i \neq s$ then $m'_i = m_i - l_i + r_i, e'_i = e_i$
    else $m'_s = (m_s - l_s + r_s) \mod M_s$

We now specify e'_s and the terms t, t'. The first two *Bottom rules*, $t = B_s$, are the rules for working with the empty stack. The next three *Top rules*, $t = X_s$, describe the rewriting of a process constant X_s. Depending on the values of $m_s - l_s + r_s$, there are *dec*, *inc*, and *basic* variants manipulating the s-th stack.

Rule	t	$m_s - l_s + r_s \in$	e'_s	t'
Bottom-basic rule	B_s	$\{0, \ldots, M_s - 1\}$	1	B_s
Bottom-inc rule	B_s	$\{M_s, \ldots, 2 * M_s\}$	0	$X_s.B_s$
Top-dec rule	X_s	$\{-M_s, \ldots, -1\}$	0	ε
Top-basic rule	X_s	$\{0, \ldots, M_s - 1\}$	0	X_s
Top-inc rule	X_s	$\{M_s, \ldots, 2 * M_s\}$	0	$X_s.X_s$

Notation. In the following Lemmata 4.1 to 4.3 let β be a reachable state of sePA Δ', $\beta = s(m_1, m_2, \ldots, m_k)(e_1, e_2, \ldots, e_k)t_1 \| t_2 \| \ldots \| t_k$, and n_i to be the number of constants X_i in the term t_i of β. We also refer to α as a marking of PN Δ corresponding to the state β, and p_i is the number of tokens at the i-th place of the marking α, and $((l_1, l_2, \ldots l_k) \xrightarrow{a} (r_1, r_2, \ldots r_k)) \in \Delta$ is a PN rule.

The following lemma shows that modulo counters never overflow.

Lemma 4.1. $-M_i + L_i - 1 < m_i < 2 * M_i - L_i$ *for all reachable states.*

Proof. If the i-th stack has been just updated, then $-1 < m_i < M_i$. As there are exactly $k - 1$ states to the next updating and each transition works with at most L_i tokens of P_i, each of the states differs from the updated one by at most $(k - 1) * L_i$ tokens at P_i. As $M_i = k * L_i$, the lemma holds. □

Lemma 4.2. $p_i = m_i + n_i * M_i$ *for all reachable states.*

Proof. For the initial state the lemma is implied directly from the definition. The inductive step proving the lemma for other reachable states is a straightforward consequence of the m_i' and t' conditions in the definition of sePA rules. □

Lemma 4.2 shows that every sePA state β saves the numbers of tokens of α. The following lemma proves that every transition of Δ can be performed in α if and only if there is a corresponding rewrite rule that can be used in β.

Lemma 4.3. $p_i \geq l_i$ *iff* $(e_i = 0$ *or* $m_i \geq l_i)$ *for all reachable states.*

Proof. If the i-th stack has just been updated and $e_i = 1$, a Bottom-basic rule was used and so $m_i = p_i$. These conditions stay unchanged till the next updating.

If e_i has been updated to 0, then $p_i \geq M_i$. There are $k - 1$ states to the next updating. Hence $p_i \geq M_i - (k - 1) * L_i = L_i$ in all these states and according to the definition of L_i, $L_i \geq l_i$ and so $p_i \geq l_i$. □

Theorem 4.4. *PN* $\subsetneq$ *sePA with respect to bisimulation.*

Proof. Lemma 4.1 and Lemma 4.2 show that the construction of sePA presented here, saves every marking correctly, while Lemma 4.3 proves that the corresponding states are bisimilar. Hence, PN $\subseteq$ sePA (with respect to bisimulation). Strictness follows from two of the results mentioned in the introduction, namely the full Turing-power of sePA [1] and the decidability of reachability for PRS [12]. □

5 Reachability for wPRS is decidable

In the following we show that for a given wPRS Δ and its states rt_1, st_2 it is decidable whether st_2 is reachable from rt_1 or not (st_2 is reachable from rt_1 if a sequence of actions σ such that $rt_1 \xrightarrow{\sigma} st_2$) exists.

