Reachability Analysis of Process Rewrite Systems *

Ahmed Bouajjani and Tayssir Touili

LIAFA, Univ. of Paris 7, Case 7014, 2 place Jussieu, F-75251 Paris 5, France.
e-mail: {abou,touili}@liafa.jussieu.fr

Abstract. Process Rewrite Systems (PRS for short) subsume many common (infinite-state) models such as pushdown systems and Petri nets. They can be adopted as formal models of parallel programs (multithreaded programs) with procedure calls. We develop automata techniques allowing to build finite representations of the forward/backward sets of reachable configurations of PRSs modulo various term structural equivalences (corresponding to properties of the operators of sequential composition and parallel composition). We show that, in several cases, these reachability sets can be represented by polynomial size finite bottom-up tree-automata. When associativity and commutativity of the parallel composition is taken into account, nonregular representations based on (a decidable class of) counter tree automata are sometimes needed.

1 Introduction

Automatic verification of software systems is nowadays one of the most challenging research problems in computer-aided verification. A major difficulty to face when considering this problem comes from the fact that, reasoning about software systems requires in general to deal with infinite-state models. Indeed, regardless from the fact that they may manipulate data ranging over infinite domains, programs can have unbounded control structures due to, e.g., recursive procedure calls and multi-threading (dynamic creation of concurrent processes).

We consider in this paper as formal models of programs term rewrite systems called PRS (for process rewrite systems) [May98], in the spirit of the approach advocated in [EK99,EP00], and we develop automata-based techniques for performing reachability analysis of these systems. A PRS is a finite set of rules of the form $t \rightarrow t'$ where t and t' are terms built up from the idle process ("0"), a finite set of process variables (X), sequential composition ("$\cdot$"), and asynchronous parallel composition ("$||$").

The semantics of PRSs (see [May98]) considers terms modulo a structural equivalence $\sim$ which expresses the fact that 0 is a neutral element of "$\cdot$" and "$||$", that "$\cdot$" is associative, and that "$||$" is associative and commutative. With this semantics, PRSs subsume well-known models such as pushdown systems (equivalent to prefix rewrite systems) and Petri nets. Their relevance in the modeling of programs (given as flow graph systems) is shown in works such as [EK99,EP00,Esp02].

The problem we consider in this paper is computing representations of the $Post^*$ and Pre^* images of sets of process terms, for given PRS systems. Since process terms

* This work has been supported in part by the European IST-FET project ADVANCE (contract No. IST-1999-29082).

can be seen as trees, we consider representations of sets of terms based on (bottom-up) tree automata. We use in particular *finite* tree automata to represent *regular* sets of terms. However, due to the associativity of "$\cdot$" and the associativity-commutativity of "$\|$", $Post^*$ and Pre^* images of regular sets are not regular in general [GD89]. Therefore, we adopt an approach inspired from [LS98] which is based on (1) considering stronger structural equivalences obtained by ignoring some of the properties of the operators "$\cdot$" and "$\|$", and (2) computing representatives of the reachability sets modulo the considered structural equivalence, that is, a set which contains at least one representative of each equivalence class, instead of computing the whole reachability set. The idea is that in many cases (1) computing reachability sets for stronger equivalences is "easier" (e.g., regular and effectively constructible), and (2) the reachability analysis modulo $\sim$ can be shown to be reducible to computing representatives of the reachability set modulo some stronger equivalence.

Therefore, our aim in this work is to explore the problem of constructing reachability sets of PRSs modulo the following equivalences: (1) term equality ($=$), (2) the relation $\sim_0$ which takes into account the neutrality of 0 w.r.t. "$\cdot$" and "$\|$", (3) the relation $\sim_s$ which, in addition, considers the associativity of "$\cdot$", and (4) the equivalence $\sim$. Given one of these equivalences $\equiv$, we denote by $Post^*_{\equiv}$ and $Pre^*_{\equiv}$ the forward and backward reachability relations modulo $\equiv$.

In a first step, we consider the reachability problem modulo term equality. We show that for every PRS system, regular tree languages are effectively closed under $Post^*_{=}$ and $Pre^*_{=}$ images, and we give polynomial-time constructions of the corresponding tree automata. Then, we show that these constructions can be adapted to the case of the equivalence $\sim_0$. Our results generalize those of [LS98] concerning the class of PA systems, i.e., the subclass of PRS where all left-hand-sides of rules are process variables. (It is well known that PA is uncomparable with pushdown systems and Petri nets since it combines context-free processes, equivalent to pushdown systems with one control state, with the so-called basic parallel processes which are equivalent to synchronization-free Petri nets.)

Then, we consider the structural equivalences $\sim_s$ and $\sim$. As said above, these equivalences do not preserve regularity, and therefore, we address the problem of constructing $\equiv$-representatives of the $Post^*_{\equiv}$ and $Pre^*_{\equiv}$ images for these equivalences.

In the case of the equivalence $\sim_s$, we prove that regular $\sim_s$-representatives of the PRS $Post^*$ and Pre^* images of regular tree languages are effectively constructible, and we give polynomial-time constructions of corresponding finite tree automata. Of course, the constructed reachability sets are in general under-approximations of the reachability sets modulo $\sim$. However, for a significant class of parallel programs, the consideration of $\sim_{\|}$ is not necessary (their PRS model modulo $\sim_s$ coincides with the one modulo $\sim$), and therefore they can be analyzed using our constructions.

In the case of the equivalence $\sim$, we restrict ourselves to the class of PAD systems, i.e., the subclass of PRS where $\|$ does not appear in left-hand-sides of rules. This subclass subsumes both pushdown and PA systems (it combines prefix rewrite systems and synchronization-free Petri nets). The interest of this class is that it allows to model parallel programs where procedures can have return values. (Taking into account return values is possible with pushdown systems for instance, but not with PA processes.)

