Process Rewrite Systems

Richard Mayr

Institut für Informatik, Technische Universität München,
Arcisstr. 21, D-80290 München, Germany;
e-mail: `mayrri@informatik.tu-muenchen.de`
fax: +49 (89) 289-28207/28483
Web: `http://www7.informatik.tu-muenchen.de/~mayrri`

Abstract

Many formal models for infinite-state concurrent systems are equivalent to special classes of rewrite systems. We classify these models by their expressiveness and define a hierarchy of classes of rewrite systems. We show that this hierarchy is strict with respect to bisimulation equivalence.

The most general and most expressive class of systems in this hierarchy is called *Process Rewrite Systems* (PRS). They subsume Petri nets, PA-Processes and pushdown processes and are strictly more expressive than any of these. Intuitively, PRS can be seen as an extension of Petri nets by subroutines that can return a value to their caller. We show that the reachability problem is decidable for PRS. It is even decidable if there is a reachable state that satisfies certain properties that can be encoded in a simple logic. Thus PRS are more expressive than Petri nets, but not Turing-powerful.

1 Introduction

Petri nets and process algebras are two kinds of formalisms used to build abstract models of concurrent systems. These abstract models are used for verification, because they are normally smaller and more easily handled than full programs. Formal models should be simple enough to allow automated verification, or at least computer-assisted verification. On the other hand they should be as expressive as possible, so that most aspects of real programs can be modeled.

Many different formalisms have been proposed for the description of infinite-state concurrent systems. Among the most common are Petri nets, Basic Parallel Processes (BPP), context-free processes (BPA) and pushdown processes. BPP are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset. PA-Processes [BK85, Kuc, May97b] are the smallest common generalization of BPP and BPA. PA-processes, pushdown processes and Petri nets are mutually incomparable.

We present a unified view of all these formalisms by showing that they can be seen as special subclasses of rewrite systems. Such unified representations have already been used by Stirling, Caucal and Moller [Cau92, Mol96], but only for purely sequential or purely parallel systems. Here we generalize this to systems with both sequential and parallel composition.

Basically, the rewriting formalism is first order prefix-rewrite systems on process terms without substitution and modulo commutativity and associativity of parallel composition and associativity of sequential composition. The most general class of these systems will be called *Process Rewrite Systems (PRS)*. All the previously mentioned formalisms can be seen as special cases of PRS, and PRS is strictly more general (see Theorem 4.14). Intuitively, PRS can be seen as an extension of Petri nets by subroutines that can return a value to their caller. As PRS is a very expressive model, model checking with any temporal logic (except Hennessy-Milner logic) is undecidable for it (see Section 7). However, we show that the reachability problem is decidable for PRS. The interesting point here is that PRS is strictly more general than Petri nets, but still not Turing-powerful.

The rest of the paper is structured as follows. In Section 2 we define process terms and the rewriting formalism. We describe a hierarchy of subclasses of it, which we call the *PRS-hierarchy*. Section 3 explains the intuition for the various classes in the PRS-hierarchy. In Section 4 we show that the PRS-hierarchy is strict with respect to bisimulation. In Section 5 we show that the reachability problem is decidable for PRS. Section 6 generalizes this result to reachability of certain classes of states that are described by state formulae. The paper closes with a section that summarizes the results.

2 Terms and Rewrite Systems

Many classes of concurrent systems can be described by a (possibly infinite) set of process terms, representing the states, and a finite set of rewrite rules describing the dynamics of the system.

Definition 2.1 Let $Act = \{a, b, \ldots\}$ be a countably infinite set of atomic actions and $Const = \{\epsilon, X, Y, Z, \ldots\}$ a countably infinite set of process constants. The process terms that describe the states of the system have the following form:

$$t ::= \epsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2$$

where ϵ is the empty term, $X \in Const$ is a process constant (used as an atomic process in this context), "$\|$" means parallel composition and "." means sequential composition. Parallel composition is associative and commutative. Sequential composition is associative. Let $\mathcal{T}$ be the set of process terms.

Convention 1: We always work with equivalence classes of terms modulo commutativity and associativity of parallel composition and modulo associativity of sequential composition. Also we define that $\epsilon.t = t = t.\epsilon$ and $t \| \epsilon = t$.

Convention 2: We defined that sequential composition is associative. However, when we look at terms we think of it as left-associative. So when we say that a term t has the form $t_1.t_2$, then we mean that t_2 is either a single constant or a parallel composition of process terms.

The *size* of a process term is defined as the number of occurrences of constants in it plus the number of occurrences of operators in it.

$$\begin{array}{rcl} size(\epsilon) & := & 0 \\ size(X) & := & 1 \\ size(t_1.t_2) & := & size(t_1) + size(t_2) + 1 \\ size(t_1 \| t_2) & := & size(t_1) + size(t_2) + 1 \end{array}$$

For a term t the set $Const(t)$ is the set of constants that occur in t.

$$\begin{array}{rcl} Const(\epsilon) & := & \emptyset \\ Const(X) & := & \{X\} \\ Const(t_1.t_2) & := & Const(t_1) \cup Const(t_2) \\ Const(t_1 \| t_2) & := & Const(t_1) \cup Const(t_2) \end{array}$$

The dynamics of the system is described by a finite set of rules Δ of the form $(t_1 \stackrel{a}{\to} t_2)$ where t_1 and t_2 are process terms and $a \in Act$ is an atomic action. The finite set of rules Δ induces a (possibly infinite) labeled transition system with relations $\stackrel{a}{\to}$ with $a \in Act$. For every $a \in Act$, the transition relation $\stackrel{a}{\to}$ is the smallest relation that satisfies the following inference rules.

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

where t_1, t_2, t_1', t_2' are process terms. Note that parallel composition is commutative and thus the inference rule for parallel composition also holds with t_1 and t_2 exchanged.

Since Δ is finite, the generated LTS is finitely branching. (For some classes of systems (e.g. Petri nets) the branching-degree is bounded by a constant that depends on Δ. For other classes (e.g. PA) the branching-degree is finite at every state, but it can get arbitrarily high.) Also every single Δ uses only a finite subset $Const(\Delta) := \bigcup_{(t_1 \stackrel{a}{\to} t_2) \in \Delta} (Const(t_1) \cup Const(t_2))$ of constants and only a finite subset $Act(\Delta) := \bigcup_{(t_1 \stackrel{a}{\to} t_2) \in \Delta} \{a\}$ of atomic actions. Thus for every Δ only finitely many of the generated transition relations $\stackrel{a_i}{\to}$ for $a_i \in Act$ are nonempty. (Those for which $a_i \in Act(\Delta)$). Still the generated transition system can be infinite. (Consider the analogy: Every labeled Petri net has only finitely many transitions and uses only finitely many different atomic actions, but the state space can be infinite.) The relation $\stackrel{a}{\to}$ is generalized to sequences of actions in the standard way. Sequences are denoted by σ.

Remark 2.2 *There is no operator "+" for nondeterministic choice in the process terms, because this is encoded in the set of rules Δ! There can be several rules with the same term on the left hand side. It is also possible that several rules are applicable at different places in a term. The rule that is applied and the position where it is applied are chosen nondeterministically.*

Also there is no such thing as action prefixes in the process terms. The atomic actions are introduced by the rules.

Many common models of systems fit into this scheme. In the following we characterize subclasses of rewrite systems. The expressiveness of a class depends on what kind of terms are allowed on the left hand side and right hand side of the rewrite rules in Δ.

Definition 2.3 (Classes of process terms)
We distinguish four classes of process terms:

1 Terms consisting of a single process constant like X.

S Terms consisting of a single constant or a sequential composition of process constants like $X.Y.Z$.

P Terms consisting of a single constant or a parallel composition of process constants like $X \| Y \| Z$.

G General process terms with arbitrary sequential and parallel composition like $(X.(Y \| Z)) \| W$.

Also let $\epsilon \in S, P, G$, but $\epsilon \notin 1$. It is easy to see the relations between these classes of process terms: $1 \subset S$, $1 \subset P$, $S \subset G$ and $P \subset G$. S and P are incomparable and $S \cap P = 1 \cup \{\epsilon\}$.

