Bounded Communication Reachability Analysis of Process Rewrite Systems with Ordered Parallelism

Mihaela Sighireanu

LIAFA, *University of Paris 7, 2 place Jussieu, 75251 Paris cedex 5, France*
sighireanu@liafa.jussieu.fr

Tayssir Touili

LIAFA, *University of Paris 7, 2 place Jussieu, 75251 Paris cedex 5, France*
touili@liafa.jussieu.fr

Abstract

We define a new model called O-PRS that extends the Process Rewrite Systems formalism with a new associative operator, “⦶”, that allows to model parallel composition while keeping the order between parallel processes. Indeed, sometimes, it is important to remember the order between the parallel processes. The reachability problem of O-PRS being undecidable, we develop tree automata techniques allowing to build *polynomial* finite representations of (1) the *exact* reachable configurations in O-PRS modulo various equivalences that omit the associativity of “⦶”, and (2) *underapproximations* of the reachable configurations if the associativity of “⦶” is considered. We show that these underapproximations are *exact* if the number of communications between ordered parallel processes is bounded. We implemented our algorithms in a tool that was used for the analysis of a concurrent lexer server.

Key words: Multithreaded programs with procedure calls, synchronisation, process algebra, program analysis, verification.

1 Introduction

Analysis of concurrent software represents a major challenge in the model-checking community. Indeed, concurrent programs include various complex features such as (1) the manipulation of data ranging over unbounded domains, (2) the presence of recursive procedure calls, which can lead to an unbounded number of calls, (3) the dynamic creation of parallel processes, and (4) the existence of synchronization

Preprint submitted to Elsevier Preprint *20 July 2006*

statements. Ramalingam [Ram00] has shown that checking whether a given control point is reachable is undecidable, even if the program includes only recursive procedures and synchronisation statements. Therefore, to be able to analyse such programs, we need either to restrict ourselves to decidable subclasses, or to use approximative techniques.

During the last few years, several authors have addressed this issue. In particular, Process Rewrite Systems (PRS for short) [May98] have been successfully used in [BT03,BT05] to model and analyse such programs. 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 considers terms modulo a structural equivalence $\simeq$ which expresses the fact that 0 is a neutral element of “$\cdot$” and “$||$”, that “$\cdot$” is associative, and that “$||$” is associative and commutative.

To model a program in this framework, process variables are used to represent control points in the program, rules of the form $X \rightarrow X_1 \cdot X_2$ represent sequential recursive calls, whereas rules of the form $X \rightarrow X_1||X_2$ model dynamic creation of parallel processes. Moreover, rules of the form $X_1 \cdot X_2 \rightarrow X$ and $X_1||X_2 \rightarrow X$ allow to model some sort of communication between sequential and parallel processes, respectively. Therefore, due to the commutativity of the parallel composition “$||$”, PRS can only model programs where the order between the concurrent processes is not important. However, sometimes, it is important to keep the order between the parallel processes. This holds for example if the communication is done between processes that are neighbors. This is the case for example of the *concurrent lexer server* described in Section 5.

To overcome this restriction, we consider a new model, called O-PRS, that extends the PRS model with a new parallel operator ⦶ that is associative but not commutative, and hence it preserves the order between parallel processes. Note that O-PRS involves the two parallel operators “$||$” and “⦶” since a given program may involve the two kinds of communications: the ordered (⦶) and the unordered ($||$) one. Note also that ⦶ is different from the sequential composition “$\cdot$” since this latter has a prefix rewriting strategy, whereas ⦶ does not.

Unfortunately, while reachability between terms is decidable for PRS [May98], it becomes undecidable for O-PRS due to the associativity of ⦶ [GD89]. Despite this undecidability, we consider in this paper the reachability problem between two (infinite) sets of terms. Since process terms can be seen as trees, we consider representations of sets of terms based on (bottom-up) tree automata. To sidestep the undecidability result, we proceed as follows:

(i) First, we follow the approach used in [BT03] and perform the *exact* reachability analysis of O-PRS modulo restricted equivalences that omit the associativity of ⦶ (the cause of undecidability). Indeed, as discussed in [BT03], the reachability analysis modulo all the equivalences can be shown in many cases to be reducible to computing representatives of the reachability set modulo some stronger equivalence.

(ii) In case the associativity of $\oplus$ cannot be avoided, we compute representatives of *underapproximations* of the reachability sets by allowing the ordered processes to communicate only a fixed number of times k. These approximations enable the discovery of bugs in the system. Then, increasing k allows to compute better underapproximations. Moreover, if for k and $k+1$ the computed underapproximations are the same, then we know that we have computed an *exact representative of* the reachability set. Note that the underapproximations we compute are *exact* if the ordered processes can only perform a bounded number of communications. This is the case of our case study, which is a *real* example.

All the constructions that we give are polynomial. We implemented our algorithms in a prototype called PRESS. PRESS has been applied to several academic examples and to the interesting example called *the concurrent lexer server* [And91].