First, we show that a regular $\sim$-representative of the PAD $Post^*_\sim$-image of any regular tree language is polynomially constructible. As for computing the $Pre^*_\sim$-images, the problem is more delicate due to the parallel operators which appear in the right-hand-sides of the rules. In the case where the initial language is closed under $\sim_{\|}$, we show that the same result as for $Post^*_\sim$-images holds. When the initial language can be any regular set, we are not able to construct a regular $\sim$-representative of the $Pre^*_\sim$-images; nevertheless, we can construct a nonregular $\sim$-representative using counter tree automata. Fortunately, the counter automata obtained by our construction have the property that the emptiness problem of their intersection with regular sets is decidable. This makes our construction useful for reachability analysis-based verification.

For lack of space, some of the constructions and all proofs are only sketched. Details can be found in the full paper.

Related work: In [May98], Mayr considers the term reachability problem, i.e., given two PRS terms t and t', determine whether t' is reachable from t. He proved that this problem is decidable by a reduction to the reachability problem in Petri nets. The problem we consider here is different since we are interested in computing (a symbolic represenation of) the set of all reachable configurations, or a representative of it modulo a structural equivalence, starting from a (potentially infinite) set of configurations.

Symbolic reachability analysis based on (word) automata techniques has been used for model-checking pushdown systems in, e.g., [BEM97,FWW97,EHRS00]. In [LS98], this approach is extended to PA processes using tree automata as symbolic representation structures. The application of symbolic reachability analysis of PRSs to program (data flow) analysis is proposed and developed in the case of pushdown and PA systems in [EK99,EP00].

Our results in this paper generalize those given in [LS98,EP00] for PA. These works consider the construction of the reachability sets modulo term equality (without structural equivalences). In our work, we address this problem also modulo structural equivalences. Moreover, we show that the approach introduced in these papers can be extended to the more general classes of PAD and PRS. This allows to reason about larger classes of programs (where procedures can have return values, and where parallel processes can have some kind of communication).

In [BET03] we consider the analysis of programs modeled by means of communicating pushdown systems. This problem is in general undecidable, and then we propose an algorithmic approach based on computing (either finite or commutative) abstractions of pushdown languages. The framework we consider here is different since (1) we do not consider in [BET03] dynamic creation of processes, and (2) we provide in [BET03] approximate analysis techniques for a precise model whereas we provide here precise analysis techniques for a "weaker" (more abstract) model (PRSs are weaker in the sense that they allow threads to synchronize only when their stacks are empty, which is the reason behind the decidability of their reachability problem).

Finally, it is well known that ground term rewriting (GTR) systems preserve regularity [DT90]. However, even if PRSs are syntactically sets of ground rewriting rules, they are not standard GTR systems due to the semantics of the operator "$\cdot$" which imposes a particular rewriting strategy. Hence, even in the case where the considered structural equivalence is term equality, our results are not covered by [DT90].

2 Preliminaries

2.1 Terms and tree automata

An alphabet Σ is ranked if it is endowed with a mapping $rank : \Sigma \to \mathbb{N}$. For $k \geq 0$, Σ_k is the set of elements of rank k. Let X be a fixed denumerable set of variables $\{x_1, x_2, \ldots\}$. The set $T_\Sigma[X]$ of terms over Σ and X is the smallest set that satisfies: $\Sigma_0 \cup X \subseteq T_\Sigma[X]$, and if $k \geq 1$, $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma[X]$, then $f(t_1, \ldots, t_k)$ is in $T_\Sigma[X]$. T_Σ stands for $T_\Sigma[\emptyset]$. Terms in T_Σ are called *ground terms*. A term in $T_\Sigma[X]$ is *linear* if each variable occurs at most once. A *context* C is a linear term of $T_\Sigma[X]$. Let $t_1, \ldots, t_n$ be terms of T_Σ, then $C[t_1, \ldots, t_n]$ denotes the term obtained by replacing in the context C the occurrence of the variable x_i by the term t_i, for each $1 \leq i \leq n$.

Definition 1 ([CDG+97]). *A* **tree automaton** *is a tuple $A = (Q, \Sigma, F, \Delta)$ where Q is a set of states, Σ is a ranked alphabet, $F \subseteq Q$ is a set of final states, and Δ is a set of rules of the form (1) $f(q_1, \ldots, q_n) \to q$, or (2) $a \to q$, or (3) $q \to q'$, where $a \in \Sigma_0$, $n \geq 0$, $f \in \Sigma_n$, and $q_1, \ldots, q_n, q, q' \in Q$. If Q is finite, A is called a* finite tree automaton.

Let $\to_\Delta$ be the move relation of A defined as follows: Given t and t' two terms of $T_{\Sigma \cup Q}$, then $t \to_\Delta t'$ iff there exist a context $C \in T_{\Sigma \cup Q}[X]$, and (1) n ground terms $t_1, \ldots, t_n \in T_\Sigma$, and a rule $f(q_1, \ldots, q_n) \to q$ in Δ, such that $t = C[f(q_1, \ldots, q_n)]$, and $t' = C[q]$, or (2) a rule $a \to q$ in Δ, such that $t = C[a]$, and $t' = C[q]$, or (3) a rule $q \to q'$ in Δ, such that $t = C[q]$, and $t' = C[q']$. Let $\xrightarrow{*}_\Delta$ be the reflexive-transitive closure of $\to_\Delta$. A term t is accepted by a state $q \in Q$ iff $t \xrightarrow{*}_\Delta q$. Let L_q be the set of terms accepted by q. The language accepted by the automaton A is $L(A) = \bigcup \{L_q \mid q \in F\}$.