Our proof exhibits a similar structure to the proof of decidability of the reachability problem for PRS [12]; at first we reduce the general problem to the reachability problem for wPRS with rules containing at most one occurrence of a sequential or parallel operator, and then we solve this subproblem using the fact that reachability problems for both PN and PDA are decidable [11, 4]. The latter part of the proof is based on a new idea of *passive steps* presented later.

As the labels on rewrite rules are not relevant here, we omit them in this section. To distinguish between rules and rewriting sequences we use $rt_1 \succ\!\!-^{\Delta} st_2$ to denote that in wPRS Δ the state st_2 is reachable from rt_1. Further, states of weak state unit are called *weak states*.

Definition 5.1. *Let Δ be a wPRS. A rewrite rule in Δ is* parallel *or* sequential *if it has one of the following forms:*

$$\begin{array}{rllll} \text{parallel}: & pX \longrightarrow qY\|Z & pX\|Y \longrightarrow qZ & pX \longrightarrow qY & pX \longrightarrow q\varepsilon, \\ \text{sequential}: & pX \longrightarrow qY.Z & pX.Y \longrightarrow qZ & pX \longrightarrow qY & pX \longrightarrow q\varepsilon, \end{array}$$

where X, Y, Z are process constants and p, q are weak states. A rule is trivial *if it is both parallel and sequential (i.e. it has the form $pX \longrightarrow qY$ or $pX \longrightarrow q\varepsilon$). A wPRS Δ is in* normal form *if every rewrite rule in Δ is parallel or sequential.*

Lemma 5.2. *For wPRS Δ, terms t_1, t_2, and weak states r, s, there are terms t'_1, t'_2 of wPRS Δ' in normal form satisfying $rt_1 \succ\!\!-^{\Delta} st_2 \iff rt'_1 \succ\!\!-^{\Delta'} st'_2$. Moreover, wPRS Δ' and terms t'_1, t'_2 can be effectively constructed.*

Proof. In this proof we assume that the sequential composition is left-associative. It means that the term $X.Y.Z$ is $(X.Y).Z$ and so its subterms are X, Y, Z, and $X.Y$, but not $Y.Z$. However, the term $Y\|Z$ is a subterm of $X.(Y\|Z)$.

Let $size(t)$ denote the number of sequential and parallel operators in term t. For every wPRS Δ, let k_i be the number of rules $(rt_1 \longrightarrow st_2) \in \Delta$ that are neither parallel nor sequential and $size(rt_1 \longrightarrow st_2) = i$, where $size(rt_1 \longrightarrow st_2) = size(t_1) + size(t_2)$. Thus, Δ is in normal form iff $k_i = 0$ for every i. In this case, let $n = 0$. Otherwise, let n be the maximal i such that $k_i \neq 0$ (n existing as the set of rules is finite). We define $norm(\Delta)$ to be the pair (n, k_n).

Now we describe a procedure transforming Δ (if it is not in a normal form) onto a wPRS Δ' and terms t_1, t_2 onto terms t'_1, t'_2 such that $norm(\Delta') < norm(\Delta)$ (with respect to lexicographical ordering) and $rt_1 \succ\!\!-^{\Delta} st_2 \iff rt'_1 \succ\!\!-^{\Delta'} st'_2$.

Let us assume that wPRS Δ is not in normal form. Then there is a rule that is neither sequential nor parallel and has the maximal *size*. Take a non-atomic subterm t of this rule and replace every subterm t in Δ (i.e. in rewrite rules and initial term) and in t_1 and t_2 by a fresh constant X. Then add two rules $pX \longrightarrow pt$ and $pt \longrightarrow pX$ for each weak state p. This yields a new wPRS Δ' and terms t'_1 and t'_2 where the constant X serves as an abbreviation for the term t. By the definition of *norm* we get $\mathit{norm}(\Delta') < \mathit{norm}(\Delta)$. The correctness of our transformation remains to be demonstrated:

