Electronic Notes in Theoretical Computer Science 7 (1997)
URL: http://www.elsevier.nl/locate/entcs/volume7.html 21 pages

Process Rewrite Systems

Richard Mayr

Institut für Informatik
Technische Universität München
Arcisstr. 21, D-80290 München, Germany
e-mail: mayrri@informatik.tu-muenchen.de
Web: http://www7.informatik.tu-muenchen.de/~mayrri

Abstract

Many formal models for infinite state concurrent systems can be expressed by special classes of rewrite systems. We classify these models by their expressiveness and define a hierarchy 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. PRS are not Turing-powerful, because the reachability problem is still decidable. It is even decidable if there is a reachable state that satisfies certain properties that can be encoded in a simple logic.

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 [8,11] 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 every single one can be seen as a special subclass of rewrite systems. Basically, the rewriting formalism is first order prefix-rewrite systems on process terms without substitution and modulo commutativity of parallel 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 2.7). As PRS is a very expressive model, model checking with any temporal logic (except Hennesy-Milner logic) is undecidable for it (see Section 6). 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 and describe a strict hierarchy of subclasses of it. Section 3 describes an example of a system that is modeled with PRS. In Section 4 we show that the reachability problem is decidable for PRS. Section 5 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 ***(Process Rewrite Systems)*** *Let $Act = \{a, b, \ldots\}$ be a countably infinite set of atomic actions and $Var = \{X, Y, Z, \ldots\}$ be countably infinite set of process variables. The process terms $\mathcal{T}$ that describe the states have the form*

$$P ::= \epsilon \mid X \mid P_1.P_2 \mid P_1 \| P_2$$

where "$\|$" means parallel composition and "." means sequential composition. ***Convention:*** *We always work with equivalence classes of terms modulo commutativity of parallel composition, thus $P_1 \| P_2 = P_2 \| P_1$. Also we define that $\epsilon.P = P$ and $P \| \epsilon = P$. W.l.o.g. let sequential composition be left-associative. So if we write $t_1.t_2$, then we mean that t_2 is a single variable or a parallel composition.*

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

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

Since Δ is finite, the generated LTS is finitely branching. Also every single Δ uses only a finite subset $Var(\Delta) \subset Var$ of variables and only a finite subset $Act(\Delta) \subset Act$ of atomic actions. Thus for every Δ only finitely many of the generated transition relations $\stackrel{a_i}{\rightarrow}$ 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.)

The relation $\stackrel{a}{\rightarrow}$ is generalized to sequences of actions in the standard way. Sequences are denoted by σ. Without restriction we can assume that the initial state of a system is described by a term consisting of a single variable. Note that 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 in Δ with the same term on the left hand side. The one that is applied is chosen nondeterministically.

Many common models of systems fit into this general 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 2.2 ***(Classes of process terms)***
We use the following classes of process terms.

1 *Terms consisting of a single process variable like X.*

S *Terms consisting of a sequential composition of process variables like $X.Y.Z$.*

P *Terms consisting of a parallel composition of process variables like $X \| Y \| Z$.*

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

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 $S \cap P = 1$.

Definition 2.3 *Let $\alpha, \beta \in \{1, S, P, G\}$. A (α, β)-PRS is a PRS where for every rewrite rule $(l \rightarrow r) \in \Delta$ we have $l \in \alpha$ and $r \in \beta$. A (G, G)-PRS is often simply called PRS.*

(α, β)-PRS where α is more general 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 sides 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 variable). Thus one may as well use a (S, S)-PRS. So we restrict our attention to (α, β)-PRS with $\alpha \subseteq \beta$.

Figure 2 shows a graphical description of the hierarchy of (α, β)-PRS.

(i) A $(1, 1)$-PRS is a finite state system. Every process variable corresponds to a state and the state space is bounded by $|Var(\Delta)|$. Every finite state

system can be encoded as a $(1,1)$-PRS.

(ii) $(1,S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [3,6]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.

(iii) It is easy to see that pushdown automata can be encoded as a subclass of (S,S)-PRS (with at most two variables on the left side of rules). Caucal [4] 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.

(iv) (P,P)-PRS are equivalent to Petri nets. Every variable corresponds to a place in the net and the number of occurrences of a variable 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.

(v) $(1,P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [3,6]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [5].

(vi) $(1,G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication [11,8].

(vii) (P,G)-PRS are called *PAN-processes* in [10]. It is the smallest common generalization of Petri nets and PA-processes and is strictly more general than both of them (i.e. PAN can describe all Chomsky-2 languages while Petri nets cannot).

(viii) (S,G)-PRS are the smallest common generalization of pushdown processes and PA-processes. We call them *PAD* (PA + PD).

(ix) Finally, there is the most general case of (G,G)-PRS (simply called PRS). They subsume all the previously mentioned classes.

The intuition is that PAN extends Petri nets by allowing sequential composition, which can be seen as the possibility to call subroutines. PRS extends PAN with the possibility that a subroutine can affect the behavior of the caller after its termination. This can be interpreted as returning a value to the caller.