We characterize classes of process rewrite systems (PRS) by the classes of terms allowed on the left hand sides and the right hand sides of rewrite rules.

Definition 2.4 (PRS)
Let $\alpha, \beta \in \{1, S, P, G\}$. A (α, β)-PRS is a finite set of rules Δ where for every rewrite rule $(l \xrightarrow{a} r) \in \Delta$ the term l is in the class α and $l \neq \epsilon$ and the term r is in the class β (and can be ϵ). The initial state is given as a term $t_0 \in \alpha$. A (G, G)-PRS is simply called PRS.

Remark 2.5 *W.l.o.g. it can be assumed that the initial state t_0 of a PRS is a single constant. There are only finitely many terms $t_1, \ldots, t_n$ s.t. $t_0 \xrightarrow{a_i} t_i$. If t_0 is not a single constant then we can achieve this by introducing a new constant X_0 and new rules $X_0 \xrightarrow{a_i} t_i$ and declaring X_0 to be the initial state.*

(α, β)-PRS where α is more general than β or incomparable to β (for example $\alpha = G$ and $\beta = S$) do not make any sense. This is because the terms that are introduced by the right side of rules must later be matched by the left sides of other rules. So in a (G, S)-PRS the rules that contain parallel composition on the left hand side will never be used (assuming that the initial state is a single constant). Thus one may as well use a (S, S)-PRS. So we restrict our attention to (α, β)-PRS with $\alpha \subseteq \beta$.

Figure 1 shows a graphical description of the hierarchy of (α, β)-PRS.

Many of these (α, β)-PRS correspond to widely known models like Petri nets, pushdown processes, context-free processes and others.

1. A $(1, 1)$-PRS is a finite-state system. Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.

2. $(1, S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [BE97, Esp97]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.

3. It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left side of rules). Caucal [Cau92] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labeling of states. Thus (S, S)-PRS are equivalent to pushdown processes, the processes described by pushdown automata.

4. (P, P)-PRS are equivalent to Petri nets. Every constant corresponds to a place in the net and the number of occurrences of a constant in a term corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.

5. $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [BE97, Esp97]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [Chr93].

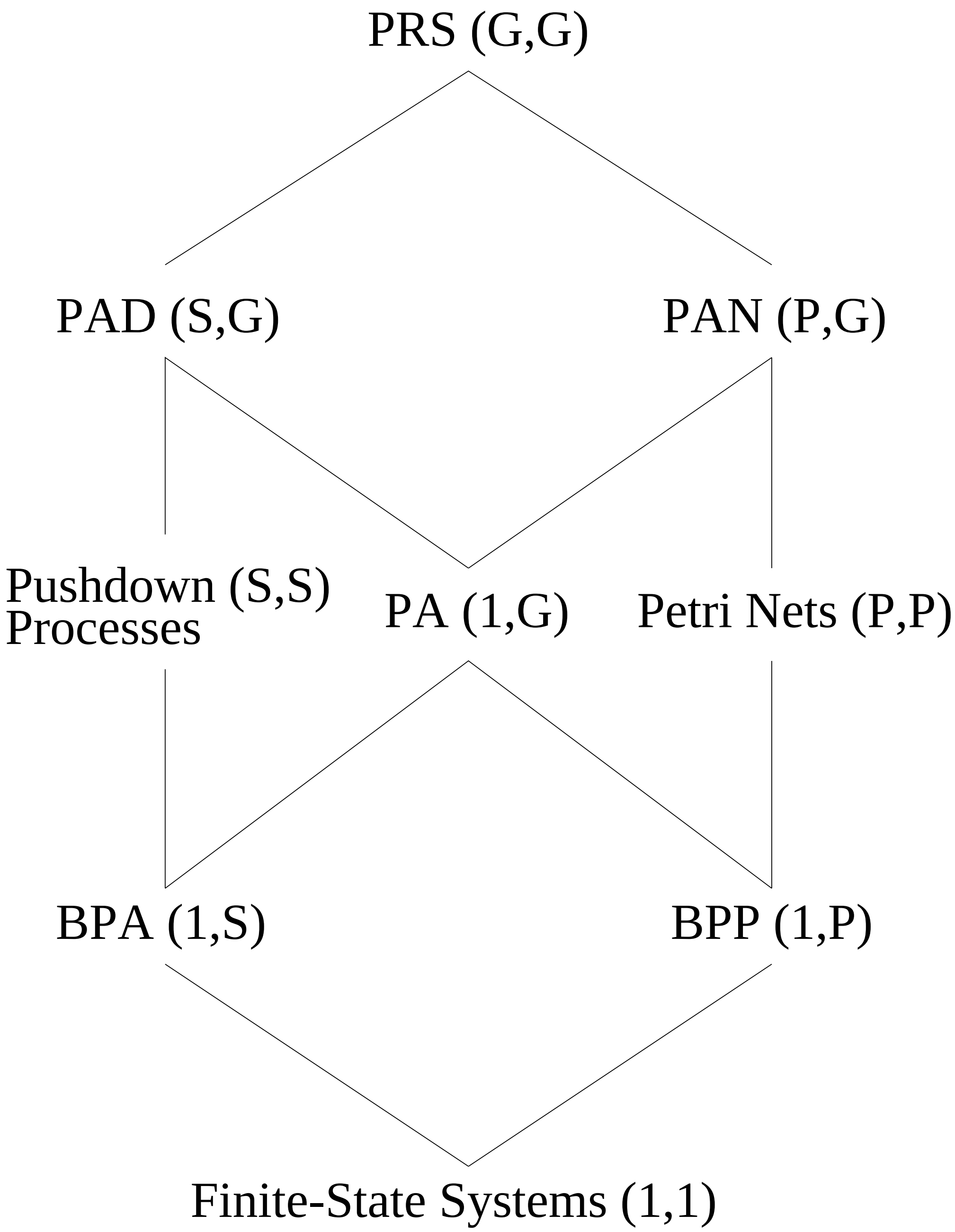

Figure 1: The PRS-hierarchy.

6. $(1, G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication (see [BK85, May97b, Kuc]).

7. (P, G)-PRS are called *PAN-processes* in [May97a]. It is the smallest common generalization of Petri nets and PA-processes and it is strictly more general than both of them (e.g. PAN can describe all Chomsky-2 languages while Petri nets cannot).

8. (S, G)-PRS are the smallest common generalization of pushdown processes and PA-processes. They are called *PAD* (PA + PD) in [May98].

9. The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [May97c]. They subsume all the previously mentioned classes.

3 The Intuition

In this section we explain the general intuition for the definition of (α, β)-PRS, i.e. what does it mean that parallel/sequential/arbitrary composition is allowed in terms on the left/right hand sides of rules?

If parallel composition is allowed on the right hand side of rules, then there can be rules of the form $t \xrightarrow{a} t_1 \| t_2$. This means that it is possible to create processes that run in parallel. The rule can be interpreted that, by action a, the process t becomes the process t_1 and spawns off the process t_2 or vice versa.

If sequential composition is allowed on the right hand side of rules, then there are rules of the form $t \xrightarrow{a} t_1.t_2$. The interpretation is that process t calls a subroutine t_1 and becomes process t_2. It resumes its execution when the subroutine t_1 terminates.

If arbitrary sequential and parallel composition is allowed on the right hand side of rules then both parallelism and subroutines are possible.

If parallel composition is allowed on the left hand side of rules, then there are rules of the form $t_1 \| t_2 \xrightarrow{a} t$. This can be interpreted as synchronization/communication of the parallel processes t_1 and t_2. This is because this action can only occur if both t_1 and t_2 change in a certain defined way.

If sequential composition is allowed on the left hand side of rules, then there can be rules of the form $t_1'.t_2 \xrightarrow{a} t'$ and $t_1''.t_2 \xrightarrow{a} t''$. The intuition is that a process t called a subroutine t_1 and became process t_2 by a rule $t \xrightarrow{a} t_1.t_2$. The subroutine may in its computation reach a state t_1' or t_1''. Now one of these rules is applicable. This means that the result of the computation of the subroutine affects the behavior of the caller when it becomes active again, since the caller can become t' or t''. The interpretation is that the subroutine returns a value to the caller when it terminates.