$$rt_1 \succ^{\Delta} st_2 \iff rt'_1 \succ^{\Delta'} st'_2$$

The implication $\Longleftarrow$ is obvious. For the opposite direction we show that every rewriting step in Δ from pl_1 to ql_2 under the rule $(pl \longrightarrow ql') \in \Delta$ corresponds to a sequence of several rewriting steps in Δ' leading from pl'_1 to ql'_2, where l'_1, l'_2 equal to l_1, l_2 with all occurrences of t replaced by X. Let us assume the rule $pl \longrightarrow ql'$ modifies a subterm t of pl_1, and/or a subterm t appears in ql_2 after the rule application (other cases are trivial). If the rule modifies a subterm t of l_1 there are two cases. Either l subsumes whole t and then the corresponding rule in Δ' (with t replaced by X) can be applied directly on pl'_1, or due to the left-associativity of sequential operator, t is not a subterm of the right part of any sequential composition in l_1 and thus the application of the corresponding rule in Δ' on pl'_1 is preceded by an application of the added rule $pX \longrightarrow pt$. The situation when subterm t appears in ql_2 after the application of the considered rule is similar. Either l' subsumes whole t and then the application of the corresponding rule in Δ' results directly in ql'_2, or t is not a subterm of the right part of any sequential composition in l_2 and thus the application of the corresponding rule in Δ' is followed by an application of the added rule $qt \longrightarrow qX$ reaching the state ql'_2.

By repeating this procedure we finally get a wPRS Δ'' in normal form and terms t''_1,t''_2 satisfying $rt_1 \succ^{\Delta} st_2 \iff rt''_1 \succ^{\Delta''} st''_2$. $\Box$

Mayr's proof of the reachability problem for PRS now completes the PRS Δ in normal form into so-called *transitive normal form* satisfying $(X \longrightarrow Y) \in \Delta$ whenever $X \succ^{\Delta} Y$. This step employs the local effect of rewriting under sequential rules in a parallel environment and vice versa. Intuitively, whenever there is a rewriting sequence

$$X \| Y \longrightarrow (X_1.X_2) \| Y \longrightarrow (X_1.X_2) \| Z \longrightarrow X_2 \| Z$$

in PRS in normal form, then the rewriting of each parallel component is independent in the sense that there are also rewriting sequences $X \longrightarrow X_1.X_2 \longrightarrow X_2$ and $Y \longrightarrow Z$. This does not hold for wPRS in normal form as the rewriting on one parallel component can influence the rewriting on other parallel components via a weak state unit. To get its independence back we introduce the concept of *passive steps* emulating changes of a weak state produced by the environment.

Definition 5.3. *A finite sequence of weak states pairs* $PS = \{(p_i, q_i)\}_{i=1}^{n}$ *satisfying* $p_1 > q_1 \geq p_2 > q_2 \geq \cdots \geq p_n > q_n$ *is called* passive steps*.*

Let Δ *be a wPRS and* PS *be passive steps. By* $\Delta + PS$ *we denote a system* Δ *with an added rule* $pX \longrightarrow qX$ *for each* (p, q) *in* PS *and* $X \in Const(\Delta)$*. For all terms* t_1, t_2 *and weak states* r, s *we write*

$rt_1 \succ_{triv}^{\Delta+PS} st_2$ *iff* $rt_1 \succ^{\Delta+PS} st_2$ *via trivial rules,*

$rt_1 \succ_{seq}^{\Delta+PS} st_2$ *iff* $rt_1 \succ^{\Delta+PS} st_2$ *via sequential rules,*

$rt_1 \succ_{par}^{\Delta+PS} st_2$ *iff* $rt_1 \succ^{\Delta+PS} st_2$ *via parallel rules.*

Informally, $rt_1 \succ^{\Delta+PS} st_2$ means that the state rt_1 can be rewritten onto state st_2 provided a weak state can be passively changed from p to q for every passive step (p, q) in PS. Thanks to the finiteness of a weak state unit, the number of different passive steps is finite.