A tree language is *regular* if it is accepted by a *finite* tree automaton. The class of regular tree languages is effectively closed under union, intersection, and complementation. The emptiness problem of finite tree automata can be solved in linear time.

2.2 Counter tree automata

We define hereafter counter tree automata and introduce a subclass of these automata, called 0-test counter tree automata we use later in reachability analysis of process rewrite systems.

Definition 2. *A* **counter tree automaton** *(CTA for short) is a tuple $A = (Q, \Sigma, F, \mathbf{c}, \Delta)$ where Q is a finite set of control states, Σ is a ranked alphabet, $F \subseteq Q$ is a set of final states, $\mathbf{c} = (c_1, \ldots, c_m)$ is a vector of integer counters, and Δ is a set of rules of the form (A) $f(q_1, \ldots, q_n) \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q$, or (B) $a \xrightarrow{true/\mathbf{k}} q$, or (C) $q \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q'$, where $a \in \Sigma_0$, $n \geq 0$, $f \in \Sigma_n$, $q_1, \ldots, q_n, q, q' \in Q$, $\mu(\mathbf{c})$ is a formula in Presburger arithmetics depending on $\mathbf{c}$, and $\mathbf{k} \in \mathbb{Z}^m$. (The rule $f(q_1, \ldots, q_n) \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q$, where $\mu = true$ and $\mathbf{k} = 0^m$, will be denoted simply by $f(q_1, \ldots, q_n) \to q$.)*

Intuitively, a CTA is a tree automaton supplied with a set of counters: The rules (A), (B), and (C) behave respectively like the rules (1), (2), and (3) of Definition 1. A rule $f(q_1, \ldots, q_k) \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q$ can be applied if the values of the counters satisfy $\mu(\mathbf{c})$, and in this case, the vector of counters $\mathbf{c}$ is incremented by $\mathbf{k}$.

Formally, we define the language of the CTA $A = (Q, \Box, F, \mathbf{c}, \Box)$ as the set of ground terms accepted by an associated infinite tree automaton A_c we introduce hereafter. Let us first consider the following notation: Given a Presburger formula $\mu(\mathbf{c})$ and $\mathbf{v}$ a valuation of $\mathbf{c}$, we write $\mathbf{v} \models \mu$ iff $\mu(\mathbf{v})$ is true. Then, the automaton A_c is given by $(Q_c, \Box, F_c, \Box_c)$ where $Q_c = Q \times \mathbb{Z}^m$, $F_c = F \times \mathbb{Z}^m$, and $\Box_c$ is the smallest set of rules s.t.:

- $f\big((q_1, \mathbf{v_1}), \ldots, (q_n, \mathbf{v_n})\big) \to (q, \mathbf{v}) \in \Box_c$ if $f(q_1, \ldots, q_n) \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q$ is in $\Box$, $\Box_{i=1}^{n} \mathbf{v_i} \models \mu$, and $\mathbf{v} = \mathbf{k} + \Box_{i=1}^{n} \mathbf{v_i}$,
- $a \to (q, \mathbf{v}) \in \Box_c$ if $a \xrightarrow{true/\mathbf{k}} q$ is in $\Box$, and $\mathbf{v} = \mathbf{k}$,
- $(q, \mathbf{v}) \to (q', \mathbf{v}') \in \Box_c$ if $q \xrightarrow{\mu(\mathbf{c})/\mathbf{k}} q'$ is in $\Box$, $\mathbf{v} \models \mu$, and $\mathbf{v}' = \mathbf{v} + \mathbf{k}$.

Obviously, the emptiness problem is undecidable for CTAs. We introduce hereafter a class of CTAs for which we can prove useful closure and decision properties.

Definition 3. *A* **0-test counter tree automaton** *(0-CTA for short) is a CTA whose rules are such that $\mu(\mathbf{c})$ is either true, or is equal to the test $\bigwedge_{i=1}^{n} c_i = 0$.*

Theorem 1. *The intersection of a regular tree language and a 0-CTA language is a 0-CTA language. Moreover, the emptiness problem of 0-CTAs is decidable in nondeterministic polynomial time.*

3 Process Rewrite Systems

3.1 Definition

Let $Var = \{X, Y, \ldots\}$ be a set of process variables, and T_p be the set of process terms t defined by the following syntax, where X is an arbitrary constant from *Var*:

$$t ::= 0 \mid X \mid t \cdot t \mid t \| t$$

Intuitively, 0 is the null (idle) process and "$\cdot$" (resp. "$\|$") denotes sequential composition (resp. parallel composition). We use both prefix and infix notations to represent terms.