If arbitrary sequential and parallel composition is allowed on the left hand sides of rules then both synchronization and returning of values by subroutines are possible. It will be shown in Section 5 that rules with nested sequential and parallel composition (on the left side or the right side) do not increase the expressiveness. It suffices to have systems of rules where every single rules only contains either sequential or parallel composition.

4 The PRS-Hierarchy is Strict

The question arises if this hierarchy of (α, β)-PRS is strict. For the description of languages this is not the case, because for example context-free processes (BPA) and pushdown processes (PDA) both describe exactly the Chomsky-2 languages. However, the hierarchy is strict with respect to bisimulation equivalence. Bisimulation equivalence [Mil89] is a finer equivalence than language equivalence. It is defined as follows:

Definition 4.1 A binary relation R over the states of a labeled transition system is a *bisimulation* iff

$$\begin{array}{ll}\forall (s_1, s_2) \in R\ \forall a \in Act. & (s_1 \xrightarrow{a} s_1' \ \Rightarrow \exists s_2 \xrightarrow{a} s_2'.\ s_1' R s_2') \wedge \\ & (s_2 \xrightarrow{a} s_2' \ \Rightarrow \exists s_1 \xrightarrow{a} s_1'.\ s_1' R s_2')\end{array}$$

Two states s_1 and s_2 are *bisimilar* iff there is a bisimulation R such that $s_1 R s_2$. This definition can be extended to states in different transition systems by putting them 'side by side' and considering them as a single transition system. It is easy to see that there always exists a largest bisimulation which is an equivalence relation. It is called *bisimulation equivalence* or *bisimilarity* and it is denoted by $\sim$.

Definition 4.2 A class of processes A is more general than a class of processes B with respect to bisimulation iff the following two conditions are satisfied:

1. For every B-process there is a semantically equivalent A-process: $\forall t \in B. \exists t' \in A.\ t' \sim t$
2. There is an A-process that is not bisimilar to any B-process: $\exists t \in A. \forall t' \in B.\ t \not\sim t'$

It has already been established in [BCS96, Mol96] that the classes of finite-state systems, BPP, BPA, pushdown systems, PA and Petri nets are all different with respect to bisimulation. For PAD, PAN and PRS this remains to be shown.

The proof has two parts: First we show that there is a pushdown process that is not bisimilar to any PAN-process. Then we show that there is a Petri net that is not bisimilar to any PAD-process.

Definition 4.3 Consider the following pushdown system:

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

with the initial state $U.X$. The execution sequences of this system are as follows: First it does a sequence of actions in $\{a, b\}^*$ and then one of two things:

1. A "c", the sequence in reverse and finally a "e".
2. A "d", the sequence in reverse and finally a "f".

Now we show that this pushdown system is not bisimilar to any PAN-process. First we need several definitions and lemmas.

Definition 4.4 Let t be an arbitrary process and σ a sequence of actions. The runs of t are its computations of maximal length. We define that $only(t, \sigma)$ is true iff the following conditions are satisfied:

- All runs of t are finite.
- All these runs do the sequence of actions σ.

Lemma 4.5 (Dickson's Lemma [Dic13])
Given an infinite sequence of vectors $M_1, M_2, M_3, \ldots$ in $\mathbb{N}^k$ there are $i < j$ s.t. $M_i \leq M_j$ ($\leq$ taken componentwise).

Remember that P is the class of process terms that contain only parallel composition; see Def. 2.3.

Lemma 4.6 *For every PAN Δ there is a sequence $\sigma \in \{a, b\}^*$ s.t. no $\alpha \in P$ satisfies any of the following two conditions:*

Cond1(σ, α)**:** $\exists \alpha_c.\ \alpha \xrightarrow{c} \alpha_c \wedge only(\alpha_c, \sigma e)$

Cond2(σ, α)**:** $\exists \alpha_d.\ \alpha \xrightarrow{d} \alpha_d \wedge only(\alpha_d, \sigma f)$

Proof We assume the contrary and derive a contradiction. So we assume that there is a PAN Δ s.t. for every $\sigma_i := a^i b$ ($i \in \mathbb{N}$) there is an $\alpha^i \in P$ s.t. *Cond1*(σ_i, α^i) or *Cond2*(σ_i, α^i).
There must be an infinite subsequence of $\alpha^1, \alpha^2, \ldots$ where *Cond1*(σ_i, α^i) is always satisfied or an infinite subsequence of $\alpha^1, \alpha^2, \ldots$ where *Cond2*(σ_i, α^i) is always satisfied. W.r. we assume that there is an infinite subsequence where *Cond1*(σ_i, α^i) is always satisfied. Now we only regard this infinite subsequence. Since Δ is finite, there are only finitely many different rules in Δ that are marked with the action c. Let $(t_1 \xrightarrow{c} t_1'), \ldots, (t_n \xrightarrow{c} t_n')$ be those rules. (Note that $t_i \in P$ for every i, because Δ is a PAN. However, t_i' need not be in P.) It follows that one of these rules must be used infinitely often to obtain α_c^i from α^i. Let this rule be $(t_k \xrightarrow{c} t_k')$ for some $k \in \{1, \ldots, n\}$. Thus there is an infinite subsequence of the sequence $\alpha^1, \alpha^2, \ldots$ where only this rule is used to obtain α_c^i from α^i. Now we consider only this infinite subsequence.
We regard the sequence α^i of the α that satisfy Cond1. *Const*(Δ) is finite and $\alpha^i \in P$. Moreover, all α_i only contain constants from the finite set

$Const(\Delta)$. Thus we can apply Dickson's Lemma. By Dickson's Lemma there are $j, j' \in \mathbb{N}$ s.t. $j' > j$ and $\alpha^{j'} \geq \alpha^j$ (this means $\alpha^{j'} = \alpha^j \| \beta$ for some $\beta \in P$).

For both α^j and $\alpha^{j'}$ the rule $(t_k \overset{c}{\to} t'_k)$ is used to obtain α^j_c, $\alpha^{j'}_c$. Thus $\alpha^j = t_k \| \gamma$ for some $\gamma \in P$ and $\alpha^j_c = t'_k \| \gamma$. Also we have $\alpha^{j'} = \alpha^j \| \beta = t_k \| \gamma \| \beta$ and $\alpha^{j'}_c = t'_k \| \gamma \| \beta = \alpha^j_c \| \beta$. By Cond1 we have $only(\alpha^j_c, \sigma_j e)$ and $only(\alpha^{j'}_c, \sigma_{j'} e)$. However, $\alpha^{j'}_c$ also enables the sequence $\sigma_j e$. This is a contradiction. ■

Lemma 4.7 *For every PAN Δ there is a sequence $\Sigma \in \{A, B\}^*$ s.t. no process term t (w.r.t. Δ) is bisimilar to the pushdown system $U.\Sigma.X$ of Def. 4.3.*

Proof We assume the contrary and derive a contradiction. Assume that there is a PAN Δ s.t. for every sequence $\Sigma \in \{A, B\}^*$ there is a term $t(\Sigma)$ s.t. $t(\Sigma) \sim U.\Sigma.X$. For every Σ let $t(\Sigma)$ be the smallest term that has this property.

For any sequence $\Sigma \in \{A, B\}^*$ let $\sigma(\Sigma)$ be the sequence of actions a and b that is obtained by converting Σ to lowercase letters.

It follows from the definition of bisimulation that no process that has only finite computations can be bisimilar to a process that has an infinite computation. Thus by Def. 4.3 it follows that for every sequence $\Sigma \in \{A, B\}^*$ and every state $t(\Sigma)$ the following properties hold:

C There is a state $t_c(\Sigma)$ s.t. $t(\Sigma) \overset{c}{\to} t_c(\Sigma)$ and $t_c(\Sigma) \sim V.\Sigma.X$ and thus $only(t_c(\Sigma), \sigma(\Sigma)e)$.

D There is a state $t_d(\Sigma)$ s.t. $t(\Sigma) \overset{d}{\to} t_d(\Sigma)$ and $t_d(\Sigma) \sim W.\Sigma.X$ and thus $only(t_d(\Sigma), \sigma(\Sigma)f)$.