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 an operational semantics described by bisimulation equivalence. Bisimilarity is a finer equivalence than language equivalence and is defined as follows:

Definition 2.4 *A binary relation R over the states of a labeled transition*

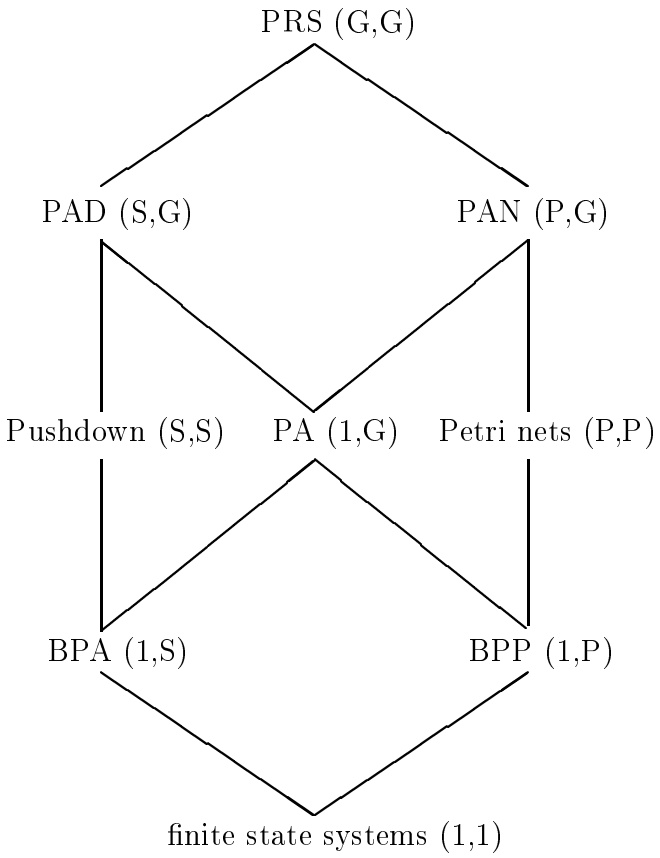

Fig. 1. The PRS-hierarchy

system (for short: LTS) is a bisimulation *iff*

$$\forall (s_1, s_2) \in R \; \forall a \in Act.\; (s_1 \xrightarrow{a} s_1' \;\Rightarrow \exists s_2 \xrightarrow{a} s_2'.\; s_1' R s_2') \wedge (s_2 \xrightarrow{a} s_2' \;\Rightarrow \exists s_1 \xrightarrow{a} s_1'.\; s_1' R s_2')$$

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 called* ***bisimulation equivalence****. It is denoted by $\sim$.*

A class of processes A is more general than a class of processes B with respect to bisimulation iff $B \subset A$ and there is a A-process that is not bisimilar to any B-process. It has already been established in [2,13] 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. Consider the following pushdown system:

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

The execution sequences of this system are as follows: First it does a sequence in $\{a, b\}^*$ and then one of two things:

(i) A c, the sequence in reverse and finally another a

(ii) A d, the sequence in reverse and finally another b

Lemma 2.5 *This system is not bisimilar to any PAN-process.*

Proof. Assume the contrary. Petri nets cannot record a sequence of actions and replay them in reverse, because this requires a stack. Thus the PAN-process must contain a context-free subprocess that does it. In PAN (unlike in PAD or PRS) this context-free subprocess cannot communicate with the rest of the process. Therefore the action c or d that indicate the start of the reverse sequence must also occur in the context-free subprocess and cannot affect the rest of the process. Unlike a pushdown process the context-free subprocess cannot remember if it did c or d, but this is important, because it affects the last action. It can only enable either both a and b or none as last action. In either case the systems are not bisimilar, which is a contradiction.□

It follows directly that this system is not bisimilar to any PA-process either. However, as PAD and PRS subsume pushdown processes, it is a PAD or PRS-process. Thus PAD is strictly more general then PA and PRS is strictly more general than PAN. As PAD subsumes BPP, which is incomparable to pushdown systems, it is also more general than pushdown processes (PDA).

Proposition 2.6 *Consider the Petri net*

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

This net is not bisimilar (not even language equivalent) to any PAD-process.

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

Theorem 2.7 *The PRS-hierarchy is strict with respect to bisimulation.*

3 Example

In this section we describe a small example of a system that is modeled with PRS. The system is a parallel program that recursively computes a boolean value. First we write the program in a PASCAL-like pseudo-code.

function f(x : data) : boolean;
var x_1, x_2 : data;
var b_1, b_2 : boolean;
begin
  if $size(x) \leq 2$ **then** return(Q(x)) **fi;** /* Q is some predicate */
  $x_1 := P_1(x, 1)$; /* Splitting into subproblems */
  $x_2 := P_1(x, 2)$; /* P_1 somehow modifies x */
  $x_3 := P_1(x, 3)$;
  $b_1 := h(x_1)$ || $b_2 := h(x_2)$; /* Parallel call */
  if (b_1 *or* b_2) /* if at least one was successful */
  then
    return($f(x_3)$); /* apply f to the new instance */
  else
    return(false);
  fi;
end;

function h(x : data) : boolean;
begin
  $x_1 := P_2(x, 1)$; /* Splitting into subproblems */
  $x_2 := P_2(x, 2)$; /* P_2 somehow modifies x */
  $x_3 := P_2(x, 3)$;
  /* parallel call with different instances */
  $b_1 := f(x_1) \parallel b_2 := f(x_2) \parallel b_3 := f(x_3)$;
  if (b_1 *and* b_2 *and* b_3) /* if all are successful */
  then
    print("Now processing ",x_1, x_2, x_3);
    return(true);
  else
    return(false);
  fi;
end;

Of course we cannot model the whole program in PRS, because PRS is not Turing-powerful. However, we can accurately model the basic control structure. An instance of problem $f(x)$ (function f, data x) will be described by the process variable X. An instance of problem $h(x)$ (function h, data x) will be described by the process variable Z. We also have to describe how to handle booleans. Let variable T stand for $true$ and F for $false$. The rules for

conjunction are

$$T\|T \xrightarrow{and} T \quad T\|F \xrightarrow{and} F \quad F\|T \xrightarrow{and} F \quad F\|F \xrightarrow{and} F$$

In this context the variables T, F are always interpreted conjunctively. In order to be able to enforce a disjunctive interpretation we define new variables to stand for the same boolean values. Let variable R (right) stand for $true$ and W (wrong) stand for $false$. The rules for disjunction are

$$R\|R \xrightarrow{or} R \quad R\|W \xrightarrow{or} R \quad W\|R \xrightarrow{or} R \quad W\|W \xrightarrow{or} W$$

Now we describe the rules for the program:

(1) $X \xrightarrow{true} T$

(2) $X \xrightarrow{false} F$

(3) $P_1 \xrightarrow{prepare_1} \epsilon$

(4) $X \xrightarrow{decomp_1} P_1.(Z\|Z).X$

(5) $W.X \xrightarrow{stop} F$

(6) $R.X \xrightarrow{nextstep} X$

(7) $P_2 \xrightarrow{prepare_2} \epsilon$

(8) $Z \xrightarrow{decomp_2} P_2.(X\|X\|X).Y$

(9) $F.Y \xrightarrow{result\ no} W$

(10) $T.Y \xrightarrow{result\ ok} G.R$

(11) $G \xrightarrow{actions} \epsilon$

(1) X describes the main program that solves an instance of the problem. If the instance is small enough then the result is clear. In this case it is $true$.