Definition 4 ([May98]). *A* Process Rewrite System *(PRS for short) is a finite set of rules of the form $t_1 \to t_2$, where $t_1, t_2 \in T_p$. A* PAD *is a PRS where all left-hand-sides of their rules do not contain "$\|$". A* PA *is a PAD where all the rules have the form $X \to t$.*

A PRS R induces a transition relation $\to_R$ over T_p defined by the following rules:

$$\frac{t_1 \to t_2 \in R}{t_1 \to_R t_2}; \frac{t_1 \to_R t_1'}{t_1 \| t_2 \to_R t_1' \| t_2}; \frac{t_1 \to_R t_1'}{t_1 \cdot t_2 \to_R t_1' \cdot t_2}; \frac{t_2 \to_R t_2'}{t_1 \| t_2 \to_R t_1 \| t_2'}; \frac{t_1 \sim_0 0 \ , \ t_2 \to_R t_2'}{t_1 \cdot t_2 \to_R t_1 \cdot t_2'}$$

where $\sim_0$ is an equivalence between process terms that identifies the terminated processes. It expresses the neutrality of the null process "0" w.r.t. "$\|$", and "$\cdot$":

A1: $\quad t \cdot 0 \ \sim_0 \ 0 \cdot t \ \sim_0 \ t \| 0 \ \sim_0 \ 0 \| t \ \sim_0 \ t$

We consider the structural equivalence $\sim$ generated by A1 and:

A2: $(t \cdot t') \cdot t'' \sim t \cdot (t' \cdot t'')$: associativity of "$\cdot$",
A3: $t \| t' \sim t' \| t$: commutativity of "$\|$",
A4: $(t \| t') \| t'' \sim t \| (t' \| t'')$: associativity of '$\|$".

We denote by $\sim_s$ the equivalence induced by the axioms A1 and A2, and by $\sim_{\|}$ the equivalence induced by the axioms A1, A3, and A4. Each equivalence $\equiv$ induces a transition relation $\Rightarrow_{\equiv,R}$ defined as follows:

$$\forall t,t' \in T_p, t \Rightarrow_{\equiv,R} t' \text{ iff } \exists u,u' \in T_p \text{ such that } t \equiv u, u \rightarrow_R u', \text{ and } u' \equiv t'$$

Let $\stackrel{*}{\Rightarrow}_{\equiv,R}$ be the reflexive transitive closure of $\Rightarrow_{\equiv,R}$. Let $Post^*_{R,\equiv}(t) = \{t' \in T_p \mid t \stackrel{*}{\Rightarrow}_{\equiv,R} t'\}$, and $Pre^*_{R,\equiv}(t) = \{t' \in T_p \mid t' \stackrel{*}{\Rightarrow}_{\equiv,R} t\}$. These definitions are extended to sets of terms in the standard way. We omit the subscript $\equiv$ when it corresponds to the identity (=).

3.2 Reachability analysis problem

A PRS process term can be seen as a tree over the alphabet $\Box = \Box_0 \cup \Box_2$, where $\Box_0 = \{0\} \cup Var$ and $\Box_2 = \{\cdot, \|\}$. A set of terms is *regular* if it can be represented by a finite tree automaton.

The $\equiv$-reachability problem consists in, given two regular sets of terms L_1 and L_2, deciding whether $Post^*_{R,\equiv}(L_1) \cap L_2 \neq \emptyset$, or equivalently whether $Pre^*_{R,\equiv}(L_2) \cap L_1 \neq \emptyset$. Therefore, the basic problems we consider in this paper are to compute, given a regular set L of terms, representations of the sets $Post^*_{R,\equiv}(L)$ and $Pre^*_{R,\equiv}(L)$. However, it can immediately be seen that, due to the associativity of "$\cdot$" and to the associativity-commutativity of "$\|$", these reachability sets are not regular in the cases of the equivalences $\sim_s$, $\sim_{\|}$, and $\sim$. The approach we follow in this case is to solve the $\equiv$-reachability problem using only *representatives* of these reachability sets. Let us introduce this notion through some definitions: Let t be a process term. We denote by $[t]_\equiv$ the equivalence class modulo $\equiv$ of t, i.e., $[t]_\equiv = \{t' \in T_p \mid t \equiv t'\}$. This definition is extended straightforwardly to sets of terms. We say that a set of terms L is $\equiv$-*compatible* if $[L]_\equiv = L$. Then, a set of terms L' is a $\equiv$-*representative* of L if $[L']_\equiv = L$.

Lemma 1. *Let L_1, L_2 be two sets of terms, and let L'_1 be a $\equiv$-representative of L_1. If L_2 is $\equiv$-compatible, then $L'_1 \cap L_2 \neq \emptyset$ iff $L_1 \cap L_2 \neq \emptyset$.*

So, computing regular $\equiv$-representatives of the $Post^*_{R,\equiv}$ and $Pre^*_{R,\equiv}$ images of regular sets allows to solve reachability problems. Actually, these $\equiv$-representative sets do not need to be regular, but only to be in some class of languages for which the emptiness of the intersection with regular sets is decidable. This is the case for instance for the class of 0-test counter tree automata.

In the remainder of the paper, we assume w.l.o.g. that PRS are in *normal form*, that is, their rules are of the forms $t_1 \rightarrow t_2$ where t_1 and t_2 are of the following forms: 0, X, $X \| Y$, or $X \cdot Y$. Indeed, from the proof of a very close fact in [May98], it can be shown that for every PRS R over a set of process variables Var, it is possible to associate a PRS R' in normal form over a new set of process variables Var' (which extends Var by some auxiliary process variables), and there exists two (composable) regular relations S_1 and S_2 such that, for every $\equiv$, $Post^*_{R,\equiv} = S_2 \circ Post^*_{R',\equiv} \circ S_1$ and $Pre^*_{R,\equiv} = S_2 \circ Pre^*_{R',\equiv} \circ S_1$.

4 Reachability modulo term equality and modulo $\sim_0$

4.1 Reachability modulo term equality

We prove in this section that for any regular language L, $Post^*_R(L)$ and $Pre^*_R(L)$ are effectively regular. Let $Sub_r(R)$ be the set of all the subterms of the right hand sides of the rules of R. Let $Q_R = \{q_t \mid t \in Sub_r(R)\}$, and let Δ_R be the following transition rules:

- $X \to q_X$, for every $X \in Sub_r(R)$,
- $\|(q_X, q_Y) \to q_{X\|Y}$, if $X\|Y \in Sub_r(R)$,
- $\cdot(q_X, q_Y) \to q_{X\cdot Y}$, if $X \cdot Y \in Sub_r(R)$.