Related work. The results in this paper generalize those given in [LS98,EP00] for the PA case, and in [BT03] for PRS, where tree automata are computed to represent representatives of the reachability sets modulo different equivalences between terms.

Models based on communication via message passing have been considered in [BET03a,BET03b,BET05,Tou05,CCK+06]. However, all these models do not consider *ordered* parallel processes. In [BMOT05], a model called CDPN has been introduced. This model allows a "restricted ordered" communication where a process can only communicate with his children (the processes that he created). Our model allows *arbitrary* ordered communication between parallel processes.

The idea of performing reachability analysis while bounding the number of communications is known as *bounded context switch reachability*, and has been introduced in [QR05], and considered later in [BESS05]. However, in [QR05,BESS05] the parallel processes are not ordered.

Communication between ordered processes has been extensively studied in the context of parametrized systems verification using Regular Model Checking [ABJN99,BJNT00,BMT01,AJNd03]. These works do not consider dynamic creation of processes.

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 $\mathcal{X}$ be a fixed denumerable set of variables $\{x_1, x_2, \ldots\}$. The set $T_\Sigma[\mathcal{X}]$ of terms over Σ and $\mathcal{X}$ is the smallest set that satisfies: $\Sigma_0 \cup \mathcal{X} \subseteq T_\Sigma[\mathcal{X}]$, and if $k \geq 1$, $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma[\mathcal{X}]$, then $f(t_1, \ldots, t_k)$ is in

$T_\Sigma[\mathcal{X}]$. T_Σ stands for $T_\Sigma[\emptyset]$. Terms in T_Σ are called *ground terms*. A term in $T_\Sigma[\mathcal{X}]$ is *linear* if each variable occurs at most once. A *context* C is a linear term of $T_\Sigma[\mathcal{X}]$. 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 2.1 ([CDG+97]) *A* **tree automaton** *is a tuple $\mathcal{A} = (Q, \Sigma, F, \delta)$ 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, $\mathcal{A}$ is called a* finite tree automaton.

Let $\to_\delta$ be the move relation of $\mathcal{A}$ defined as follows: Given t and t' two terms of $T_{\Sigma \cup Q}$, then $t \to_\delta t'$ iff there exist a context $C \in T_{\Sigma \cup Q}[\mathcal{X}]$, 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 $t = C[f(q_1(t_1), \ldots, q_n(t_n))]$, and $t' = C[q(f(t_1, \ldots, t_n))]$, or (2) a rule $a \to q$ in δ, such that $t = C[a]$, and $t' = C[q(a)]$, or (3) a rule $q \to q'$ in δ, such that $t = C[q(u)]$, and $t' = C[q'(u)]$. Let $\xrightarrow{*}_\delta$ be the reflexive-transitive closure of $\to_\delta$. A term t is accepted by a state $q \in Q$ iff $t \xrightarrow{*}_\delta q(t)$. In this case, we say that t is annotated with q. Let L_q be the set of terms accepted by q. The language accepted by the automaton $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup\{L_q \mid q \in F\}$. A tree language is regular if it is accepted by a finite tree automaton.

The class of regular tree languages is closed under union, intersection, and complementation. Moreover, the emptiness problem of these automata can be solved in linear time.

3 Process Rewrite Systems with Ordered Parllelism

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:

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

Intuitively, 0 is the null process and "." denotes sequential composition, "||" denotes asynchronous parallel composition, and "$\oplus$" denotes the ordered parallel composition. We use both prefix and infix notations to represent process terms.

Definition 3.1 *A* Process Rewrite System with Ordered Parallelism *(O-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 PRS [May98] is an O-PRS without the "$\oplus$" operator. A* O-PAD *(resp.* PAD*) is a O-PRS (resp. PRS) where all the rules have no parallel composition "||" in the left hand sides of the rules.*

A O-PRS R induces a transition relation $\rightarrow_R$ over T_p defined by the following inference rules:

$$\frac{t_1 \rightarrow t_2 \in R}{t_1 \rightarrow_R t_2};\frac{t_1 \rightarrow_R t_1'}{t_1||t_2 \rightarrow_R t_1'||t_2};\frac{t_1 \rightarrow_R t_1'}{t_1 \obar t_2 \rightarrow_R t_1' \obar t_2};\frac{t_1 \rightarrow_R t_1'}{t_1 \cdot t_2 \rightarrow_R t_1' \cdot t_2};\frac{t_2 \rightarrow_R t_2'}{t_1||t_2 \rightarrow_R t_1||t_2'};$$

$$\frac{t_2 \rightarrow_R t_2'}{t_1 \obar t_2 \rightarrow_R t_1 \obar t_2'};\frac{t_1 \sim_0 0 \;\;,\;\; t_2 \rightarrow_R t_2'}{t_1 \cdot t_2 \rightarrow_R t_1 \cdot t_2'}$$

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. “$||$”, “$\obar$”, and “$\cdot$”:

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

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

A2: $\quad (t \cdot t') \cdot t'' \sim t \cdot (t' \cdot t'')$: associativity of “$\cdot$”,

A3: $\quad t||t' \sim t'||t$: commutativity of “$||$”,

A4: $\quad (t||t')||t'' \sim t||(t'||t'')$: associativity of “$||$”,

A5: $\quad (t \obar t') \obar t'' \sim t \obar (t' \obar t'')$: associativity of “$\obar$”.