(2) In this case it is $false$.

(3) P_1 stands for some computations that are necessary to decompose the problem X.

(4) In this case the problem is decomposed into smaller problems. First we do some preparation P_1. Then we solve two independent instances of a problem $(h(x_1), h(x_2))$ described by Z. This can be done in parallel. The two results are interpreted disjunctively. If one of them is true, then we solve a (smaller) instance of the main problem X. Otherwise we return $false$.

(5) If the previous result was W $(wrong)$, then there is no reason to go on. The result is F $(false)$.

(6) If the previous result was R $(right)$, then the result only depends on the smaller instance of the main problem X ($f(x_3)$ in the example).

(7) P_2 stands for some computations that are necessary to decompose the problem Z.

(8) The problem Z is also decomposed into three independent (parallel) in-

stances of the problem X. The results are interpreted conjunctively.

(9) If the result was F ($false$), then we terminate immediately and return the value W ($wrong$).

(10) If the result was true, then we first do some other actions G, before returning the value R ($right$).

(11) G stands for some actions that are necessary if an instance of the problem Z was successful. It could be updating a lookup table (for dynamic programming) or outputting a progress message (as it is done here in the program).

Let Δ be the set of rules defined here. It is clear that Δ is a PRS, but no PAN, PAD, PA or Petri net. This is because here the subroutines return values to their callers when they terminate. In the next section we'll show that the reachability problem is decidable for PRS. Then we can verify the system by checking the following properties. Let X be the initial state.

(i) It is possible to reach the state T.

(ii) It is not possible to reach the state $X\|Z$.

(iii) It is not possible to reach the state $W\|T$.

4 The Reachability Problem

In this section we show that PRS are not Turing-powerful.

Definition 4.1 *The reachability problem is the problem if a given state is reachable from an initial state.*

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

Question: *Is there a sequence σ of actions s.t. $t_0 \stackrel{\sigma}{\to} t$?*

For Basic Parallel Processes (BPP) reachability is NP-complete [3,6]. Although PA-processes are more expressive than BPP, the reachability problem is still NP-complete [12]. For Petri nets reachability is decidable and EXPSPACE-hard [9]. 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 rection and write just $t_1 \to t_2$ instead of $t_1 \stackrel{a}{\to} t_2$.

We prove 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 these PRS by reducing it to the reachability problem for Petri nets.

Definition 4.2 *For a PRS Δ define*

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

Δ is in normal form *iff all rules in Δ are of one of the following two forms:*

Par-Rule $X_1\|X_2\|\ldots\|X_i \to Y_1\|Y_2\|\ldots\|Y_k$, $i,k \in N$.

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

The only rules that are both seq-rules and par-rules are of the form $X \to Y$. The following relations $\succ^{\Delta}_{par}$ and $\succ^{\Delta}_{seq}$ are only technicalities used in the proofs.

$$t \succ^{\Delta}_{par} t' :\iff \exists\sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are par-rules from } \Delta$$

$$t \succ^{\Delta}_{seq} t' :\iff \exists\sigma.\ t \xrightarrow{\sigma} t' \text{ and all rules used in } \sigma \text{ are seq-rules from } \Delta$$

A PRS Δ is in transitive normal form *iff it is in normal form and for all $X, Y \in Var$*

$$X \succ^{\Delta} Y \ \Rightarrow\ (X \to Y) \in \Delta$$

Proposition 4.3 *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 $t_1 \succ^{\Delta}_{par} t_2$.*

Proof. Directly from the decidability of the reachability problem for Petri nets [9]. □

The transformation of a PRS into transitive normal form is done in two steps in the following lemmas.

Lemma 4.4 *Let Δ be a PRS using only variables from the finite set $Var(\Delta)$. Let $t_1, t_2 \in \mathcal{T}$ be two terms containing only variables from $Var(\Delta)$.*

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 variables from the finite set V' (with $Var(\Delta) \subseteq V' \subset Var$) and

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

Proof. Let k_i be the number of rules $t_1 \to t_2$ in Δ that are neither par-rules nor seq-rules and $size(t_1) + size(t_2) = i$. Let n be the maximal i s.t. $k_i \neq 0$. (n exists because Δ is finite). We define

$$Norm(\Delta) := (k_n, k_{n-1}, \ldots, k_1)$$

These norms are ordered lexicographically. Δ is in normal form iff $Norm(\Delta) = (0, \ldots, 0)$. 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. $Norm(\Delta') <_{lex} Norm(\Delta)$ and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$.

If Δ is not in normal form, then there exists a rule in Δ that is neither a seq-rule nor a par-rule. We call such rules *"bad rules"*. There are five types of bad rules:

(i) The bad rule is $u \to u_1.u_2$. Let Z, Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u \to Z \quad Z \to Z_1.Z_2 \quad Z_1 \to u_1 \quad Z_2 \to u_2$$

$t_1 = t_1'$ and $t_2 = t_2'$.

(ii) The bad rule is $u \to u_1 \| u_2$. Let Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u \to Z_1 \| Z_2 \quad Z_1 \to u_1 \quad Z_2 \to u_2.$$

$t_1 = t_1'$ and $t_2 = t_2'$.