For every $t(\Sigma)$ the action c disables the action d and vice versa. Thus the actions c and d must both occur in the same subterm α of $t(\Sigma)$ and $\alpha \in P$. (This is because in a PAN no single action can change two separate subterms. For example in the term $(t_1.t_2)\|t_3$ (where t_1, t_2 and t_3 are not ϵ) no single action can change both t_1 and t_3.) Let α be the maximal parallel subterm of $t(\Sigma)$ where the actions c or d occur. This means that α is part of a subterm of the form $\alpha.\beta$ or $\alpha\|(\beta.\gamma)$, but not of the form $\alpha\|\beta$ for some $\beta \in P$. It follows that α cannot immediately synchronize with the rest of the term $t(\Sigma)$.

We have that $\alpha \overset{c}{\to} \alpha_c$ and $\alpha \overset{d}{\to} \alpha_d$. Let $t(\Sigma)[\alpha \to \alpha']$ be the term that one gets by replacing this one particular α in $t(\Sigma)$ by α'. (Not every subterm

α is replaced by α' !) This means that $t_c(\Sigma) = t(\Sigma)[\alpha \to \alpha_c]$ and $t_d(\Sigma) = t(\Sigma)[\alpha \to \alpha_d]$.

Without restriction we now assume that Σ begins with A (the other case is symmetric). Then $t(\Sigma)_c$ ($t(\Sigma)_d$) must enable action a, but not action b. We show that the action a must be enabled by a subterm of $t(\Sigma)_c$ ($t(\Sigma)_d$) that is different from α_c (α_d). We assume the contrary and derive a contradiction. In this case the rest of $t(\Sigma)_c$ ($t(\Sigma)_d$), without α_c (α_d), enables neither a nor b. It follows that the rest of $t(\Sigma)_c$ ($t(\Sigma)_d$) cannot do action a or b before α_c (α_d) terminates. If α_c (α_d) does not terminate then by Lemma 4.6 the conditions **C** and **D** cannot be satisfied for some Σ, a contradiction. If α_c (α_d) does terminate then for some suffixes σ', σ'' of $\sigma(\Sigma)$ we get $only(t(\Sigma)[\alpha \to \epsilon], \sigma' e)$ and $only(t(\Sigma)[\alpha \to \epsilon], \sigma'' f)$. Again this is a contradiction.

Thus the action a must be enabled at a subterm of $t(\Sigma)_c$ ($t(\Sigma)_d$) that is different from α_c (α_d). As we have $t(\Sigma) \sim U.\Sigma.X$ and $U.\Sigma.X \xrightarrow{b} U.B.\Sigma.X$ there must be a t' s.t. $t \xrightarrow{b} t'$ and $t' \sim U.B.\Sigma.X$. This action b must occur in α, because the rest of $t(\Sigma)$ cannot do b. Thus $\alpha \xrightarrow{b} \alpha'$ and $t' = t(\Sigma)[\alpha \to \alpha']$. We have $U.B.\Sigma.X \xrightarrow{c} V.B.\Sigma.X$. As the rest of t' (without α') cannot do action c we get $\alpha' \xrightarrow{c} \alpha''$ and $t(\Sigma)[\alpha \to \alpha''] \sim V.B.\Sigma.X$. However now the rest of the term $t(\Sigma)[\alpha \to \alpha'']$ (without α'') can do action a, but $V.B.\Sigma.X$ cannot. Thus $t(\Sigma)[\alpha \to \alpha''] \not\sim V.B.\Sigma.X$ and we have a contradiction. ■

Lemma 4.8 *The pushdown system $U.X$ of Def. 4.3 is not bisimilar to any PAN Δ with initial state t_0.*

Proof We assume the contrary and derive a contradiction. Assume that there is a PAN Δ with initial state t_0 s.t. $t_0 \sim U.X$. Let Σ be the sequence from Lemma 4.7. (Note that Σ depends on Δ.) The process $U.X$ can reach the state $U.\Sigma.X$. Thus t_0 must be able to reach a state t s.t. $t \sim U.\Sigma.X$. By Lemma 4.7 such a term t does not exist, a contradiction. ■

It follows directly that the pushdown system from Def. 4.3 is not bisimilar to any PA-process either. However, as PAD and PRS subsume pushdown processes, it is a PAD and PRS-process. Thus PAD is strictly more general then PA and PRS is strictly more general than PAN. PAD subsumes BPP and BPP is incomparable to pushdown systems. Thus PAD is also more general than pushdown processes. Now we show that there is a Petri net that is not bisimilar to any PAD-process.

Definition 4.9 Consider the following Petri net (given as a (P, P)-PRS).

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

The initial state is $X\|A\|B$.

Lemma 4.10 *If there is a PAD-process that is bisimilar to the state $X\|A\|B$ of the Petri net of Def. 4.9, then there is also a pushdown process that is bisimilar to $X\|A\|B$.*

Proof Let Δ be a PAD and Q the initial state s.t. $Q \sim X\|A\|B$. W.r. we can assume that Q is a single constant (see Def. 2.4). We construct a pushdown process (an (S, S)-PRS) Δ' that is also bisimilar to $X\|A\|B$.
First we show that in every reachable state of Δ of the form $(t_1\|t_2).t_3$ (t_3 can be ϵ) t_1 or t_2 must be deadlocked.
Assume that there is a state $(t_1\|t_2).t_3$ that is reachable from Q. Then a state M must be reachable from $X\|A\|B$ s.t. $(t_1\|t_2).t_3 \sim M$. There are two cases:

1. If M is deadlocked then t_1 and t_2 must be deadlocked.

2. If M is not deadlocked then there is an M' s.t. $M \stackrel{d}{\to} M'$ and M' is deadlocked. By the definition of PAD a single action d can only change t_1 or t_2, but not both. Thus either t_1 or t_2 must be deadlocked.

Thus if parallel composition occurs in a state that is reachable from Q, then all but one part of it must be deadlocked. Since Q is a single constant, parallel composition can only be introduced by PAD-rules. If such a rule has the form $(u \stackrel{x}{\to} u_1\|u_2) \in \Delta$ for some action x, then u_1 or u_2 must be deadlocked. W.r. let u_1 be deadlocked. However, the term $u_1.t$ for some term t is not necessarily deadlocked. Thus in Δ' we replace the rule $(u \stackrel{x}{\to} u_1\|u_2)$ by the rule $u \stackrel{x}{\to} u_2.u_1$. The new system is equivalent up to bisimulation. (We assume w.r. that u_2 cannot influence u_1. This means that there is no rule in Δ' whose left hand side is $v_2.v_1$ where v_2 is a nonempty suffix of u_2 and v_1 is a nonempty prefix of u_1. This can be achieved by renaming of constants in u_2 and Δ' if necessary.)
The other case where parallel composition occurs in a rule in Δ is when a rule has the form $u \stackrel{x}{\to} u_1.(u_2\|u_3).u_4$, where u_1 or u_4 can be ϵ. There are two cases:

1. If u_1 can terminate then the term $(u_2 \| u_3)$ can become active. Therefore u_2 or u_3 must be deadlocked. W.r. let u_2 be deadlocked. Then in Δ' we replace this rule by the rule $u \stackrel{x}{\to} u_1.u_3.u_2.u_4$. Note that u_2 is deadlocked, but $u_2.u_4$ is not necessarily deadlocked. (We assume w.r. that u_1 cannot influence u_3 and u_3 cannot influence u_2. This can be achieved by renaming of constants in u_1 and u_3 and Δ' if necessary.)

2. If u_1 cannot terminate then in Δ' we replace this rule by the equivalent rule $u \stackrel{x}{\to} u_1$.

Thus we get a new system Δ' that is equivalent to Δ up to bisimulation, but Δ' does not contain parallel composition. Thus, if the preconditions are satisfied, the (S, S)-PRS Δ' with initial state Q is bisimilar to $X \| A \| B$. This is the pushdown process that we are looking for. ∎

Definition 4.11 Let Δ be a (α, β)-PRS for $\alpha, \beta \in \{1, S, P, G\}$ and t_0 the initial state. The language generated by this system is the set of all sequences σ s.t. $\exists t.\ t_0 \stackrel{\sigma}{\to} t$ and t is deadlocked.