We denote by $\sim_s$ the equivalence induced by the axioms A1 and A2, by $\sim_{\obar}$ the equivalence induced by A5, by $\sim_{\obar,s}$ the equivalence induced by the axioms A1, A2, and A5, and by $\simeq$ the equivalence induced by the axioms A1, A2, A3, and A4. For each equivalence $\equiv$, we denote by $[t]_\equiv$ the equivalence class modulo $\equiv$ of the process term t, i.e., $[t]_\equiv = \{t' \in T_p \mid t \equiv t'\}$. This definition is extended to sets of terms straightforwardly. A set of terms L is $\equiv$-*compatible* if $[L]_\equiv = L$. A set of terms L' is a $\equiv$-*representative* of L if $[L']_\equiv = L$. 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 $\overset{*}{\Rightarrow}_{\equiv,R}$ be the reflexive transitive closure of $\Rightarrow_{\equiv,R}$. Let $Post^*_{R,\equiv}(t) = \{t' \in T_p \mid t \overset{*}{\Rightarrow}_{\equiv,R} t'\}$. This definition is extended to sets of terms in the standard way. We omit the subscript $\equiv$ when it corresponds to the identity (=).

A O-PRS R is in *normal form* if $R = R' \cup R_{\obar}$ where R' is a PRS and $R_{\obar}$ is a set of rules of the form $t_1 \rightarrow t_2$ where t_1 and t_2 are either 0, X, or $X \obar Y$. Notice that the systems R' and $R_{\obar}$ are not independent in the sense that they can share process constants.

It can be shown, by adapting the proof of a very close fact in [May98], that for every O-PRS R over a set of process constants Var, it is possible to associate a O-PRS R' in normal form over a new set of process constants Var' (which extends Var

by some auxiliary process constants), and there exist two ground term substitutions S_1 and S_2 such that $Post^*_R = S_2 \circ Post^*_{R'} \circ S_1$. Therefore, we assume wlog in the remainder of the paper that O-PRS are always in normal form.

3.2 Reachability analysis problem

An O-PRS process term can be seen as a tree over the alphabet $\Sigma = \Sigma_0 \cup \Sigma_2$, where $\Sigma_0 = \{0\} \cup Var$ and $\Sigma_2 = \{., ||, \obar\}$. 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 $Post^*_{R,\equiv}(L_1) \cap L_2 \neq \emptyset$. Unfortunately, because of the associativity of the $\obar$ operator, it follows from [GD89] that:

Theorem 3.1 *The $\equiv$-reachability problem is undecidable for O-PRS if $\equiv \in \{\sim_{\obar}, \sim_{\obar,s}, \sim\}$. This holds even if L_1 and L_2 are single terms.* [1]

Therefore, the basic problems we consider in this paper is to compute, given a regular set L of terms, representations of the sets (resp. of underapproximations of the sets) $Post^*_{R,\equiv}(L)$ if $\equiv \in \{=, \sim_0, \sim_s, \simeq\}$ (resp. if $\equiv \in \{\sim_{\obar}, \sim_{\obar,s}, \sim\}$). More precisely, since these sets are in general not regular due to the associativity of "$\cdot$" and "$\obar$", and to the associativity-commutativity of "$||$", we will compute representatives of them. Indeed:

Lemma 3.1 *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 $L_1 \cap L_2 \neq \emptyset$.*

Therefore, computing regular $\equiv$-representatives of (approximations of) the $Post^*_{R,\equiv}$ images of regular sets allows to solve reachability problems.

4 Reachability Analysis of O-PRS

4.1 Reachability modulo term equality, $\sim_0$, $\sim_s$, and $\simeq$

It has been shown in [BT03] that if R is a PRS and L a regular set of PRS terms. Then:

- $Post^*_R(L)$ and $Post^*_{R,\sim_0}(L)$ are regular and effectively computable.
- A regular $\sim_s$-representative of $Post^*_{R,\sim_s}(L)$ can be effectively computed. This gives a $\simeq$-representative of $Post^*_{R,\simeq}(L)$ if R is a PAD since in this case, a $\sim_s$-representative of $Post^*_{R,\sim_s}(L)$ is also a $\simeq$-representative of $Post^*_{R,\simeq}(L)$ [BT03,Tou03].