(iii) The bad rule is $u_1 \| (u_2.u_3) \to u_4$. Let Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u_1 \to Z_1 \quad u_2.u_3 \to Z_2 \quad Z_1 \| Z_2 \to u_4$$

$t_1 = t_1'$ and $t_2 = t_2'$.

(iv) The bad rule is $u_1.u_2 \to u_3$, where u_2 is not a single variable. Let $Z \in V$ be a new variable. Δ', t_1', t_2' are constructed as follows: Substitute Z for u_2 in all rules in Δ and in t_1 and t_2. Then add the rule $Z \to u_2$.

(v) The bad rule is $(u_1 \| u_2).u_3 \to u_4$. Let Z be a new variable. We get Δ' by replacing the bad rule with the following two rules

$$u_1 \| u_2 \to Z \quad Z.u_3 \to u_4$$

$t_1 = t_1'$ and $t_2 = t_2'$.

In all these cases $Norm(\Delta') <_{lex} Norm(\Delta)$ and $t_1 \succ^{\Delta} t_2 \iff t_1' \succ^{\Delta'} t_2'$. Repeated application of this procedure yields the desired result. □

The following Lemma is used to prove the correctness of the algorithm in Lemma 4.6.

Lemma 4.5 *Let Δ be a PRS in normal form. If there are variables X, Y s.t. $X \succ^{\Delta} Y$ and $(X \to Y) \notin \Delta$ then there are also variables X', Y' with $(X' \to Y') \notin \Delta$ and $X' \succ^{\Delta}_{par} Y'$ or $X' \succ^{\Delta}_{seq} Y'$.*

Proof. Choose a pair of variables X, Y s.t. $(X \to Y) \notin \Delta$ and $X \xrightarrow{\sigma} Y$ for a sequence σ of minimal length. This means that the length of σ is minimal over the choice of X, Y and σ. If σ consists only of applications of par-rules or only of seq-rules, then the proof is complete. Otherwise there are two cases:

(i) 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 variable Z_4 s.t. $Z_2.Z_3 \xrightarrow{\sigma'} Z_4$. This contradicts the minimality of the length of σ.

(ii) The last nontrivial rule in σ is a seq-rule. If a par-rule $Z \to Z_1 \| \ldots \| Z_n$ occurs in σ then there is a subsequence σ' of σ and a variable Z' s.t. $Z \xrightarrow{\sigma'} Z'$. This contradicts the minimality of the length of σ.

Thus σ consists only of applications of par-rules or only of seq-rules. □

Lemma 4.6 *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 variables X, Y s.t. $X \succ^{\Delta} Y$ and to add rules $(X \to Y)$ to Δ. By Lemma 4.5 it suffices to check for $X \succ^{\Delta}_{par} Y$ and

$X \succ_{seq}^{\Delta} Y$. This is decidable because of Proposition 4.3 and the decidability of the reachability problem for pushdown processes. Lemma 4.5 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 variables X, Y with $(X \to Y) \notin \Delta'$ **do**
    If $X \succ_{par}^{\Delta'} Y$ or $X \succ_{seq}^{\Delta'} Y$ **then** $\Delta' := \Delta' \cup (X \to Y)$, flag := true **fi**;
  od;
od;

□

To simplify the presentation in the following lemma we define two subsets of the set of process terms $\mathcal{T}$.

Definition 4.7

(i) *Let $\mathcal{T}_{par} \subseteq \mathcal{T}$ be the set of process terms where the outermost operator is **not** sequential composition.*

(ii) *Let $\mathcal{T}_{seq} \subseteq \mathcal{T}$ be the set of process terms where the outermost operator is **not** parallel composition.*

Note that simple variables are allowed in both $\mathcal{T}_{par}$ and $\mathcal{T}_{seq}$. In fact $Var = \mathcal{T}_{par} \cap \mathcal{T}_{seq}$.

Lemma 4.8 *Let Δ be a PRS in transitive normal form and $V := Var(\Delta)$. It is decidable for two terms t, t' if $t \succ^{\Delta} t'$.*

Proof. by induction on $size(t) + size(t')$. The base case $(0, 0)$ is trivial. Otherwise there are several cases which are decidable by induction hypothesis and Proposition 4.3. Note that the following decompositions only hold because Δ is in transitive normal form.

(i) Let $\epsilon \neq t'_i \in \mathcal{T}_{seq}$ for $1 \leq i \leq n$ and $t_1, t_2, t_3 \neq \epsilon$, then

$$\begin{aligned}(t_1.t_2)\|t_3 \succ t'_1\| \ldots \|t'_n \iff\ & (t_1 \succ \epsilon \ \wedge\ t_2\|t_3 \succ t'_1\| \ldots \|t'_n) \vee \\ & (\exists i.\ t_1.t_2 \succ t'_i \wedge \\ & t_3 \succ t'_1\| \ldots \|t'_{i-1}\|t'_{i+1}\| \ldots \|t'_n) \vee \\ & (\exists X \in V.\ t_1.t_2 \succ X \ \wedge\ X\|t_3 \succ t'_1\| \ldots \|t'_n)\end{aligned}$$

These conditions are decidable by induction hypothesis.

(ii) Let $t \in P$ be a term that doesn't contain the operator for sequential composition. (So t can also be a single variable) and $n \geq 2$. Let $t'_i \in \mathcal{T}_{seq}$

for $1 \leq i \leq n$, then

$$t \succ t'_1 \| \ldots \| t'_n \iff \exists X_1, \ldots, X_n \in V.\ t \succ^{\Delta}_{par} X_1 \| \ldots \| X_n \wedge \forall i \in \{1, \ldots, n\}.\ X_i \succ t'_i$$

There are only finitely many choices for $X_1, \ldots X_n$. The first condition is decidable by Proposition 4.3, the others by induction hypothesis.