It is clear that with Δ_R, for every $t \in Sub_r(R)$, the state q_t accepts $\{t\}$ (i.e., $L_{q_t} = \{t\}$). We define the automaton $A^*_R = (Q', \Delta, F', \Delta')$ as follows:

- $Q' = \{q, q^{nil}, q^T \mid q \in Q \cup Q_R\}$. We denote by $\tilde{q}$ any state in $\{q, q^T, q^{nil}\}$.
- $F' = \{q, q^{nil}, q^T \mid q \in F\}$,
- Δ' is the smallest set of rules containing $\Delta \cup \Delta_R$ s.t. for every $q_1, q_2, q \in Q \cup Q_R$:

1. $q \to q^T \in \Delta'$ for every $q \in Q \cup Q_R$,
2. if $0 \xrightarrow{*}_\Delta q$, then $0 \to q^{nil} \in \Delta'$,
3. if $t_1 \to t_2 \in R$, and there is a state $q \in Q \cup Q_R$ such that $t_1 \xrightarrow{*}_\Delta q^T$, then:
   (a) $q^T_{t_2} \to q^T \in \Delta'$,
   (b) $q^{nil}_{t_2} \to q^{nil} \in \Delta'$, and
   (c) $q^T_{t_2} \to q^{nil} \in \Delta'$ if $t_1 = 0$,
4. if $\cdot(q_1, q_2) \to q \in \Delta \cup \Delta_R$, then:
   (a) $\cdot(q^{nil}_1, \tilde{q}_2) \to q^T \in \Delta'$,
   (b) $\cdot(q^{nil}_1, q^{nil}_2) \to q^{nil} \in \Delta'$,
   (c) $\cdot(q^T_1, q_2) \to q^T \in \Delta'$,
5. if $\|(q_1, q_2) \to q \in \Delta \cup \Delta_R$, then:
   (a) $\|(q^{nil}_1, q^{nil}_2) \to q^{nil} \in \Delta'$,
   (b) $\|(\tilde{q}_1, \tilde{q}_2) \to q^T \in \Delta'$,
6. if $q \to q' \in \Delta$, then $q^T \to q'^T \in \Delta'$, and $q^{nil} \to q'^{nil} \in \Delta'$.

The definition of Δ' is inductive and can be computed as the limit of a finite sequence of increasing sets of transitions $\Delta'_1 \subset \Delta'_2 \subset \ldots \subset \Delta'_n$, where Δ'_{i+1} contains at most three transitions more that Δ'_i. These transitions are added by the inference rules (3). This procedure terminates because there is a finite number of states in $Q \cup Q_R$.

Theorem 2. *Let L be a regular set of process terms, and $A = (Q, \Delta, F, \Delta)$ be a finite tree automaton that recognizes L. Then, $Post^*_R(L)$ and $Pre^*_R(L)$ are recognized by the finite tree automata A^*_R and $A^*_{R^{-1}}$, respectively.*

Let us sketch the proof idea. To show that $Post^*_R(L)$ is accepted by A^*_R, it suffices to show that for every $q \in Q \cup Q_R$, the state q^T accepts the successors of L_q (i.e., $L_{q^T} = Post^*_R(L_q)$) and the state q^{nil} accepts the successors u of L_q that have been obtained from null successors of L_q (i.e., $L_{q^{nil}} = Post^*_R\big(Post^*_R(L_q) \cap \{u \in T_p \mid u \sim_0 0\}\big)$). In particular this means that for every $t \in Sub_r(R)$, $L_{q^T_t} = Post^*_R(t)$, and $L_{q^{nil}_t} = Post^*_R\big(Post^*_R(t) \cap \{u \in T_p \mid u \sim_0 0\}\big)$. Rules (1) express that $L_q \subseteq Post^*_R(L_q)$. Rules (2) mark the null leaves with the superscript ^{nil}, indicating that they are null. Rules (3) express that if a term t in $Sub_r(R)$ is a successor of a term in L_q, then so are all the successors of t. The "$\|$" nodes are annotated by the rules (5). For example, the intuition formalized by the rules (5b) is that if $u_1 \in Post^*_R(L_{q_1})$, and $u_2 \in Post^*_R(L_{q_2})$, then $u_1\|u_2 \in Post^*_R(L_q)$ if

$\|(q_1,q_2) \to q \in \Delta \cup \Delta_R$. The rules (4) annotate the "." nodes according to the semantics of ".". The states q and q^{nil} play an important role for these rules. Indeed, the rules (4a) and (4c) ensure that the right child of a "." node cannot be rewritten if the left child was not null, i.e., if the left child is not labeled by a state q^{nil}. The rules (6) express that if $L_q \subseteq L_{q'}$, then $Post^*_R(L_q) \subseteq Post^*_R(L_{q'})$. Finally, the fact that $Post^*_R(L)$ is accepted by A^*_R implies that $A^*_{R^{-1}}$ accepts $Pre^*_R(L)$ since $Pre^*_R(L) = Post^*_{R^{-1}}(L)$.

Remark 1. If A has k states and Δ transitions, then A^*_R has $O(k|Sub_r(R)| + |Sub_r(R)|^2 + \Delta)$ transitions and $3(k + |Sub_r(R)|)$ states.

4.2 Reachability modulo $\sim_0$:

The construction above can be adapted to perform reachability analysis modulo $\sim_0$.