Since in absence of the associativity of $\obar$ and of the associativity/commutativity of $||$, these two operators are similar, the constructions given in [BT03] can straightforwardly be extended to O-PRSs (simply by treating the new operator $\obar$ as "$||$"). Therefore, we get the following result:

[1] This is not the case for PRS even if "$\cdot$" is associative thanks to its prefix-rewriting semantics.

Theorem 4.1 *Let R be a O-PRS, L be a regular set of process terms, and $\mathcal{A}$ be a finite tree automaton that recognizes L. Then, we can effectively compute finite tree automata that recognize $Post^*_R(L)$, $Post^*_{R,\sim_0}(L)$, and a $\sim_s$-representative of $Post^*_{R,\sim_s}(L)$. Moreover, if R is a O-PAD, then we can effectively compute a finite tree automaton that recognizes a $\simeq$-representative of $Post^*_{R,\simeq}(L)$.*

4.2 Reachability modulo $\sim_{\obar}, \sim_{\obar,s}$, and $\sim$

As mentioned in Theorem 3.1, reachability is undecidable modulo $\sim_{\obar}$, $\sim_{\obar,s}$, and $\sim$. Therefore, we propose in this section to compute $\equiv$-representatives of *underapproximations* of the reachability sets $Post^*_{R,\equiv}(L)$ for $\equiv \in \{\sim_{\obar}, \sim_{\obar,s}, \sim\}$. The sets that we compute are underapproximations because we will allow to each process to communicate only a bounded number of times with his neighbors using the rules $X \obar Y \rightarrow t$. These underapproximations enable the discovery of bugs in the system.

Let us start with $\sim_{\obar}$. The main difficulty in reasoning modulo this equivalence comes from the fact that the rules of the form $X \obar Y \rightarrow t$ are not applied locally anymore. Indeed, so far such a rule is applied to a term u only if u has $X \obar Y$ as an explicit subterm. This is no longer the case when we consider terms modulo $\sim_{\obar}$. Indeed, this rule should be applied for instance to the terms $X \obar \big(Y \obar (Z \obar T)\big)$ and $X \obar \big((Y \obar Z) \obar T\big)$ since they are $\sim_{\obar}$-equivalent to $(X \obar Y) \obar (Z \obar T)$. One can argue that the same problem occurs with the rules of the form $X \cdot Y \rightarrow t$ when we consider the equivalence $\sim_s$. This is true, but the case of the operator $\obar$ is much more complicated due to the fact that the operator "$\cdot$" follows a prefix-rewriting strategy, whereas $\obar$ does not. To be able to handle this kind of rules of the form $X \obar Y \rightarrow t$, since due to the undecidability result it is impossible to compute a representative of the whole reachability set, we will compute a representative of the set of terms that are reachable from L by applying the rules of the previous kind an arbitrary number of times in different positions of the terms of L, while ensuring that at each leaf, only one rewriting occurs. Intuitively, since rules of the form $X \obar Y \rightarrow t$ model communication between ordered processes, this means that we will compute representatives of the sets of configurations that are reachable by allowing every process to communicate only once with his neighbors. We first show how we solve the problem when the system R contains only rules of the form above, and then, we show how to handle the general case.

4.2.1 The case where R has only rules of the form $X \obar Y \rightarrow t$

We suppose in this subsection that R contains only rules of the form $X \obar Y \rightarrow t$. We suppose w.l.o.g. that all the rules of R are of the form $X \obar Y \rightarrow Z \obar T$, where T is either a variable in Var, or the null process 0 (we write the rules of the form $X \obar Y \rightarrow Z$ as $X \obar Y \rightarrow Z \obar 0$).

Let L be a regular set of terms. We show in what follows how to compute a finite-state automaton that recognizes a representative of $Post^{\circledast}_{R,\sim_{\obar}}(L)$ defined as the set of terms that are reachable from L by applying the rules of R an arbitrary number of times in different positions of the terms of L, and such that at each leaf, only one rewriting occurs. To do so, let us introduce the notion of $\oplus$-*context*:

Definition 4.1 *Let $x,y \in \mathcal{X}$, a $\oplus$-*context *is a 2-variable context $C[x,y]$ such that there exist 2 single-variable contexts C_1 and C_2 such that: (1) $C[x,y] = \oplus(C_1[x],C_2[y])$, (2) x is the rightmost leaf of C_1, and y is the leftmost leaf of C_2, and (3) all the ancestors of the variables x and y in C_1 and C_2, respectively, are labeled by "$\oplus$".*

Then, modulo $\sim_\oplus$, a rule $X \oplus Y \to Z \oplus T$ can be applied to any term of the form $C[X,Y]$ for an $\oplus$-context C, to yield a term that is $\sim_\oplus$-equivalent to $C[Z,T]$. We define the relation Ξ_R that performs this transformation as follows: For every rule $X \oplus Y \to Z \oplus T$ in R, and every $\oplus$-context C, $\big(C[X,Y],C[Z,T]\big) \in \Xi_R$. This transformation is depicted on Figure 1, where the bold lines represent nodes labeled by "$\oplus$". It can be shown that $\Xi\ _R(L)$ is a $\sim_\oplus$-representative of $Post_{R,\sim_\oplus}(L)$.