(iii) Let $t \in P$ be a term that doesn't contain the operator for sequential composition. (So t can also be a single variable).

$$t \succ t_1.t_2 \iff \exists X.\ t \succ^{\Delta}_{par} X \wedge (X \to Y_1.Y_2) \in \Delta \wedge Y_2 = t_2 \wedge Y_1 \succ t_1$$

This is true, because Δ is in transitive normal form and sequential composition is left-associative. The conditions are decidable by induction hypothesis.

(iv) Let t_1, t_2, t'_1, t'_2 be arbitrary terms $\neq \epsilon$

$$\begin{aligned} t_1.t_2 \succ t'_1 \| t'_2 \iff & (t_1 \succ \epsilon \wedge t_2 \succ t'_1 \| t'_2) \vee \\ & (\exists X, Y, Z \in V.\ t_2 = X \wedge t_1 \succ Y \wedge (Y.X \to Z) \in \Delta \\ & \wedge Z \succ t'_1 \| t'_2) \end{aligned}$$

The conditions are decidable by induction hypothesis.

(v) Let $X \in V$ and $t_1, t_2 \neq \epsilon$.

$$\begin{aligned} t_1.t_2 \succ X \iff & (t_1 \succ \epsilon \ \wedge \ t_2 \succ X) \vee \\ & (\exists Y, Z, W \in V. \\ & t_2 = Z \ \wedge \ t_1 \succ Y \ \wedge \ (Y.Z \to W) \in \Delta \ \wedge \ W \succ X) \end{aligned}$$

These conditions are decidable by induction hypothesis and because Δ is in transitive normal form.

(vi) Let $X \in V$ and $\epsilon \neq t'_i \in \mathcal{T}_{par}$ for $1 \leq i \leq n$.

$$\begin{aligned} X \succ t'_1 . \ldots . t'_n \iff & \exists Y, Z, W \in V.\ X \succ Y \ \wedge \ (Y \to Z.W) \in \Delta \wedge \\ & Z \succ t'_1 . \ldots . t'_{n-1} \ \wedge \ W = t'_n \end{aligned}$$

This is decidable by induction hypothesis.

(vii) Let $X \in V$.

$$X \succ \epsilon \iff X \succ^{\Delta}_{par} \epsilon$$

This is decidable by Proposition 4.3.

(viii) Let $n \geq 2$ and $\epsilon \neq t'_i \in \mathcal{T}_{par}$ for $1 \leq i \leq n$ and $t_1, t_2 \neq \epsilon$, then

$$\begin{aligned} t_1.t_2 \succ t'_1.\ldots.t'_n \iff & (t_1 \succ \epsilon \ \wedge \ t_2 \succ t'_1.\ldots.t'_n) \vee \\ & (\exists 0 \leq j < k.\ t_1 \succ t'_1.\ldots.t'_j \ \wedge \ t_2 = t'_{j+1}.\ldots.t'_n) \vee \\ & (\exists X, Y, Z \in V.\ t_2 = X \ \wedge \ t_1 \succ Y \ \wedge \\ & (Y.X \to Z) \in \Delta \ \wedge \ Z \succ t'_1.\ldots.t'_n) \end{aligned}$$

This is decidable by induction hypothesis.

(ix) Let t_1, t_2, t'_1, t'_2 be arbitrary terms $\neq \epsilon$.

$$\begin{aligned} t_1 \| t_2 \succ t'_1.t'_2 \iff & (t_1 \succ \epsilon \ \wedge \ t_2 \succ t'_1.t'_2) \vee \\ & (t_2 \succ \epsilon \ \wedge \ t_1 \succ t'_1.t'_2) \vee \\ & (\exists X \in V.\ t_1 \| t_2 \succ X \ \wedge \ X \succ t'_1.t'_2) \end{aligned}$$

This is decidable by induction hypothesis.

□

Theorem 4.9 *Let Δ be a PRS and $t_1, t_2 \in \mathcal{T}$. It is decidable if $t_1 \succ^{\Delta} t_2$.*

Proof. First apply Lemma 4.4 and transform Δ into a PRS Δ' in normal form and t_1, t_2 into the corresponding terms t'_1, t'_2. Then use Lemma 4.6 to transform Δ' into a PRS Δ'' in transitive normal form. It suffices to decide $t'_1 \succ^{\Delta''} t'_2$. By Lemma 4.8 this is possible. □

5 Extensions

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. First we define a simple logic to describe properties of states, and then we show that it is decidable if a PRS can reach a state that satisfies a given property.

Definition 5.1 *Let Act be the set of atomic actions. The syntax of the state-formulae is*

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

where $a \in Act$. The denotation of a formula is defined by

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

So $[\![\Phi]\!]$ denotes a (possibly infinite) set of PRS-terms. To simplify the notation we use sets of actions. Let $A := \{a_1, \ldots, a_k\} \subseteq Act$ and

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

By transformation to disjunctive normal form every state-formula Φ can be written as

$$(A_1^+ \wedge -A_1^-) \vee \ldots \vee (A_n^+ \wedge -A_n^-)$$

where $A_i^+, A_i^- \subseteq Act$. The modal operator $\Diamond$ is defined as usual.

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

Let $t \in \mathcal{T}$ be a PRS-term. For $t \in [\![\Phi]\!]$ we also write $t \models \Phi$.

We now define a generalized reachability problem. It is somewhat similar to the submarking reachability problem for Petri nets and often more useful for verification than pure reachability.

Definition 5.2 *The* reachable property problem *is the problem if there is a reachable state that satisfies certain properties.*

Instance: *A PRS Δ with initial state t_0 and a state formula Φ.*

Question: *Is it true that $t_0 \models \Diamond\Phi$?*

Let there be a PRS Δ **in transitive normal form** with initial state t_0 and Φ a state-formula. We now describe a tableau system that decides the problem $t_0 \models \Diamond\Phi$. 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 $\Diamond(A^+ \wedge -A^-)$. The nodes in the tableau will be sets of formulae (subgoals), which will be interpreted conjunctively. Γ denotes a set of formulae. The branches are interpreted disjunctively. The tableau is successful iff there is a successful leaf.