Definition 5.4. *Let wPRS* Δ *be in normal form. If for every* $X, Y \in Const(\Delta)$*, weak states* r, s*, and passive steps* PS *it holds that*

- $rX \succ^{\Delta+PS} sY \implies rX \succ_{triv}^{\Delta+PS} sY$
  then Δ *is in* flatted normal form*,*
- $rX \succ_{seq}^{\Delta+PS} sY \implies rX \succ_{triv}^{\Delta+PS} sY$
  then Δ *is in* sequential flatted normal form*,*
- $rX \succ_{par}^{\Delta+PS} sY \implies rX \succ_{triv}^{\Delta+PS} sY$
  then Δ *is in* parallel flatted normal form*.*

The following lemma says that it is sufficient to check reachability via sequential rules and via parallel rules in order to construct a wPRS in flatted normal form. This allows to reduce the reachability problem for wPRS to the reachability problems for wPN and wPDA (i.e. to the reachability problems for PN and PDA).

Lemma 5.5. *If a wPRS is in both sequential and parallel flatted normal form then it is in flatted normal form as well.*

Proof. We assume the contrary and derive a contradiction. Let Δ be a wPRS in sequential and parallel flatted normal form. Now let us choose passive steps PS and a rewriting sequence in $\Delta + PS$ leading from rX to sY such that $rX \not\succ_{triv}^{\Delta+PS} sY$ and the number of applications of non-trivial rewrite rules used in the sequence is minimal.

As the wPRS Δ is in both sequential and parallel flatted normal form, $rX \not\succ_{seq}^{\Delta+PS} sY$ and $rX \not\succ_{par}^{\Delta+PS} sY$. Hence, both sequential and parallel operators occur in the rewriting sequence. There are two cases.

1. Assume that a sequential operator appears first. The parallel operator is then introduced by a rule in the form $pU \longrightarrow qU_1\|U_2$ applied to a state $pU.t$, where $t \in S$. From $q(U_1\|U_2).t \succ^{\Delta+PS} sY$ and the fact that at most one process constant can be removed in one rewriting step, it follows that in the rest of the sequence considered, the term $(U_1\|U_2)$ is rewritten onto a process constant (say V) first. Let PS' be PS in this case.

2. Assume that a parallel operator appears first. The sequential operator is then introduced by a rule in the form $pU \longrightarrow qU_1.U_2$ applied to a state $pU\|t$, where $t \in P$. The rest of the sequence subsumes steps rewriting the term $U_1.U_2$ onto a process constant (say V). Contrary to the previous case, these steps can be interleaved with steps rewriting other parallel components and possibly changing weak state. Let PS' be passive steps PS merged with these changes of weak states.

Consequently, we have a rewriting sequence in $\Delta + PS'$ from pU to oV (for some o) with fewer applications of non-trivial rewrite rules. As the number of applications of non-trivial rewrite rules used in the original sequence is minimal we get $pU \not\succ_{triv}^{\Delta+PS'} oV$. This contradicts our choice of rX, sY, and PS. $\square$

Lemma 5.6. *For every wPRS system Δ in normal form, terms t_1, t_2 over $Const(\Delta)$, and weak states r, s of Δ a wPRS Δ' can be constructed such that Δ' is in flatted normal form satisfying $rt_1 \succ^{\Delta} st_2 \iff rt_1 \succ^{\Delta'} st_2$.*

Proof. To obtain Δ' we enrich Δ by trivial rewrite rules transforming the system into sequential and parallel flatted normal forms, which suffices thanks to Lemma 5.5. Using algorithms deciding reachability for PDA and PN, the algorithm checks if there are some weak states r, s, constants $X, Y \in Const(\Delta)$, and passive steps $PS = \{(p_i, q_i)\}_{i=1}^{n}$ (satisfying $r \geq p_1$

and $q_n \geq s$ as weak states pairs beyond this range are of no use here) such that $rX \succ_{seq}^{\Delta+PS} sY \ \vee\ rX \succ_{par}^{\Delta+PS} sY$ and $rX \not\succ_{triv}^{\Delta+PS} sY$. We finish if the answer is negative. Otherwise we add to Δ rules $rX \longrightarrow p_1Z_1$, $q_iZ_i \longrightarrow p_{i+1}Z_{i+1}$ for $i = 1, \ldots, n-1$, and $q_nZ_n \longrightarrow sY$, where $Z_1, \ldots, Z_n$ are fresh process constants (if $n = 0$ then we add just the rule $rX \longrightarrow sY$). The algorithm then repeats this procedure on the system with added rules with one difference; the X, Y ranges over the constants of the original system Δ. This is sufficient as new constants occur only in trivial rules[3]. The algorithm terminates as the number of iterations is bounded by the number of pairs of states rX, sY of Δ, times the number of passive steps PS. The correctness follows from the fact that added rules have no influence on reachability. □