Theorem 3. *Let R be a PRS and L be a regular tree language, then a finite tree automaton A^{*nil}_R can be effectively computed in polynomial time such that $Post^*_{R,\sim_0}(L)$ is recognized by A^{*nil}_R, and $Pre^*_{R,\sim_0}(L)$ is recognized by $A^{*nil}_{R^{-1}}$.*

5 Reachability modulo $\sim_s$

Computing $post^*$- and pre^*-images modulo $\sim_s$ does not preserve regularity [GD89]. Therefore, we propose in this section to compute $\sim_s$-representatives of the reachability sets $Post^*_{R,\sim_s}(L)$ and $Pre^*_{R,\sim_s}(L)$. We show hereafter that for any regular language L, we can effectively compute finite tree automata that recognize $\sim_s$-representatives of $Post^*_{R,\sim_s}(L)$ and $Pre^*_{R,\sim_s}(L)$.

The main difficulty comes from the fact that the rewritings are not done locally as in the previous case. Indeed, so far, a rule $X \cdot Y \to t$ is applied to a term u only if u has $X \cdot Y$ as an explicit subterm. This is no longer the case when we consider terms modulo $\sim_s$. Indeed, this rule should be applied for instance to the terms $X \cdot \big(Y \cdot (Z \cdot T)\big)$ and $X \cdot \big((Y \cdot Z) \cdot T\big)$ since they are $\sim_s$-equivalent to $(X \cdot Y) \cdot (Z \cdot T)$. To be more precise, let us introduce the notion of *seq-context*:

Definition 5. *Let $x \in X$, a* seq-context *is a single-variable context $C[x]$ such that: (1) x is the leftmost leaf of C, and (2) all the ancestors of the variable x are labeled by "."*.

Then, modulo $\sim_s$, a rule $X \cdot Y \to t$ can be applied to any term of the form $\cdot(X, C[Y])$ for a seq-context C, to yield a term that is $\sim$-equivalent to $C[t]$. We define now the relation Δ_R that performs this transformation for an arbitrary PRS as the smallest transition relation over T_p that contains $\Rightarrow_{\sim_0,R}$ (since $\sim_s$ includes $\sim_0$) and such that for every rule $X \cdot Y \to t$ in R, and every seq-context C, $\big(\cdot(X, C[Y]), C[t]\big) \in \Delta_R$. It can be shown that $\Delta^*_R(L)$ is a $\sim_s$-representative of $Post^*_{R,\sim_s}(L)$.

Proposition 1. *For every PRS R and regular language L, $Post^*_{R,\sim_s}(L) = [\Delta^*_R(L)]_{\sim_s}$ and $Pre^*_{R,\sim_s}(L) = [\Delta^*_{R^{-1}}(L)]_{\sim_s}$.*

Then, our next objective is to compute $\Delta^*_R(L)$ for a regular language L.

Theorem 4. *For every PRS R and every regular language L, the set $\square^*_R(L)$ is effectively regular. Furthermore, from an automaton with k states and $\square$ transitions that recognizes L, it is possible to construct in polynomial time an automaton accepting $\square^*_R(L)$, with $O\big((k+|Sub_r(R)|)^2|Var|^2+\square\big)$ transitions $O(k|Var|+|Sub_r(R)||Var|)$ states.*

We give hereafter the idea behind the construction. When annotating a subterm of the form $C[t]$ where C is a seq-context and t is the right-hand-side of a rule $X \cdot Y \to t$ in R, the automaton guesses when it reaches the root of the subterm t that at this place there was a Y which was rewritten, and memorizes the X in his state (moving to a state of the form $\langle q, X\rangle$) in order to validate his guess afterwards when reaching the root of the seq-context C. At that point, the automaton checks that the term $C[t]$ he annotated so far is indeed a successor of a term of the form $\cdot(X, C[Y])$ accepted at the current state. In such a case, the automaton considers that $C[t]$ should also be accepted at the same state. (For that, we add, for every transition rule $\cdot(q_1,q_2) \to q$ in the automaton, a new transition rule $\langle q_2, X\rangle \to q$, if $X \in L_{q_1^T}$.) The crucial point is that it can be shown that at each node the automaton has to memorize at most one guess. This is due to the semantics of the "$\cdot$" operator which ensures that at one node, at most two nonlocal rewritings can occur.

6 Reachability modulo $\sim$ for PAD systems

We consider in this section the problem of reachability analysis (modulo $\sim$) of the class of PAD processes (see Definition 4). We show that forward analysis can be done in polynomial time by computing regular representatives of $Post^*_\sim$ images. Backward analysis can also be done similarly, but only if we start from a $\sim_{\|}$-compatible set. In the general case, we show that backward reachability analysis can still be done, but this time by computing nonregular representatives of $Pre^*_\sim$ images using 0-test counter tree automata.

6.1 Preliminaries

Definition 6. *A* paral-context *is a context $C[x_1,\ldots,x_n]$ such that all the ancestors of the variables $x_1,\ldots,x_n$ are labeled by "$\|$".*

The main difficulty comes from the rules $X\|Y \to t$ that can be applied nonlocally. More precisely, modulo $\sim$, this rule can be applied to any term of the form $C[X,Y]$ for a paral-context $C[x_1,x_2]$, to yield a term that is $\sim$-equivalent to $C[0,t]$. For technical reasons, we introduce a special new process constant "$\tilde{0}$" that is considered as $\sim$-equivalent to 0, and consider the term $C[\tilde{0},t]$ (which is $\sim$-equivalent to $C[0,t]$). The difference between "$\tilde{0}$" and "0" is that "$\tilde{0}$" is never simplified (modulo $\sim_0$) nor rewritten. Technically, it has the role of a marker which allows to determine the positions where $\|$-rules have been applied. Then, we have to deal with the presence of "$\tilde{0}$" in the terms.