Lemma 4.12 *If a process t is bisimilar to a pushdown process then the language generated by t is a context-free language.*

Proof Directly from Def. 4.1 and the definition of pushdown processes. ∎

Lemma 4.13 *The Petri net of Def. 4.9 is not bisimilar to any PAD-process.*

Proof We assume the contrary and derive a contradiction. If there is a PAD-process that is bisimilar to the Petri net of Def. 4.9, then by Lemma 4.10 there is a pushdown process that is bisimilar to this Petri net. Then by Lemma 4.12 the Petri net of Def. 4.9 generates a context-free language L. By the definition of this Petri net L is

$$\{g^m c\sigma \mid m \geq 0 \wedge \sigma \in \{a,b\}^* \wedge \#_a\sigma = m+1 \wedge \#_b\sigma = m+1\} \cup$$

$$\{g^m d \mid m \geq 0\} \cup$$

$$\left\{ g^m c\sigma d \;\middle|\; \begin{array}{l} m \geq 0 \wedge \sigma \in \{a,b\}^* \wedge \\ \#_a\sigma \leq m+1 \wedge \#_b\sigma \leq m+1 \wedge \#_a\sigma + \#_b\sigma \leq 2m+1 \end{array} \right\}$$

It follows that $L \cap g^*ca^*b^* = \{g^m c a^{m+1} b^{m+1} \mid m \geq 0\}$. By applying the pumping lemma for context-free languages [HU79] it is easy to show that L is not context-free. Thus we have a contradiction. ∎

It follows that PAD and PAN are incomparable and PRS is strictly more general than PAD. By combining these results with the other results above we get the following theorem.

Theorem 4.14 *The PRS-hierarchy is strict with respect to bisimulation.*

5 The Reachability Problem

In this section we show that the reachability problem is decidable for PRS. Thus PRS are not Turing-powerful.

<u>Reachability</u>

Instance: A PRS Δ with initial state t_0 and a given state t.

Question: Is the state t reachable from t_0 ? Formally: Is there a sequence of actions σ s.t. $t_0 \stackrel{\sigma}{\rightarrow} t$?

For Petri nets reachability is decidable and *EXPSPACE*-hard [May84, Lip76]. Here we show that reachability is decidable for PRS by reducing the problem to the reachability problem for Petri nets. As the atomic actions are not important for reachability, we'll ignore them for the rest of this section and write just $t_1 \rightarrow t_2$ instead of $t_1 \stackrel{a}{\rightarrow} t_2$.

We prove the decidability of reachability in two steps. First we show that it suffices to decide the problem for a special class of PRS, the PRS in transitive normal form (see below). Then we solve the problem for this subclass of PRS.

Definition 5.1 For a PRS Δ and process terms $t, t' \in \mathcal{T}$ we define

$$t \succ^{\Delta} t' \ :\Longleftrightarrow \ \exists \sigma.\ t \stackrel{\sigma}{\rightarrow} t'$$

where σ is a sequence of applications of rules in Δ. If Δ is fixed, then we just write $t \succ t'$.

Δ is in *normal form* iff all rules in Δ are in normal form. A rule is in normal form if it has one of the following two forms:

Par-Rule $X_1 \| X_2 \rightarrow Y$ or $X \rightarrow Y_1 \| Y_2$ or $X \rightarrow Y$.

Seq-Rule $X_1.X_2 \rightarrow Y$ or $X \rightarrow Y_1.Y_2$ or $X \rightarrow Y$.

where X, Y, X_i, Y_i are process constants and Y can be ϵ.
The only rules that are both seq-rules and par-rules are of the form $X \to Y$.
The following relations $\succ_{par}^{\Delta}$ and $\succ_{seq}^{\Delta}$ are technicalities used in the proofs.

$$t \succ_{par}^{\Delta} t' \;\; :\Longleftrightarrow \;\; \exists \sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are par-rules from } \Delta$$
$$t \succ_{seq}^{\Delta} t' \;\; :\Longleftrightarrow \;\; \exists \sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are seq-rules from } \Delta$$

A PRS Δ is in *transitive normal form* iff it is in normal form and for all $X, Y \in \mathit{Const}$
$$X \succ^{\Delta} Y \;\Rightarrow\; (X \to Y) \in \Delta$$

Proposition 5.2 *Let Δ be a PRS in transitive normal form and t_1, t_2 process terms that do not contain the operator for sequential composition. It is decidable if $t_1 \succ_{par}^{\Delta} t_2$.*

Proof This follows directly from the decidability of the reachability problem for Petri nets [May84]. ■

The reachability problem for PRS is reducible to the reachability problem for PRS in normal form.

Lemma 5.3 *Let Δ be a PRS and $t_1, t_2 \in \mathcal{T}$.*
Then a PRS Δ' in normal form and terms t_1' and t_2' can be effectively constructed s.t. Δ', t_1' and t_2' use only constants from the finite set V' (with $\mathit{Const}(\Delta) \subseteq V' \subset \mathit{Const}$) and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$.

Proof For any rule $(u_1 \to u_2)$ in Δ let

$$\mathit{norm}(u_1 \to u_2) := \mathit{size}(u_1) + \mathit{size}(u_2)$$

Let k_i be the number of rules $(u_1 \to u_2)$ in Δ that are not in normal form and $\mathit{norm}(u_1 \to u_2) = i$. Let n be the maximal i s.t. $k_i \neq 0$. (n exists because Δ is finite). We define $\mathit{Norm}(\Delta) := (k_n, k_{n-1}, \ldots, k_1)$. These norms are ordered lexicographically. Δ is in normal form iff $\mathit{Norm}(\Delta) = (0, \ldots, 0)$. Now we describe a procedure that transforms Δ into a new PRS Δ' and terms t_1, t_2 into t_1', t_2' s.t. $\mathit{Norm}(\Delta') <_{lex} \mathit{Norm}(\Delta)$ and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$.

Remember that sequential composition is left-associative. This means that the term $X.Y.Z$ is $(X.Y).Z$. It has the subterms X, Y, Z and $X.Y$, but not $Y.Z$. However, the term $X.(Y\|Z)$ has a subterm $Y\|Z$.

If $Norm(\Delta) \neq (0, \ldots, 0)$ then there is a rule in Δ that is not in normal form. Take a non-constant subterm t of this rule and replace every subterm t in Δ and in t_1 and t_2 by a new constant X. Then add two rules $X \to t$ and $t \to X$. This yields a new set of rules Δ' and t_1' and t_2'. By the definition of *Norm* and *size* we get $Norm(\Delta') <_{lex} Norm(\Delta)$. The constant X serves as an abbreviation for the term t. There are only two problems:

1. A rule is applicable to a subterm of t, but not to X. For example $t = Y.Z.W$ and there is a rule $Y.Z \to V$. In this case the rule $X \to t$ must be applied first. So the term X can be rewritten to $V.W$ in two steps.

2. During the rewriting a subterm t is created. However, the rule that contains t as a subterm on the left side is no longer applicable, because the subterm t has been replaced by X. For example let $t = Y\|Z$, the initial state is $(W\|Z).W$ and there are rules $W \to Y$ and $(Y\|Z).W \to V$. By the above algorithm the rule $(Y\|Z).W \to V$ has been transformed into $X.W \to V$ and rules $X \to Y\|Z$ and $Y\|Z \to X$ have been added. The initial state $(W\|Z).W$ can be rewritten to $(Y\|Z).W$, but now the changed rule $X.W \to V$ is not applicable. However, by applying the new rule $Y\|Z \to X$ first we get $X.W$ and can finally rewrite the term to V.

Thus we get

$$t_1 \succ^{\Delta} t_1 \iff t_1' \succ^{\Delta'} t_2'$$

By repeating this algorithm we finally get a set of rules Δ'' and terms t_1'' and t_2'' s.t. $Norm(\Delta'') = (0, \ldots, 0)$ and

$$t_1 \succ^{\Delta} t_1 \iff t_1'' \succ^{\Delta''} t_2''$$

Δ'' is in normal form. ∎

The following lemma will be used to prove the correctness of the algorithm in Lemma 5.5.