Theorem 5.7. *The reachability problem for wPRS is decidable.*

Proof. Let Δ be a wPRS and rt_1, st_2 its states. We want to decide whether $rt_1 \succ^{\Delta} st_2$ or not. Clearly $rt_1 \succ^{\Delta} st_2 \iff rX \succ^{\Delta'} sY$, where X, Y are fresh constants and Δ' arises from Δ by the addition of the rules $rX \longrightarrow rt_1$ and $st_2 \longrightarrow sY$[4]. Hence we can directly assume that t_1, t_2 are process constants, say X, Y. Lemma 5.2 and Lemma 5.6 successively reduce the question whether $rX \succ^{\Delta} sY$ to question whether $rX \succ^{\Delta'} sY$, where Δ' is in flatted normal form – note that Lemma 5.2 does not change terms t_1, t_2 if they are process constants. The definition of flatted normal form implies $rX \succ^{\Delta'} sY \iff rX \succ_{triv}^{\Delta'} sY$. Finally the relation $rX \succ_{triv}^{\Delta'} sY$ is easy to check. □

6 Conclusions

We have presented a unified view on some (non-conservative) extensions of Process Rewrite Systems. Comparing the mutual expressiveness of the respective subclasses up to bisimulation equivalence, we have added some new strict relations, including the class of Petri Nets being less expressible than the class of Process Algebra extended with finite state control unit. We have shown that our extensions keep the reachability problem decidable and we believe that they may be suitable for modeling some real-life software systems.

[3]If the system with added rules is not in sequential or parallel flatted normal form, then there is a counterexample with the constants X, Y of the original system Δ.

[4]If $t_2 = \varepsilon$ then this is not a correct rule. In this case we need to add to Δ' a rule $pt \longrightarrow qY$ for each rule $(pt \longrightarrow q\varepsilon) \in \Delta$.

References

[1] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Proc. of LICS'95*. IEEE, 1995.

[2] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal on Foundations of Computer Science*, 14(4):551–582, 2003.

[3] A. Bouajjani and T. Touili. Reachability Analysis of Process Rewrite Systems. In *Proc. of FST&TCS-2003*, volume 2914 of *LNCS*, pages 74–87. Springer, 2003.

[4] J. R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.

[5] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.

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

[7] J. Esparza. Grammars as processes. In *Formal and Natural Computing*, volume 2300 of *LNCS*. Springer, 2002.

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

[9] M. Křetínský, V. Řehák, and J. Strejček. Process Rewrite Systems with Weak Finite-State Unit. Technical Report FIMU-RS-2003-05, Masaryk University Brno, 2003. to appear in ENTCS as Proc.of INFINITY 03.

[10] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 50–66, 1998.

[11] E. W. Mayr. An algorithm for the general petri net reachability problem. In *Proc. of 13th Symp. on Theory of Computing*, pages 238–246. ACM Press, 1981.

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

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

[14] F. Moller. Infinite results. In *Proc. of CONCUR'96*, volume 1119 of *LNCS*, pages 195–216. Springer, 1996.

[15] F. Moller. Pushdown Automata, Multiset Automata and Petri Nets, MFCS Workshop on concurrency. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[16] D. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoret. Computer Science*, 97(1–2):233–244, 1992.

[17] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of 17th POPL*, pages 232–245. ACM Press, 1990.

[18] J. Srba. Roadmap of infinite results. *EATCS Bulletin*, (78):163–175, 2002. `http://www.brics.dk/~srba/roadmap/`.

[19] J. Strejček. Rewrite systems with constraints, EXPRESS'01. *Electronic Notes in Theoretical Computer Science*, 52, 2002.