Definition 7. *A* null-context *is a single-variable context $C[x]$ such that all the leaves other than the variable x are labeled by "$\tilde{0}$".*

In other words, if C_0 is a null-context, then $C_0[X]$ is $\sim$-equivalent to X. Therefore, if C is a paral-context, and C_0, C_0' are null-contexts, then the rule $X \| Y \to t$ has to be applied to any term of the form $C\big[C_0[X], C_0'[Y]\big]$, to yield a term $\sim$-equivalent to $C\big[C_0[\tilde{0}], C_0'[t]\big]$. In the same manner, if C_s is a seq-context, then a rule of the form $X \cdot Y \to t$ has to be applied to any term of the form $\cdot(C_0[X], C_s[Y])$, and rewrite it into a term $\sim$-equivalent to $\cdot(C_0[\tilde{0}], C_s[t])$. Observe that on the right child $C_s[Y]$, it is not possible to have $\tilde{0}$'s. This is due to the prefix rewriting policy of "$\cdot$" and to the fact that $\tilde{0}$'s are absent from the initial language. We introduce a relation $\square_R$ which corresponds to these transformations. This relation is defined, for a given PRS R, as the smallest relation over T_p which contains $\square_R$ (since $\sim$ includes $\sim_s$) and satisfies:

(B1) For every rule $X \| Y \to t$ in R, if there exist a paral-context C and two null-contexts C_0 and C_0' such that $t = C\big[C_0[X], C_0'[Y]\big]$ and $t' = C\big[C_0[\tilde{0}], C_0'[t]\big]$, then $(t, t') \in \square_R$.

(B2) For every rule $X \cdot Y \to t$ in R, if there exist a seq-context C, and a null-context C_0 such that $t_1 = \cdot(C_0[X], C[Y])$, and $t_2 = \cdot(C_0[\tilde{0}], C[t])$, then $(t_1, t_2) \in \square_R$.

Proposition 2. *For every PRS R and every language L, $Post^*_{R,\sim}(L) = [\square^*_R(L)]_\sim$ and $Pre^*_{R,\sim}(L) = [\square^*_{R^{-1}}(L)]_\sim$.*

6.2 Regular analysis of PAD systems

Let R be a PAD system. Then, given a language L, it is easy to see that we have $\square^*_R(L) = \square^*_R(L)$. This is due to the fact that PAD systems do not contain rules of the form $X \| Y \to t$. Therefore, by Proposition 2, $Post^*_{R,\sim}$ images can be computed using the construction of Section 5. Moreover, if we assume that L is $\sim_\|$-compatible, then $\square^*_{R^{-1}}(L) = \square^*_{R^{-1}}(L)$. This is due to the fact that for PAD systems, the rules of R^{-1} do not create terms with the $\|$ operator, and thus, their application preserves $\sim_\|$-compatibility. Therefore, by Proposition 2, representatives of $Post^*_{R,\sim}$ images of regular sets, as well as representatives of $Pre^*_{R,\sim}$ images of $\sim_\|$-compatible sets, can be computed using the polynomial construction of Section 5.

Theorem 5. *Let R be a PAD system. Then, for every regular (resp. regular $\sim_\|$-compatible) tree language L, a regular $\sim$-representative of the set $Post^*_{R,\sim}(L)$ (resp. $Pre^*_{R,\sim}(L)$) can be effectively constructed in polynomial time.*

6.3 Nonregular backward reachability analysis of PAD systems

We consider now the remaining problem of computing $\sim$-representatives of $Pre^*_{R,\sim}$ images starting from any regular set. We show that we can compute in this case for any given regular set L, a 0-CTA representing $\square^*_{R^{-1}}(L)$, (which is, by Proposition 2, a $\sim$-representative of $Pre^*_{R,\sim}(L)$). This allows to perform the backward reachability analysis of PADs (using Theorem 1 and Lemma 1).

Theorem 6. *Let L be a regular language, and let R be a PAD, then $\square^*_{R^{-1}}(L)$ can be effectively characterized by a 0-CTA. Furthermore, from an automaton with k states and $\square$ transitions that recognizes L, it is possible to construct an automaton accepting $\square^*_{R^{-1}}(L)$, with $O\big(2^{|Var|^2} \cdot |Var|^2 \cdot (k + |Sub_r(R^{-1})|)\big)$ states, and $O\big(2^{|Var|^2} \cdot |Var|^4 \cdot (k + |Sub_r(R^{-1})|) \cdot |Sub_r(R^{-1})| + k \cdot |Var|^2 \cdot 4^{|Var|^2}\big)$ transitions.*