For technical reasons we introduce a new operator ∇ that is defined by

$$\|\nabla\Phi\| := \{t \mid \exists\sigma, t' \neq \epsilon.\ t \xrightarrow{\sigma} t' \in \|\Phi\|\}$$

Now we define the tableau rules. Every node in the tableau consists of a set of formulae. We describe the transformations of single elements of these sets. So the formula $\cup\Gamma$ should be appended to every node, where Γ denotes a set of formulae. We leave this out to simplify the notation.

$$SP1\ \frac{\{(t_1.(t_2\|t_3))\|t_4 \vdash \Diamond(A^+ \wedge -A^-)\}}{\{t_1 \succ \epsilon,\ t_2\|t_3\|t_4 \vdash \Diamond(A^+ \wedge -A^-)\} \quad \ldots\{t_1 \vdash \nabla(A_1^+ \wedge -A^-),\ t_4 \vdash \Diamond(A_2^+ \wedge -A^-)\}\ldots}$$

where $A^+ = A_1^+ \cup A_2^+$

$$SP2\ \frac{\{(t_1.Y)\|t_2 \vdash \Diamond(A^+ \wedge -A^-)\}}{\begin{array}{l}\{t_1 \succ \epsilon,\ Y\|t_2 \vdash \Diamond(A^+ \wedge -A^-)\} \\ \{t_1 \vdash \nabla(A_1^+ \wedge -A^-),\ t_2 \vdash \Diamond(A_2^+ \wedge -A^-)\} \\ \{t_1 \succ X,\ X.Y \vdash (A_1^+ \wedge -A^-),\ t_2 \vdash \Diamond(A_2^+ \wedge -A^-)\} \\ \{t_1 \succ X,\ (X.Y \to Z) \in \Delta,\ Z\|t_2 \vdash \Diamond(A^+ \wedge -A^-)\}\end{array}}\ (X \in Var(\Delta))$$

where $A^+ = A_1^+ \cup A_2^+$

$$SP3\ \frac{\{(t_1.(t_2\|t_3))\|t_4 \vdash \nabla(A^+ \wedge -A^-)\}}{\{t_1 \succ \epsilon,\ t_2\|t_3\|t_4 \vdash \nabla(A^+ \wedge -A^-)\} \quad \ldots\{t_1 \vdash \nabla(A_1^+ \wedge -A^-),\ t_4 \vdash \Diamond(A_2^+ \wedge -A^-)\}\ldots}$$

where $A^+ = A_1^+ \cup A_2^+$

$$SP4\ \frac{\{(t_1.Y)\|t_2 \vdash \nabla(A^+ \wedge -A^-)\}}{\begin{array}{l}\{t_1 \succ \epsilon,\ Y\|t_2 \vdash \nabla(A^+ \wedge -A^-)\} \\ \{t_1 \vdash \nabla(A_1^+ \wedge -A^-),\ t_2 \vdash \Diamond(A_2^+ \wedge -A^-)\} \\ \{t_1 \succ X,\ X.Y \vdash (A_1^+ \wedge -A^-),\ t_2 \vdash \Diamond(A_2^+ \wedge -A^-)\} \\ \{t_1 \succ X,\ (X.Y \to Z) \in \Delta,\ Z\|t_2 \vdash \nabla(A^+ \wedge -A^-)\}\end{array}}\ (X \in Var(\Delta))$$

where $A^+ = A_1^+ \cup A_2^+$

$$PAR1\ \frac{\{t \vdash \Diamond(A^+ \wedge -A^-)\}}{\ldots\{t_1 \vdash \nabla(A_1^+ \wedge -A^-),\ \ldots,\ t_k \vdash \nabla(A_k^+ \wedge -A^-)\}\ldots}$$

where $t \in P$ and $(X_i \to t_i) \in \Delta$, $(i = 1, \ldots, k)$ are seq-rules
$\exists t' \in P.\ t \succ_{par} (t'\|X_1\|\ldots\|X_k)$ with $t' \models (A_0^+ \wedge -A^-)$
and $A^+ = A_0^+ \cup A_1^+ \cup \ldots \cup A_k^+$

$$PAR2\ \frac{\{t \vdash \nabla(A^+ \wedge -A^-)\}}{\ldots\{t_1 \vdash \nabla(A_1^+ \wedge -A^-), \ldots, t_k \vdash \nabla(A_k^+ \wedge -A^-)\}\ldots}$$

where $t \in \mathcal{T}_{par}$ and $(X_i \to t_i) \in \Delta$, $(i = 1, \ldots, k)$ are seq-rules
$\exists t' \in P.\ t \succ_{par} (t'\|X_1\|\ldots\|X_k)$ with $t' \models (A_0^+ \wedge -A^-)$
and $A^+ = A_0^+ \cup A_1^+ \cup \ldots \cup A_k^+$, and $(k \neq 0 \vee t' \neq \epsilon)$

$$E1\ \frac{\{t \succ t'\} \cup \Gamma}{\Gamma}\ \text{if } t \succ t'$$

$$E2\ \frac{\{(X.Y \to Z) \in \Delta\} \cup \Gamma}{\Gamma}\ \text{if } (X.Y \to Z) \in \Delta$$

In the rules SP1,SP2,SP3,SP4 we have to consider all different (but only finitely many) ways of partitioning A^+ into A_1^+ and A_2^+. In PAR1 and PAR2 the dots symbolize all different ways of choosing k, the rules $(X_i \to t_i)$ and the partitioning of A^+ into $A_0^+, \ldots, A_k^+$. Again there are only finitely many.