Lemma 5.4 *Let Δ be a PRS in normal form. If there are constants X, Y s.t. $X \succ^{\Delta} Y$ and $(X \to Y) \notin \Delta$, then there are also constants X', Y' with $(X' \to Y') \notin \Delta$ and $X' \succ^{\Delta}_{par} Y'$ or $X' \succ^{\Delta}_{seq} Y'$.*

Proof It follows from the preconditions that we can choose a pair of constants X', Y' s.t. $(X' \to Y') \notin \Delta$ and $X' \xrightarrow{\sigma} Y'$ for a sequence σ of minimal length. More precisely the length of σ is minimal over the choice of X', Y' and σ.

Now we show that $X' \succ^{\Delta}_{par} Y'$ or $X' \succ^{\Delta}_{seq} Y'$. We do this by assuming the contrary and deriving a contradiction. We say that a rule is trivial if it has the form $(X'' \to Y'')$. We assume that σ contains both seq-rules and par-rules that are nontrivial. There are two cases:

1. The last nontrivial rule in σ is a par-rule. If a seq-rule $Z_1 \to Z_2.Z_3$ occurs in σ then there is a subsequence σ' of σ and a constant Z_4 s.t. $Z_2.Z_3 \xrightarrow{\sigma'} Z_4$. This contradicts the minimality of the length of σ.
2. The last nontrivial rule in σ is a seq-rule. This seq-rule must have the form $Z_1.Z_2 \to Z$. The first nontrivial par-rule that occurs in σ must have the form $Z' \to Z'_1 \| Z'_2$. Then there is a subsequence σ' of σ and a constant Z'' s.t. $Z' \xrightarrow{\sigma'} Z''$. This contradicts the minimality of the length of σ.

Thus σ consists either only of applications of par-rules (and thus $X' \succ^{\Delta}_{par} Y'$) or only of seq-rules (and thus $X' \succ^{\Delta}_{seq} Y'$). ∎

Lemma 5.5 *Let Δ be a PRS in normal form. Then a PRS Δ' in transitive normal form can be effectively constructed s.t.*

$$\forall t_1, t_2 \in \mathcal{T}.\ t_1 \succ^{\Delta'} t_2 \iff t_1 \succ^{\Delta} t_2$$

Proof It suffices to find all pairs of constants X, Y s.t. $X \succ^{\Delta} Y$ and to add the rules $(X \to Y)$ to Δ. By Lemma 5.4 it suffices to check $X \succ^{\Delta}_{par} Y$ and $X \succ^{\Delta}_{seq} Y$. This is decidable because of Proposition 5.2 and the decidability of the reachability problem for pushdown processes (see [BEM97]). Lemma 5.4 basically says that while there are new rules to add we can find at least one to add.

The algorithm is as follows:

$\Delta' := \Delta$; flag := true;
While flag **do**
  flag := false;
  For every pair of constants X, Y with $(X \to Y) \notin \Delta'$ **do**
    If $X \succ^{\Delta'}_{par} Y$ or $X \succ^{\Delta'}_{seq} Y$ **then** $(\Delta' := \Delta' \cup (X \to Y)$; flag := true) **fi**;
  od;
od;

∎

Theorem 5.6 *The reachability problem is decidable for PRS. The complexity is polynomially equivalent to reachability for Petri nets.*

Proof Let Δ be a PRS and $t_1, t_2 \in \mathcal{T}$. The question is if $t_1 \succ^{\Delta} t_2$.

We construct a new PRS Δ' by adding new constants X_1 and X_2 and rules $X_1 \to t_1$ and $t_2 \to X_2$. It follows that $t_1 \succ^{\Delta} t_2 \;\Leftrightarrow\; X_1 \succ^{\Delta'} X_2$. Then we use Lemma 5.3 and transform Δ' into a PRS Δ'' in normal form. Normally the terms X_1, X_2 would also change in this transformation, but since they are single constants they stay the same. This procedure adds at most $2k$ new rules, where k is the number of non-constant strict subterms of rules in Δ. Thus $k = \mathcal{O}(n^2)$ and $size(\Delta')$ is polynomial in $size(\Delta)$. We get $t_1 \succ^{\Delta} t_2 \;\Leftrightarrow\; X_1 \succ^{\Delta''} X_2$. Then we use Lemma 5.5 to transform Δ'' into a PRS Δ''' in transitive normal form. It follows that $t_1 \succ^{\Delta} t_2 \;\Leftrightarrow\; X_1 \succ^{\Delta'''} X_2$. Since $|Const(\Delta)| = \mathcal{O}(n)$ there are $\mathcal{O}(n^2)$ pairs of constants. Thus the algorithm of Lemma 5.5 uses $\mathcal{O}(n^2)$ instances of the reachability problem for Petri nets and for pushdown processes in every instance of the loop. The loop is done at most $\mathcal{O}(n^2)$ times. Thus it uses at most $\mathcal{O}(n^4)$ instances of the reachability problem for Petri nets and pushdown processes. Since Δ''' is in transitive normal form we have

$$t_1 \succ^{\Delta} t_2 \;\Leftrightarrow\; X_1 \succ^{\Delta'''} X_2 \;\Leftrightarrow\; (X_1 \to X_2) \in \Delta'''$$

The condition $(X_1 \to X_2) \in \Delta'''$ is trivial to check.

The reachability problem for pushdown processes is polynomial [BEM97]. The algorithm for PRS uses only polynomially many instances of Petri net reachability. Since PRS are more general than Petri nets, it follows that reachability for PRS is polynomially equivalent to Petri net reachability. ∎

6 The Reachable Property Problem

In the previous section the problem was if one given state is reachable. Here we consider the question if there is a reachable state that has certain properties. We call this problem the *reachable property problem*. Unlike for reachability, the atomic actions are important for this problem. Properties are described by state formulae that have the following syntax:

$$\Phi := a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2$$

The denotation $[\![\Phi]\!]$ of a state formula Φ is a (possibly infinite) set of process terms.

$$\begin{array}{rcl} [\![a]\!] & := & \{t \mid \exists t'.\, t \xrightarrow{a} t'\} \\ [\![\neg\Phi]\!] & := & \mathcal{T} - [\![\Phi]\!] \\ [\![\Phi_1 \wedge \Phi_2]\!] & := & [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\ [\![\Phi_1 \vee \Phi_2]\!] & := & [\![\Phi_1]\!] \cup [\![\Phi_2]\!] \end{array}$$

To simplify the notation we use sets of actions. Let $A := \{a_1, \ldots, a_k\} \subseteq Act$.

$$\begin{array}{rcl} [\![A]\!] & := & [\![a_1]\!] \cap \ldots \cap [\![a_k]\!] \\ [\![-A]\!] & := & [\![\neg a_1]\!] \cap \ldots \cap [\![\neg a_k]\!] \end{array}$$

By transformation to disjunctive normal form every state-formula Φ can be written as $(A_1 \wedge -B_1) \vee \ldots \vee (A_n \wedge -B_n)$, where $A_i, B_i \subseteq Act$.
We consider the question if there is a reachable state that satisfies a given state formula. To express this problem, we define another operator.

$$[\![\Diamond\Phi]\!] := \{t \mid \exists \sigma, t'.\ t \xrightarrow{\sigma} t' \in [\![\Phi]\!]\}$$

Note that state-formulae do not contain the operator $\Diamond$. Let $t \in \mathcal{T}$ be a process term. For $t \in [\![\Phi]\!]$ we also write $t \models \Phi$.

<u>REACHABLE PROPERTY PROBLEM</u>

Instance: A PRS Δ with initial state t_0 and a state-formula Φ.
Question: $t_0 \models \Diamond\Phi$?

We prove the decidability of the reachable property problem for PRS in two steps. First we show that it suffices to solve the problem for PRS in transitive normal form.