Proof (Sketch): For the sake of clarity, we consider here only the most significant rules of R^{-1}, which are of the form $X||Y \to t$ and apply them only nonlocally, i.e. as described in (B1). Let u and u' be two terms such that $u' \in \Box^*_{R^{-1}}(u)$, and s.t. there exist a paral-context C, n process variables $A_1,\ldots,A_n$, and n terms $t_1,\ldots,t_n$ such that $u = C[A_1,\ldots,A_n]$, and $u' = C[t_1,\ldots,t_n]$. The term u' is obtained from u after several rewritings on the leaves as follows: There exist intermediate terms $u_0,\ldots,u_k$, and a sequence $(i_1,j_1),\ldots,(i_k,j_k)$ of pairs of indices in $\{1,\ldots,n\}$ not necessarily distinct but satisfying the fact that $i_l \neq j_l$ for each $l \leq k$, such that $u_0 = u$, $u_k = u'$, and u_{l+1} is a successor of u_l by $\Box_{R^{-1}}$ obtained as follows: If u_l is of the form $u_l = C[s_1,\ldots,s_n]$, where the s_i's are terms, then there exists in R^{-1} a rule of the form $X_l||Y_l \to t$ (or $Y_l||X_l \to t$) such that $s_{i_l} = X_l$, $s_{j_l} = Y_l$, and $u_{l+1} = C[s'_1,\ldots,s'_n]$ with $s'_{i_l} = \tilde{0}$, $s'_{j_l} = t$, and $s'_i = s_i$ for all the other indices. This means that u_{l+1} is obtained by applying the rule $X_l||Y_l \to t$ (or $Y_l||X_l \to t$) at the positions (i_l,j_l) in the term u_l. Observe that t is either equal to some of the t_{j_l}'s appearing in u', or it is equal to a process variable B that will be rewritten later on a step $l' > l$ (this means that there exists a rule $B||B' \to s$ (or $B'||B \to s$) that can be applied at the positions $(i_{l'},j_{l'})$, where j_l is either equal to $i_{l'}$ or to $j_{l'}$.

The automaton has to recognize the term u' as a successor of u. For that, it starts from the leaves and proceeds upwards to the root. At each position i_l (resp. j_l), it guesses that the current node has, roughly speaking, "interacted" with the node j_l (resp. i_l) as described above. The automaton has to memorize all these guesses and validate them when it reaches the root of u'. The problem is that the number of these guesses is not bounded for all possible terms. Let us show how does the automaton behave to memorize all these guesses: First, the automaton has a counter c_X for each process variable X in Var. When it guesses that at the positions (i_l,j_l) a rule $X_l||Y_l \to t$ (or $Y_l||X_l \to t$) has been applied, the automaton decrements the counter c_{X_l} at position i_l, and increments this counter at position j_l. The idea is that, if the guesses are valid, the counters must be null at the root of the term u'. But this is not sufficient because the order in which rules are applicable is important. So, in addition, the automaton memorizes at each position $p \in \{1,\ldots,n\}$ a graph G whose edges are in $Var \cup \{\bot\} \times Var$ such that: (1) $\bot \to X_l$ is in G iff $p = j_l$ and $u_p \neq \tilde{0}$, and (2) $X_l \to Y_{l'}$ is in G iff $p = i_l = j_{l'}$ (this means that $u_p = \tilde{0}$).

After performing the guesses at the leaves, the automaton percolates them to the root by decorating the inner nodes as follows: if the term v_1 (resp. v_2) is decorated with the graph G_1 (resp. G_2) and the value $\mathbf{c}_1$ of the counters (resp. $\mathbf{c}_2$) (we consider the vector of counters $\mathbf{c} = (c_{V_1},\ldots,c_{V_m})$, where $Var = \{V_1,\ldots,V_m\}$), then $||(v_1,v_2)$ is decorated with $G_1 \cup G_2$ and $\mathbf{c} = \mathbf{c}_1 + \mathbf{c}_2$. Note that there is a finite number of these graphs (there are at most $2^{|Var|^2}$ possible graphs). The key point of the construction is that we can prove that the guesses that are performed by the automaton at the leaves are correct *if and only if* the automaton reaches the root of the tree in a configuration where all the counters are null, and the graph G satisfies the property: every vertex appearing in G is reachable from the vertex $\bot$.

□

7 Conclusion

We have investigated the problem of computing reachability sets of Process Rewrite Systems. We have considered PRSs with different operational semantics induced by several structural equivalences on terms. These equivalences correspond to the consideration of different combinations of properties of the used operators (associativity, commutativity, neutral element). We have shown that this allows to extend and unify the automata-based approaches developed in [BEM97,LS98,EHRS00,EP00] for pushdown and PA systems to more general classes of models.

In the full paper, we provide a translation from parallel programs, given as parallel flow graph systems, to PRS models. Through this translation, we show that our results for computing post* and pre* images for PRS modulo $\sim_s$ (Proposition 1 and Theorem 4) can be used to analyze precisely a significant class of parallel programs with recursive calls (and of course a restricted policy of communication).

Moreover, we provide an abstract translation from programs to PAD models which consists in abstracting away synchronizations, but keeping all informations about dynamic creation of processes, recursive calls, and return values of procedures. This translation refines the one given in [EP00] where return values of procedures are ignored. Therefore, through our PAD-based abstract semantics, our results of Theorems 5 and 6 can be used for a conservative analysis of parallel programs with procedure calls.

References

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

[BET03] A. Bouajjani, J. Esparza, and T. Touili. A Generic Approach to the Static Analysis of Concurrent Programs with Procedures. In *POPL'03*, 2003.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *LICS'90*, 1990.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. In *CAV'00*, volume 1885 of *LNCS*, 2000.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FOSSACS'99*. LNCS 1578, 1999.

[EP00] J. Esparza and A. Podelski. Efficient Algorithms for Pre * and Post * on Interprocedural Parallel Flow Graphs. In *POPL'00*, 2000.

[Esp02] J. Esparza. Grammars as processes. In *Formal and Natural Computing*. LNCS 2300, 2002.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*. volume 9 of ENTCS, 1997.

[GD89] R. Gilleron and A. Deruyver. The Reachability Problem for Ground TRS and Some Extensions. In *TAPSOFT'89*. LNCS 351, 1989.

[LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *CONCUR'98*. LNCS 1466, 1998.

[May98] R. Mayr. Decidability and complexity of model checking problems for infinite-state systems. Phd. thesis, TU Munich, 1998.