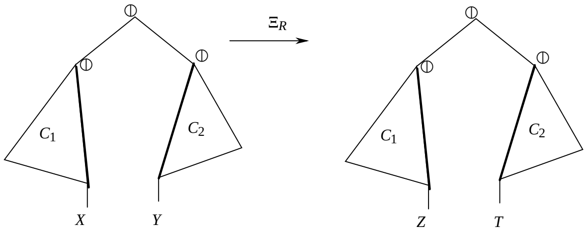


Fig. 1. Application of the rule $X \oplus Y \to Z \oplus T$ modulo $\sim_\oplus$

Let t be a term. We define $\Xi_R^{\circledast}(t)$ as the set of terms obtained by applying Ξ_R an arbitrary number of times to t, while ensuring that each leaf is rewritten at most once. This definition is extended to sets of terms in the obvious manner. We prove in what follows that for any regular language L, $\Xi_R^{\circledast}(L)$ is effectively regular.

$\Xi_R^{\circledast}(L)$ is effectively regular.

Let $R_1,\ldots,R_n$ be the different rules of R. Let $\mathcal{A} = (Q,\Sigma,F,\delta)$ be a tree automaton that recognizes L. We define the automaton $\mathcal{A}' = (Q',\Sigma,F',\delta')$ as follows:

- $Q' = Q \cup \{(q,R_i),(q,\overline{R_i}),(q,R_i\overline{R_j}) \mid q \in Q, R_i,R_j \in R\}$.
- $F' = F$.
- δ' contains δ and the following rules:

(α_1) If $R_i = X \oplus Y \to Z \oplus T$ is a rule of R, then:
(a) if $Y \xrightarrow{*}_\delta q(Y)$, then $T \to (q,R_{\underline{i}}) \in \delta'$;
(b) if $X \xrightarrow{*}_\delta q(X)$, then $Z \to (q,R_i) \in \delta'$.

(α_2) If $\oplus(q_1,q_2) \to q \in \delta$, then for every $R_i,R_j,R_k \in R$, we have:
(a) $\oplus\big((q_1,R_i),\underline{q}_2\big) \to (q,\underline{R}_i) \in \delta'$,
(b) $\oplus\big(q_1,(q_{\underline{2}},R_i\big) \to (q,R_i) \in \delta'$,
(c) $\oplus\big((q_1,R_i),(q_2,R_{\underline{i}})\big) \to q \in \delta'$,_
(d) $\oplus\big((q_1,R_i),(q_2,R_j)\big) \to (q,R_iR_j) \in \delta'$,

$(e)\ \ \oplus\big((q_1, R_i\overline{\underline{R_j}}), (q_2, R_j\overline{R_k})\big) \rightarrow (q, R_i\overline{R_k}) \in \delta'$,
$(f)\ \ \oplus\big((q_1, \underline{R_i}R_j), (q_2, \underline{R_j})\big) \rightarrow (q, \underline{R_i}) \in \delta'$,
$(g)\ \ \oplus\big((q_1, R_j), (q_2, R_jR_k)\big) \rightarrow (q, R_k) \in \delta'$.

The intuition behind the construction above is the following: If we consider the rewriting step depicted in Figure 1, the automaton $\mathcal{A}'$ needs to recognize the term on the right side as a successor of the term on the left side. To do so, it has to guess that in the place of the Z was an X, that in the place of the T there was a Y, and that these two positions were rewritten using a rule of the form $R_i = X \oplus Y \rightarrow Z \oplus T$. To do so, the automaton annotates the Z by state $(q_1, \overline{R_i})$ if $X \xrightarrow{*}_{\delta} q_1(X)$ (rule $\alpha_1 b$), and the T by state (q_2, R_i) if $Y \xrightarrow{*}_{\delta} q_2(Y)$ (rule $\alpha_1 a$). These guesses have then to reach the root of the term where they have to be validated. The role of the rules (α_2) is to propagate the guesses upward the terms until they are validated. Validation is done using the rules c, e, f, and g. States of the form $(q, R_i\overline{R_j})$ memorize two guesses. We do not need to memorize more because the rules are applied to adjacent subterms. So, at the end, we get that:

Lemma 4.1 *$t \xrightarrow{*}_{\delta'} q(t)$ iff $t \in \Xi_R^{\circledast}(L_q)$.*

The proof of this lemma follows the lines of the proofs given in [BT03]. It follows that:

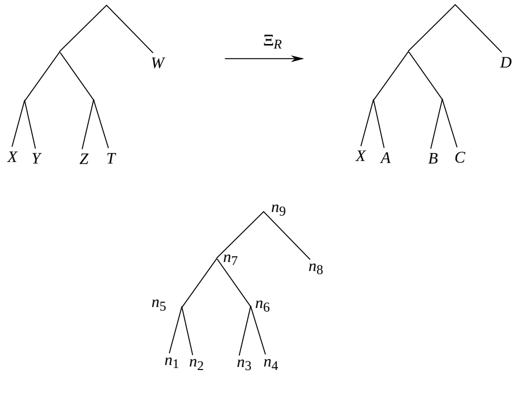
Theorem 4.2 *Let L be a regular set of process terms, and $\mathcal{A} = (Q, \Sigma, F, \delta)$ be a finite tree automaton that recognizes L. Then, $\Xi_R^{\circledast}(L)$ is recognized by the finite tree automaton $\mathcal{A}'$.*

Example.