Lemma 5.3 *If the side conditions of a rule are satisfied, then the antecedent of a rule is true iff one of its consequents is true.*

Proof. For the rules $SP1, SP2, SP3, SP4, E1$ and $E2$ this follows directly

from the definitions. PAR1 and PAR2 are true, because Δ is in parallel normal form. The only difference between PAR1 and PAR2 is the condition $k \neq 0 \vee t' \neq \epsilon$. This ensures that the reachable state that satisfies $(A^+ \wedge -A^-)$ is not ϵ. □

Definition 5.4 ***(Termination conditions)*** *A node marked with a set of formulae Γ is a terminal node iff one of the following conditions is satisfied.*

(i) Γ *is empty.*

(ii) $\Gamma = \Gamma' \cup \{t \succ t'\}$ *for some* $t, t' \in \mathcal{T}$ *and* ***not*** $t \succ t'$.

(iii) $\Gamma = \Gamma' \cup \{(X.Y \to Z) \in \Delta\}$ *and* ***not*** $(X.Y \to Z) \in \Delta$.

(iv) *The same node Γ occurred earlier on the same branch.*

Terminals of type 1 are successful, while terminals of types 2,3 and 4 are unsuccessful.

Lemma 5.5 *The tableau for a given root can be effectively constructed.*

Proof.

(i) The side conditions of the rules are decidable: For rule $E1$ this follows from Theorem 4.9. For $PAR1$ and $PAR2$ this follows from the decidability of the submarking reachability problem for Petri nets, because $Var(\Delta)$ is finite.

(ii) The tableau is finitely branching: This is because there are only finitely many different ways to partition A^+ into nontrivial subsets and because Δ is finite.

(iii) The tableau is finite: Let t_0 be the state in the root-node. There are only finitely many different subterms of t_0. As Δ is finite there are only finitely many different seq-rules. Only finitely many variables are used in Δ, thus $Var(\Delta)$ is finite. Only finitely many different formulae of the form $\Diamond(A^+ \wedge -A^-)$ or $\nabla(A^+ \wedge -A^-)$ can occur in the tableau. Therefore there are only finitely many different nodes in the tableau. Thus the construction of the tableau must terminate, because of termination condition 4.

□

Now we prove the soundness and completeness of the tableau system.

Lemma 5.6 *If there is a successful tableau with root $\{t \vdash \Diamond(A^+ \wedge -A^-)\}$, then $t \models \Diamond(A^+ \wedge -A^-)$.*

Proof. If the tableau is successful, then it has a branch that ends with a successful (empty) node. This node is certainly true. By Lemma 5.3 the root-node must be true as well. □

Lemma 5.7 *Let $Op \in \{\Diamond, \nabla\}$. Let there be a node of the form $\{t \vdash Op(A^+ \wedge -A^-)\} \cup \Gamma$, s.t. $t \models Op(A^+ \wedge -A^-)$ and Γ is true.*

Then the tableau with this root has a branch leading to a node Γ.

Proof. by complete induction on lexicographically ordered pairs (x, y) s.t. $(x, y) := (length(\sigma), size(t))$, where σ is a sequence of minimal length s.t. $t \xrightarrow{\sigma} t'$ and $t' \models (A^+ \wedge -A^-)$ (and $t' \neq \epsilon$ if $Op = \nabla$). Such a sequence must exist, because $t \models Op(A^+ \wedge -A^-)$.

If $(x, y) = (0, 0)$ then $t = \epsilon$ and $A^+ = \{\}$. The rule PAR1 is applicable and the one child-node is Γ.

Apply the rules for the construction of the tableau. By Lemma 5.3 at least one child-node must be true. Choose the true child-node that corresponds to σ. For the rules SP1, SP2, SP3 and SP4 in the child node x is lower or equal and y is smaller. With the induction hypothesis and rules E1,E2 the result follows. For the rules PAR1 and PAR2 the second component y may have increased in the child-node, but the first component x is always smaller. By multiple application of the induction hypothesis the result follows. This construction cannot be interrupted by termination condition 4, because this would contradict the minimality of the length of σ. □

Corollary 5.8 *If* $t \models \Diamond(A^+ \wedge -A^-)$*, then there is a successful tableau with root* $\{t \vdash \Diamond(A^+ \wedge -A^-)\}$.

So far we have only proved decidability of the reachable property problem for PRS in transitive normal form. For the general case more work is needed. It is not possible to apply the same algorithms as in Lemma 4.4 and Lemma 4.6 to transform a PRS into transitive normal form, because these transformation do not preserve the properties we want to check. A generalized version of Lemma 4.4 is necessary.

Lemma 5.9 *Let* Δ *be a PRS using only variables from the finite set* $Var(\Delta) \subset Var$ *and* t *a process term.*

Then a PRS Δ' *in normal form and a term* t' *can be effectively constructed s.t. for every state formula* Φ $t \models \Diamond\Phi$ *with respect to* Δ *iff* $t' \models \Diamond(\Phi \wedge \neg\lambda)$ *with respect to* Δ'*. (*λ *is a new action.)*

Proof. Let k_i be the number of rules $t_1 \to t_2$ in Δ that are neither par-rules nor seq-rules and $size(t_1) + size(t_2) = i$. Let n be the maximal i s.t. $k_i \neq 0$. (n exists because Δ is finite). We define

$$Norm(\Delta) := (k_n, k_{n-1}, \ldots, k_1)$$

These norms are ordered lexicographically. Δ is in normal form iff $Norm(\Delta) = (0, \ldots, 0)$. Now we describe a procedure that transforms Δ into a new PRS Δ' and t into t', with the above properties. For this we introduce two completely new atomic actions λ and τ that are not in Act and do not occur in any state formula Φ.

If Δ is not in normal form, then there exists a rule in Δ that is neither a seq-rule nor a par-rule. We call such rules *bad rules*. There are five types of

bad rules:

(i) The bad rule is $u \stackrel{a}{\to} u_1.u_2$. Let Z, Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u \stackrel{a}{\to} Z \quad Z \stackrel{\lambda}{\to} Z_1.Z_2 \quad Z_1 \stackrel{\lambda}{\to} u_1 \quad Z_2 \stackrel{\lambda}{\to} u_2$$

and $t' = t$.