Lemma 6.1 *Let Δ be a PRS that uses only constants from the finite set $Const(\Delta) \subset Const$, and let $t_0 \in \mathcal{T}$ be a process term.*

Then a PRS Δ' in normal form and a term t_0' can be effectively constructed s.t. for every state formula Φ, $t_0 \models \Diamond\Phi$ with respect to Δ iff $t_0' \models \Diamond\Phi$ with respect to Δ'.

Proof We use the same algorithm to transform Δ and t as in Lemma 5.3. The new rules that are added are labeled with the new (silent) action τ, that doesn't occur in Φ.

The only problem that remains is that if a subterm t is replaced by a new constant X, then X does not enable the same actions as t. Thus, for example, the new system might satisfy a formula $\Diamond(\neg a)$, although the original doesn't.

The solution is as follows: Compute the set of actions $\{b_1, \ldots, b_m\}$ that are enabled by the term t in w.r.t. Δ. Then add new rules $X \stackrel{b_1}{\to} X, \ldots, X \stackrel{b_m}{\to} X$. This must be done after every step where a subterm a replaced by a constant.

Then the new system Δ', t_0' satisfies exactly the same formulae $\Diamond\Phi$ as the old one. ∎

Lemma 6.2 *Let Δ be a PRS in normal form. Then a PRS Δ' in transitive normal form can be effectively constructed s.t. for every term t and every state-formula Φ, $t \models \Phi$ w.r.t Δ iff $t \models \Phi$ w.r.t Δ'.*

Proof We use the same algorithm as in Lemma 5.5. The only difference is that we label the newly added rules with the special (silent) action τ that does not occur in any state-formula. ∎

Remark 6.3 *By Lemma 6.1 and Lemma 6.2 it follows that it suffices to solve the reachable property problem for PRS in transitive normal form. Let there be a PRS Δ in transitive normal form with initial state t_0 and Φ a state-formula. The problem is if $t_0 \models \Diamond\Phi$. As Φ can be transformed into disjunctive normal form and*

$$t \models \Diamond(\Phi_1 \vee \Phi_2) \iff t \models \Diamond(\Phi_1) \vee t \models \Diamond(\Phi_2)$$

it suffices to show decidability for formulae of the form $\Diamond(A \wedge -B)$, where $A, B \subseteq Act$.

The following definition and lemma by Jančar [Jan90] are used to show the effectiveness of the procedures *check* and *check'* that are used to show the decidability of the reachable property problem.

Definition 6.4 For a given Petri net N the set L_N of formulae is defined as follows:

- There is one variable M that stands for a marking of the net.
- A term is either
  - a term $M(p)$ where p is a place, or
  - a constant $c \in \mathbb{N}$, or
  - of the form $t_1 + t_2$.
- A formula is either
  - an atomic formula $t_1 < t_2$ or $t_1 \leq t_2$, where t_1, t_2 are terms, or
  - of the form $f_1 \& f_2$ where f_1, f_2 are formulae.

For a concrete marking M, $f(M)$ denotes the instance of f with this M. The semantics is natural.

Lemma 6.5 *([Jan90])*
For a Petri net N with initial marking M_0 it is decidable if there is a reachable marking M s.t. $f(M)$.

Definition 6.6 Let $C \subset Const$ and $t \in P$. Let h be a function s.t. $h(C, t)$ is true iff t contains only constants from C and false otherwise.
Let Δ be a PRS in transitive normal form, $X \in Const$ and let A, A', B be finite sets of actions. Let j be a mapping $j : 2^A \mapsto 2^{Const}$.
$check(X, j, A', B)$ iff there exists a $t \in P$ s.t. $X \succ_{par}^{\Delta} t$ and

$$(t = t' \| \|_{(C \in 2^A)} t_c) \ \wedge \ t' \models (A' \wedge -B) \ \wedge \ \bigwedge_{C \in 2^A} h(j(C), t_c)$$

and the additional constraint that $t \in Const \Rightarrow t' = \epsilon$.
$check'(X, A, B)$ iff there exists a $t \in P$ s.t. $X \succ_{par}^{\Delta} t$ and $t \notin Const$ and

$$t \models (A \wedge -B)$$

Lemma 6.7 *The functions check and check′ are decidable.*

Proof Directly from Lemma 6.5, because par-rules correspond to Petri net transitions. ■

Definition 6.8 The function *snd* returns the nesting-depth of sequential composition in a process term.

$$
\begin{aligned}
snd(\epsilon) &:= 0 \\
snd(X) &:= 0 \\
snd(t_1 \| t_2) &:= max(snd(t_1), snd(t_2)) \\
snd(t_1.t_2) &:= max(snd(t_1)+1, snd(t_2))
\end{aligned}
$$

Definition 6.9 Let Δ be a PRS in transitive normal form, $X \in Const$, $n \in \mathbb{N}$ and A, B finite sets of actions.
Let $reach(X, n, A, B)$ be true iff there exists a term t s.t. $t \notin Const$, $X \succ^{\Delta} t$, $t \models (A \wedge -B)$ and the nesting-depth of sequential composition in t is at most n, i.e. $snd(t) \leq n$.
The function *reachseq* is defined like *reach*, except that the first rule applied to X must be a seq-rule of the form $X \overset{a}{\to} Y.Z$ and Z is never changed afterwards. (This implies that *reachseq* is only defined for $n \geq 1$.)

Now we describe recursive algorithms for *reach* and *reachseq*.

$reach(X, n, A, B)$
  case $n = 0$:
    return$(check'(X, A, B))$;
  case $n > 0$:
    for every mapping $j : 2^A \longmapsto 2^{Const}$ and every $A' \subseteq A$
      if $A' \cup \bigcup_{C \in 2^A \wedge j(C) \neq \emptyset} C = A$ then
        if $check(X, j, A', B)$ then
          if $\bigwedge_{C \in 2^A} \left(\bigwedge_{X' \in j(C)} reachseq(X', n, C, B) \right)$ then return(true);
    return(false);

The function *reachseq* is only defined for arguments $n \geq 1$.

```
1  reachseq(X, n, A, B)
2      for every X → Y.Z
3          if reach(Y, n − 1, A, B) then return(true);
4          for every Y ≻ W
5              if W.Z ⊨ (A ∧ −B) then return(true);
6      return(false);
```

Remark: Note that seq-rules of the form $X.Y \to Z$ (sequential composition on the left side) are almost never used in the algorithm. The only exception is in line 5 of the function *reachseq* where they might be needed to enable some action in A. The reason why they are not used anywhere else is because they are not needed since Δ is in transitive normal form.

Lemma 6.10 *The above algorithms are correct and effective implementations of the functions reach and reachseq.*

Proof By induction on n.

Base case: For *reach* the base case is $n = 0$. The correctness follows immediately from the definition of the function *check'* and Lemma 6.7. Note that no seq-rules are used, because Δ is in transitive normal form.

For *reachseq* the base case is $n = 1$. By definition the first rule application must have the form $X \to Y.Z$ (as in line 2). Line 3 deals with the case that Y alone develops into some term $t \in P$ that satisfies $A \wedge -B$ and Z does not play a role. However, t must not be a single constant, because otherwise it might be able to interact with Z via a seq-rule. The function *reach* is called with argument $n = 0$ and just calls the function *check'* which guarantees that t is not a single constant. In line 4,5 we consider the case that Y is rewritten to a single constant W (possibly Y itself) s.t. $W.Z \models (A \wedge -B)$. Since Δ is in transitive normal norm the condition in line 4 is trivial to check: either $W = Y$ or $(Y \to W) \in \Delta$. Line 5 is there to deal with the case that some seq-rule of the form $W.Z \xrightarrow{a} Z'$ is needed to enable some action a in A.

Step: In the function *reach* we split the set of actions A into subsets. The special subset A' are the actions that should become enabled after applying only par-rules to X. The other subsets of actions are assigned sets of constants by the function j. These constants require further application of seq-rules. By the function *check* we test for the reachability of a state t with

$$(t = t' \| \|_{(C\in 2^A)} t_c) \ \wedge \ t' \models (A' \wedge -B) \ \wedge \bigwedge_{C\in 2^A} h(j(C), t_c)$$

and the additional constraint that $t \in Const \Rightarrow t' = \epsilon$.