Let us illustrate the construction above with an example. Let t be the term depicted on the left side of Figure 2 (the nodes are labeled with $\oplus$, we do not depict them for the sake of representation). Let $R = \{R_1, R_2\}$ where $R_1 = Y \oplus Z \rightarrow A \oplus B$ and $R_2 = T \oplus W \rightarrow C \oplus D$. t is recognized by the automaton having the states $Q = \{q_1, \ldots, q_9\}$, $F = \{q_9\}$, and the rules $\delta = \{X \rightarrow q_1, Y \rightarrow q_2, Z \rightarrow q_3, T \rightarrow q_4, W \rightarrow q_8, \oplus(q_1, q_2) \rightarrow q_5, \oplus(q_3, q_4) \rightarrow q_6, \oplus(q_5, q_6) \rightarrow q_7, \oplus(q_7, q_8) \rightarrow q_9\}$.

Then, the term on the right side of the figure will be recognized by the automaton $\mathcal{A}'$ constructed from $\mathcal{A}$ as follows:

- X is annotated with q_1, since $\delta \subseteq \delta'$;
- A is annotated with $(q_2, \overline{R_1})$, and C with $(q_4, \overline{R_2})$ using $\alpha_1 b$;
- B is annotated with (q_3, R_1), and D with (q_8, R_2) using $\alpha_1 a$;
- node n_5 is annotated with $(q_5, \overline{R_1})$ using $\alpha_2 b$;
- node n_6 is annotated with $(q_6, R_1\overline{R_2})$ using $\alpha_2 d$;
- node n_7 is annotated with $(q_7, \overline{R_2})$ using $\alpha_2 g$;
- Finally, node n_9 is annotated with q_9 using $\alpha_2 c$.



Names of the nodes

Fig. 2. An example

Since the root of the term is annotated by $q_9 \in F$, this means that the computed automaton recognizes this term as a successor of t.

Remark 4.1 *Observe that the fact that each leaf can be rewritten only once is crucial for our construction to work. Indeed, if we omit this condition, then the automaton would need an infinite number of states to perform the guesses, since at each leaf, an arbitrary number of rewritings (and therefore of guesses) can occur.*

4.2.2 Reachability analysis of arbitrary O-PRS

Let $R = R' \cup R_{\obar}$ be a O-PRS, where R' is a PRS and $R_{\obar}$ has rules involving only the operator $\obar$. Let $R^1_{\obar}$ be the rules of $R_{\obar}$ of the form $X \longrightarrow t$, and $R^2_{\obar}$ be the rules of $R_{\obar}$ of the form $X \obar Y \longrightarrow t$. Then, we can compute a representative of an underapproximation of $Post^*_{R,\sim_{\obar}}(L)$ where the ordered processes communicate a bounded number of times k. To do so, it suffices to compute $\left(\Xi^{\circledast}_{R^2_{\obar}}\left(Post^*_{R,=}(L)\right)\right)^k$ by applying the construction underlying Theorem 4.1 followed by the $\Xi^{\circledast}_{R^2_{\obar}}$ construction k times. Obviously, the obtained set is an underapproximation of $Post^*_{R,\sim_{\obar}}(L)$. Further, when increasing k, we can compute better underapproximations. Moreover, if for k and $k+1$ the computed underapproximations are the same, then we know that we have computed an *exact representative of* $Post^*_{R,\sim_{\obar}}(L)$.

The same principle can be applied to compute underapproximations of $\sim_{\obar,s}$-representatives of $Post^*_{R,\sim_{\obar,s}}(L)$. To do so, it suffices to apply the construction of Theorem 4.1 that produces $\sim_s$-representatives of $Post^*_{R,\sim_s}(L)$ instead of $Post^*_{R,=}(L)$. This gives us $\sim$-representatives of underapproximations of $Post^*_{R,\sim}(L)$ in the case of O-PADs.

5 A case study and experiments

We implemented our algorithms in a prototype called PRESS[ST]. PRESS has been applied to several academic examples and to the interesting example called *the concurrent lexer server* [And91].

5.1 The concurrent lexer server

We consider a multi-threaded server whose service is to do lexical analysis of texts it receives from clients. Each time that a client request arrives, i.e., a connection of the client to the server is successful, the server creates a thread that does the lexical analysis work and communicates the result to the client.