(ii) The bad rule is $u \to u_1 \| u_2$. Let Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u \stackrel{a}{\to} Z_1 \| Z_2 \quad Z_1 \stackrel{\lambda}{\to} u_1 \quad Z_2 \stackrel{\lambda}{\to} u_2.$$

and $t' = t$.

(iii) The bad rule is $u_1 \| (u_2.u_3) \stackrel{a}{\to} u_4$. Let Z_1, Z_2, Z_3 be new variables. We get Δ' by replacing the bad rule by the following rules

$$u_1 \stackrel{\tau}{\to} Z_1 \quad u_2.u_3 \stackrel{\tau}{\to} Z_2 \quad Z_1 \| Z_2 \stackrel{\tau}{\to} Z_3 \quad Z_1 \stackrel{\lambda}{\to} Z_1 \quad Z_2 \stackrel{\lambda}{\to} Z_2 \quad Z_3 \stackrel{a}{\to} u_4$$

Then for all actions b that are enabled by the term $u_1 \| (u_2.u_3)$ with respect to Δ add to Δ' a rule $Z_3 \stackrel{b}{\to} Z_3$. Finally replace the term $u_1 \| (u_2.u_3)$ by Z_3 in all rules in Δ' and in t, thus obtaining t'.

(iv) The bad rule is $u_1.(u_2 \| u_3) \stackrel{a}{\to} u_4$. Let Z be a new variable. Δ' and t' are constructed as follows: Substitute Z for $(u_2 \| u_3)$ in all rules in Δ and in t. Then add the rule $Z \stackrel{\lambda}{\to} u_2 \| u_3$.

(v) The bad rule is $(u_1 \| u_2).u_3 \stackrel{a}{\to} u_4$. Let Z_1, Z_2 be new variables. We get Δ' by replacing the bad rule with the following rules

$$u_1 \| u_2 \stackrel{\tau}{\to} Z_1 \quad Z_1 \stackrel{\lambda}{\to} Z_1 \quad Z_1.u_3 \stackrel{\tau}{\to} Z_2 \quad Z_2 \stackrel{a}{\to} u_4$$

Then for all actions b that are enabled by the term $(u_1 \| u_2).u_3$ with respect to Δ add to Δ' a rule $Z_2 \stackrel{b}{\to} Z_2$. Finally replace the term $(u_1 \| u_2).u_3$ by Z_2 in all rules in Δ' and in t, thus obtaining t'.

In all these cases $Norm(\Delta') <_{lex} Norm(\Delta)$ and the property of the state formulae is preserved. Repeated application of this procedure yields the desired result. □

Now we can prove decidability for the general case.

Theorem 5.10 *The reachable property problem is decidable for PRS.*

Proof. Let there be a PRS Δ with initial state t_0 and Φ a state-formula. The problem is to decide if $t_0 \models \Diamond\Phi$.

First apply Lemma 5.9 to get a Δ' in normal form and a t_0' s.t. $t_0 \models \Diamond\Phi$ w.r.t. Δ iff $t_0' \models \Diamond(\Phi \wedge \neg\lambda)$ w.r.t. Δ'. Then we use the algorithm in Lemma 4.6 to transform the PRS Δ' into an equivalent PRS Δ'' in transitive normal form. All new rules that are added in this process are labeled with the special new action τ. It follows that $t_0 \models \Diamond\Phi$ w.r.t. Δ iff $t_0' \models \Diamond(\Phi \wedge \neg\lambda)$ w.r.t. Δ''. It suffices to show decidability for formulae of the form $\Diamond(A^+ \wedge -A^-)$. As Δ'' is in transitive normal form we can apply the tableau system. By Lemma 5.5

the tableau can be effectively constructed. It follows from Lemma 5.6 and Corollary 5.8 that the property holds if and only if the tableau is successful.□

This result can also be used to decide the reachability of a state of deadlock. Let Δ be a PRS with initial state t_0 and $Act(\Delta)$ the (finite!) set of actions used in Δ. A state of deadlock is reachable iff $t_0 \models \Diamond(-Act(\Delta))$.

Let us consider the example from Section 3 again. We can now do further verification: Let $Act(ex)$ be the set of all actions used in the example. It is possible to reach a state where $decomp_2$ is the only possible action

$$X \models \Diamond(decomp_2 \wedge -(Act(ex) - \{decomp_2\}))$$

but there is no reachable state where $decomp_1$ and $decomp_2$ are the only possible actions.

$$X \models \neg\Diamond(decomp_1 \wedge decomp_2 \wedge -(Act(ex) - \{decomp_1, decomp_2\}))$$

6 Conclusion

We have presented a unified view of many classes of infinite state concurrent systems by representing them as special cases of a general rewriting formalism.

The most general class of these rewrite systems is called *Process Rewrite Systems (PRS)*. It 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 are a very general model for concurrent systems, thus many temporal logics (EF, CTL, LTL, linear time μ-calculus, modal μ-calculus) are undecidable for it. This is because EF is undecidable for Petri nets [6,3], CTL is undecidable for BPP [7] and LTL and linear time μ-calculus are undecidable for PA-processes [1]. However, PRS is not Turing powerful, since reachability is still decidable.

References

[1] 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.

[2] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of*

CONCUR'96, volume 1119 of *LNCS*. Springer Verlag, 1996.

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

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

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

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

[7] 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.

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

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

[10] Richard Mayr. Combining Petri nets and PA-processes. In *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, LNCS. Springer Verlag, 1997. to appear.

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

[12] Richard Mayr. Tableau methods for PA-processes. In D. Galmiche, editor, *Analytic Tableaux and Related Methods (TABLEAUX'97)*, volume 1227 of *LNAI*. Springer Verlag, 1997.

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