The constants in the terms t_c require further applications of seq-rules. The additional constraint ensures that at least one t_c is not ϵ or t' is not a constant. This ensures that the reachable state that finally satisfies $(A \wedge -B)$ is not a constant. (The applications of seq-rules in *reachseq* always yield non-constant terms.)

Now for the correctness of *reachseq*: By definition the first rule application must have the form $X \rightarrow Y.Z$ (as in line 2). Line 3 deals with the case that Y alone develops into some term $t \in P$ that satisfies $A \wedge -B$ and Z does not play a role. However, t must not be a single constant, because otherwise it might be able to interact with Z via a seq-rule. The function *reach* guarantees this and by induction hypothesis the correctness follows. In line 4,5 we consider the case that Y is rewritten to a single constant W (possibly Y itself) s.t. $W.Z \models (A \wedge -B)$. Since Δ is in transitive normal norm the condition in line 4 is trivial to check: either $W = Y$ or $(Y \rightarrow W) \in \Delta$. Line 5 is there is deal with the case that some seq-rule of the form $W.Z \xrightarrow{a} Z'$ is needed to enable some action a in A. ■

Now we show that it suffices to consider terms with bounded nesting-depth of sequential composition.

Lemma 6.11 *Let Δ be a PRS in transitive normal form, $X \in Const(\Delta)$ and $A, B \subseteq Act(\Delta)$.*
Then $X \models \Diamond(A \wedge -B)$ iff there is a term t s.t. $X \succ^{\Delta} t$, $t \models (A \wedge -B)$ and $snd(t) \leq |A| * |Const(\Delta)|$.

Proof X is transformed into t by applying rewrite rules from Δ. The nesting-depth of sequential composition is only increased when a seq-rule of the form $Z \rightarrow Z'.Z''$ is applied to some constant Z which is a subterm of an intermediate term. In the end Z should be rewritten to a subterm t'

of of t that satisfies a part of the formula $(A \wedge -B)$. Thus this subterm Z is required to satisfy $\Diamond(A' \wedge -B)$ for some $A' \subseteq A$. Let a chain be a sequence of applications of rewrite rules s.t. every rule rewrites at least part of the term which was introduced by the previous one. Consider a chain and the sequence of constants Z_i in it to which seq-rules are applied and the sequence of formulae $\Diamond(A'_i \wedge -B)$ that the Z_i are required to satisfy. These subsets A'_i can never get bigger in a chain. Furthermore, if they get smaller they must be subsets of previous ones. Therefore in any chain at most $|A|$ different formulae $\Diamond(A'_i \wedge -B)$ must be satisfied by constants Z_i to which seq-rules are applied. We can assume that in any chain no constant (to which a seq-rule is applied) appears twice with the same formula, because this means that a constant has been rewritten to a term containing this constant without making any progress in the formula. It follows that any chain contains at most $|A| * |Const(\Delta)|$ applications of seq-rules, because there are only $|Const(\Delta)|$ different constants. Thus we get $snd(t) \leq |A| * |Const(\Delta)|$. ∎

Theorem 6.12 *The reachable property problem is decidable for PRS.*

Proof An instance is given by a PRS Δ, an initial state t_0 and a state-formula Φ. The question is if $t_0 \models \Diamond\Phi$. Without restriction we can assume that t_0 is a single constant X. (Otherwise just add a rule $X \xrightarrow{\tau} t_0$.) By Lemma 6.1 and Lemma 6.2 the problem can be reduced to a problem for PRS in transitive normal form. By Remark 6.3 the problem can be reduced to problems for formulae of the form $\Diamond(A \wedge -B)$. If $X \models \Diamond(A \wedge -B)$ then there are two cases:

1. X can reach a term $Y \in Const$ s.t. $Y \models (A \wedge -B)$. This can be easily checked, because Δ is in transitive normal form. For every constant $Y \in Const$ check if $(X \rightarrow Y) \in \Delta$ and $Y \models (A \wedge -B)$. Also check if $X \models (A \wedge -B)$.

2. X can reach a term $t \notin Const$ s.t. $t \models (A \wedge -B)$. By Lemma 6.11 there is such a t with $snd(t) \leq |A| * |Const(\Delta)|$. Thus the condition can be checked by computing $reach(X, |A| * |Const(\Delta)|, A, B)$. By Lemma 6.10 this can be done with the algorithms given above.

$X \models \Diamond(A \wedge -B)$ iff one of those checks yields a positive answer. ∎

Remark 6.13 *This result can also be used to decide deadlock-freedom. Let Δ be a PRS with initial state t_0 and $Act(\Delta)$ the (finite!) set of actions used in Δ. A deadlock is reachable iff $t_0 \models \Diamond(-Act(\Delta))$. Thus the system is deadlock-free iff $t_0 \not\models \Diamond(-Act(\Delta))$.*

7 Conclusion

The algorithms for the reachability problem and the reachable property problem for PRS rely on the reachability problem for Petri nets, which has a high complexity (*EXPSPACE*-hard [May84, Lip76]). So it might seem that they are not applicable in practice because of their very high complexity. However, there are three arguments in their favor:

1. In many examples the system is not very large and the structure of the Petri nets that are contained in them is often simple.

2. In a large PRS there may be many Petri nets as substructures, but often each of these Petri nets is quite small. These Petri nets are either not connected with each other at all, or their influence on each other is very limited. Thus they yield small subproblems that can be solved in acceptable time.

3. Finally, the reachability problem for Petri nets has been studied for many years and ways of dealing with it have been developed. There are semi-decision procedures that give yes/no/don't know answers in acceptable time [CH78, Mur89, ME96]. These algorithms mostly use constraints to represent sets of states and approximate the behavior of the system.

Therefore the algorithms of Section 5 and Section 6 can still be useful in practice to verify systems that are modeled with PRS.

Process Rewrite Systems (PRS) is a very expressive model of infinite-state concurrent systems that subsumes PAN, PAD, Petri nets, PA-processes, pushdown processes, BPP and BPA. PRS extends Petri nets by introducing an operator for sequential composition. This can be seen as the possibility to call subroutines. The calling of subroutines is already possible in PAN-processes. However, there is a major difference: In PAN subroutines that terminate have no effect on their caller, while in PRS subroutines can return a value to the caller when they terminate. This is an important aspect in modeling real programs. Thus PRS-processes can be used to model systems that exceed the bounds of the expressiveness of Petri nets and PAN.

PRS is a very general model for concurrent systems. Thus model checking with many temporal logics (EF, CTL, LTL, linear time μ-calculus, modal μ-calculus) is undecidable for it. This is because EF is undecidable for Petri nets [Esp97, BE97], CTL is undecidable for BPP [EK95] and LTL and the linear time μ-calculus are undecidable for PA-processes [BH96]. However, PRS is not Turing powerful, since reachability is still decidable.

Finally, it should be noted that PRS are (roughly) equivalent to ground AC rewrite systems (i.e. rewrite systems without substitution, but with an associative and commutative operator). The general idea is that e.g. a ground AC term $Z(X+Y)$ (where '+' is the associative and commutative operator) corresponds to a PRS-term $(X\|Y).Z$ and vice versa.

Acknowledgments: I thank Michaël Rusinowitch and Javier Esparza for helpful discussions and three anonymous referees for their detailed comments.

References

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[BE97] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 5, 1997.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages*. ACM-Press, 1978.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[Dic13] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[Jan90] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Comuter Science*, 74:71–93, 1990.

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

[Lip76] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[May84] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460, 1984.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In Martin Abadi and Takayasu Ito, editors, *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*. Springer Verlag, 1997.

[May97b] R. Mayr. Model checking PA-processes. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[May97c] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 7, 1997. Proceedings of Expressiveness in Concurrency (EXPRESS'97).

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.

[ME96] S. Melzer and J. Esparza. Checking system properties via integer programming. In H.R. Nielson, editor, *Proc. of ESOP'96*, volume 1058 of *Lecture Notes in Computer Science*, pages 250–264. Springer Verlag, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mol96] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[Mur89] T. Murata. Petri nets: Properties, analysis und applications. *Proc. of the IEEE*, 77(4):541–580, 1989.