The main specificity of this example is that the lexical analysis is done in a concurrent manner, as suggested in [And91]. The advantage of concurrency is that it improves the complexity of the analysis.

Let us specify the work done by the lexical analyzer. We suppose that the input language of texts sent by clients is a simple language of arithmetical expressions, i.e., the input texts t_{in} are sequences of blanks, letters, digits, and arithmetical operators (e.g., $+, -, *$). The output of the analyzer should be an array of length equal to the length of t_{in}, each entry of the result being either `BL` (for blank), `ID` (identifier), `NUM` (for number) or `OP` (for arithmetical operator). Identifiers are C-like identifiers, i.e., they begin with a letter and may contain letters or digits. Numbers are strings built from digits only.

The implementation of the specification above is concurrent inside each thread launched by the server. A thread starts a number of sub-threads equal to the length of the input text. The sub-thread i computes the output for the i-th entry of the input text. If the i^{th} position of t_{in} is a blank, a letter or an operator, then the corresponding sub-thread terminates returning respectively `BL`, `ID`, `OP`. Otherwise, i.e., if it is a digit, the corresponding sub-thread has to wait for the result of its left neighbor. If this latter computed `BL`, `NUM`, or `OP`, then the sub-thread returns `NUM`; otherwise it returns `ID`.

5.2 The model

We give in this subsection the O-PRS model of the server example described above. Note that to be able to model this server of concurrent lexers, we need all the operators “$\cdot$”, “$||$”, and “⦶”. Indeed, the two first operators model the server, and “⦶” is needed to model the concurrency between the sub-threads. Indeed, the order between these sub-threads is important since each sub-thread corresponds to a position in the input arithmetical expression. Sub-threads work in parallel, but they maintain their ordering. Hence, the parallel operator $||$ cannot be used here since it is commutative. Note also that in this example, *at most two communications* are done between sub-tasks. Therefore, our algorithms compute representatives of the *exact* reachability sets.

The JAVA code below corresponds to a concurrent server that launches a new thread that does the lexical analysis for each new client request. The number of launched threads is unbounded.

```
public void server() {
    Socket socket;
    while(true) {
        try{
           socket=serverSocket.accept();
        } catch (Exception e){
          System.err(e);
          continue;
        }
        Thread t=new Thread(runLexerService(socket));
        t.start();
    }
}
```

The entry point of the server is represented by the process variable X. The server is waiting for connections. If a request for connection arrives (process variable Y), it may be successful (process variable T) or erroneous (process variable F). For successful connections, the server launches a thread in parallel (process variable L) and waits for another connection. In case of failures, the server simply loops to wait for another connection.

The above server can be modeled by the following rules:

$$X \rightarrow Y \,.\, X \qquad \text{server waits for a request for a connection}$$

$$Y \rightarrow T \qquad \text{successful request, } Y \text{ returns true}$$

$$Y \rightarrow F \qquad \text{failure request, } Y \text{ returns false}$$

$$T \,.\, X \rightarrow X \| L \qquad \text{new thread is launched if successful connection}$$

$$F \rightarrow 0 \qquad \text{request ignored if failure}$$

The O-PRS model of each thread of the server (process variable L) is given below. The thread starts in parallel a number of sub-threads (process variable P). The order in which these sub-threads are created is important since each sub-thread corresponds to a position in the input arithmetical expression. Sub-threads work in parallel, but they maintain their ordering. Hence, we need to use the new operator $\obar$ to model the parallelism between these subthreads, since it preserves the order.

Each process P reads its input. If the character read is an operator, a blank or a letter, it returns directly the result (process variables `OP`, `BL`, `ID`). Otherwise, it had read a digit and it has to wait for the result of its left brother: if it is an operator, a blank or a number, then it returns $\mathtt{NUM}$, if not, it returns identifier. The "$\obar$" operator allows to model communications between neighbor sub-tasks. It is important to remark that at most two communications are done between sub-tasks. Therefore, our algorithms compute representatives of the *exact* reachability sets. Note also that to model this example, we needed all the operators of O-PRS ("$\cdot$", "$\|$", and "$\obar$").

$L \longrightarrow \mathtt{BL} \obar \mathtt{loop}$	start to create sub-threads
$\mathtt{loop} \longrightarrow \mathtt{loop} \obar P$	create sub-threads
$\mathtt{loop} \longrightarrow P$	end of sub-thread creation
$P \longrightarrow \mathtt{OP}$	sub-thread reads and returns an operator
$P \longrightarrow \mathtt{BL}$	sub-thread reads and returns a blank
$P \longrightarrow \mathtt{ID}$	sub-thread reads a letter and returns identifier
$P \longrightarrow \mathtt{dig}$	sub-thread reads a digit
$\mathtt{OP} \obar \mathtt{dig} \longrightarrow \mathtt{OP} \obar \mathtt{NUM}$	digit sub-thread begins a number
$\mathtt{BL} \obar \mathtt{dig} \longrightarrow \mathtt{BL} \obar \mathtt{NUM}$	
$\mathtt{NUM} \obar \mathtt{dig} \longrightarrow \mathtt{NUM} \obar \mathtt{NUM}$	digit sub-thread belongs to a number
$\mathtt{ID} \obar \mathtt{dig} \longrightarrow \mathtt{ID} \obar \mathtt{ID}$	digit sub-thread belongs to an identifier

5.3 The analysis

Our aim is to check that the lexical analysis is correct, i.e., that the lexer does not output terms having $ID \obar NUM$ as subterms. The set of such terms can be represented by a finite tree automaton, and we need to check whether the intersection of the reachability set with these bad configurations is empty. To perform this, we applied our algorithms and we found that the intersection is indeed empty, which means that the program is correct. Indeed, our algorithms compute representatives of the *exact* reachability sets in this case since (1) the obtained model is a O-PAD, and (2) at most two communications are done between the ordered parallel processes.

6 Implementation and experiments

We implemented our algorithms in a tool called PRESS [ST]. The input of the tool is a O-PRS model, a tree automaton describing the set of initial process terms, and a list of tree automata describing the target reachability process terms.

We used the TIMBUK [Gen] library for the manipulation of tree automata. TIM-

Table 1
Experimental results for concurent lexer server.

	CLexer	**Server**
Size of *Post**	300 st./1167 trans.	644 st./2398 trans.
Size of min *Post**	28 st./424 trans	44 st./889 trans
Nb. of iterations	5	5
Execution time	10’51”	15’32”

BUK is written in OCAML and provides all the functions we needed for tree automata.

PRESS has been applied to several academic examples and to the concurrent lexer server described previously. For this last example, we considered two models:

- **Server**: full specification of the server of concurrent lexers with initial configuration X (process name corresponding to the entry point of the server), and
- **CLexer**: one thread specification with the initial configuration L.

The experimental results obtained are given on table 6. They have been obtained on a bi-processor Pentium with 4 Go of memory running on Linux. The execution time includes the computation of the set of reachable configurations, the minimization of this set, and its intersection with the target sets. However, in both cases, more than 90% of the execution time is taken by the reachability computation.

References

[ABJN99] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. *Lecture Notes in Computer Science*, 1633:134–150, 1999.

[AJNd03] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. Algorithmic improvements in regular model checking. In *CAV03*, volume 2725 of *LNCS*, pages 236–248, 2003.

[And91] Gregory Andrews. *Concurrent programming: principles and practice*. Addison-Wesley, 1991.

[BESS05] A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS*, 2005.

[BET03a] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL’03*. ACM, 2003.

[BET03b] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science*, 2003.

[BET05] A. Bouajjani, J. Esparza, and T. Touili. Reachability Analysis of Synchronised PA systems. In *INFINITY'04*. ENTCS, 2005.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *12th Intern. Conf. on Computer Aided Verification (CAV'00)*. LNCS, Springer-Verlag, 2000.

[BMOT05] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR'05*, LNCS, 2005.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation Rewriting and Algorithmic Verification. In *Proc. 17th Symp. on Logic in Computer Science (LICS'01)*. IEEE, 2001.

[BT03] A. Bouajjani and T. Touili. Reachability Analysis of Process Rewrite Systems. In *Proc. of FSTTCS 2003*, volume 2914, pages 74–87, 2003.

[BT05] Ahmed Bouajjani and Tayssir Touili. On Computing Reachability Sets of Process Rewrite Systems. In *RTA'05*. LNCS, 2005.

[CCK$^+$06] S. Chaki, E. Clarke, N. Kidd, T. Reps, and T. Touili. Verifying concurrent message-passing C programs with recursive calls. In *TACAS*, 2006.

[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.

[EP00] Javier Esparza and Andreas Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *Symposium on Principles of Programming Languages*, pages 1–11, 2000.

[GD89] R. Gilleron and A. Deruyver. The reachability problem for ground TRS and some extensions. In *Proc. Int. Joint Conf. Theory and Practice of Software Development (TAPSOFT'89)*, pages 227–243. LNCS 351, 1989.

[Gen] T. Genet. Timbuk. http://www.irisa.fr/lande/genet/timbuk/.

[LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, pages 50–66. LNCS 1466, 1998.

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

[QR05] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, 2005.

[Ram00] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22:416–430, 2000.

[ST] M. Sighireanu and T. Touili. The PRESS tool. http://www.liafa.jussieu.fr/∼sighirea/press/.

[Tou03] Tayssir Touili. *Analyse symbolique de systèmes infinis basée sur les automates: Application à la vérification de systèmes paramétrés et dynamiques*. PhD thesis, University of Paris 7, 2003.

[Tou05] T. Touili. Dealing with communication for dynamic multithreaded recursive programs. In *1st VISSAS workshop*, 2005. Invited Paper.