Int J Softw Tools Technol Transfer (2012) 14:145–165
DOI 10.1007/s10009-011-0208-8

Widening techniques for regular tree model checking

Ahmed Bouajjani · Tayssir Touili

Published online: 3 August 2011

Abstract We consider the framework of *regular tree model checking* where sets of configurations of a system are represented by regular tree languages and its dynamics is modeled by a term rewriting system (or a regular tree transducer). We focus on the computation of the reachability set $R^*(L)$ where R is a regular tree transducer and L is a regular tree language. The construction of this set is not possible in general. Therefore, we present a general acceleration technique, called *regular tree widening* which allows to speed up the convergence of iterative fixpoint computations in regular tree model checking. This technique can be applied uniformly to various kinds of transformations. We show the application of our framework to different analysis contexts: verification of parameterized tree networks and data-flow analysis of multithreaded programs. Parametrized networks are modeled by relabeling tree transducers, and multithreaded programs are modeled by term rewriting rules encoding transformations on control structures. We prove that our widening technique can emulate many existing algorithms for special classes of transformations and we show that it can deal with transformations beyond the scope of these algorithms.

Keywords Regular model checking · Widening · Automata

1 Introduction

Verification of infinite-state systems is motivated by the fact that many important features of complex systems, especially software systems, are beyond the scope of the finite-state framework. Among these features, we can mention: parametrization (systems are defined as networks with an arbitrary number of components), dynamic control structures (recursive calls and multithreading), and manipulation of unbounded data structures (counters, stacks, queues, etc.).

Regular model checking has been proposed as a general and uniform framework for reasoning about infinite-state systems [14,18,41,50]. In this framework, systems are modeled and analyzed using automata-based symbolic representations: configurations of the system are encoded as words or trees (of arbitrary sizes). This suggests the use of regular finite-state word/tree automata to represent sets of configurations, and the use of regular relations represented as word/tree transducers (or rewriting systems) to model the dynamics of the system, i.e., the transition relation between configurations. Then, verification problems based on performing reachability analysis are reduced to the computation of closures of regular languages under regular word/tree transducers (rewriting systems), i.e., given a regular relation R and a regular language L, compute $R^*(L)$, where R^* is the reflexive-transitive closure of R. A more general problem is to construct a representation of the relation R^* as a finite transducer. This problem is harder than the previous one: there are regular relations having nonregular transitive closures, but under which subclasses of regular languages are effectively closed (see, e.g., [17]). Computing $R^*(L)$ is impossible in general since the transition relation of any Turing machine is a regular word transduction. Therefore, the main issue in regular model checking is (1) to determine classes of regular languages L and relations R such that the closure $R^*(L)$ is effectively constructible, and (2) to find accurate and powerful fixpoint acceleration techniques which help the convergence of language closures (reachability analysis) in the general case.

A. Bouajjani · T. Touili (✉)
LIAFA, CNRS and Univ. Paris Diderot,
Case 7014, 2 Place Jussieu, 75251 Paris 5, France
e-mail: Tayssir.Touili@liafa.jussieu.fr; touili@liafa.jussieu.fr

A. Bouajjani
e-mail: Ahmed.Bouajjani@liafa.jussieu.fr

During the last years, several authors addressed this issue. (1) First in the case of regular *word* model checking where configurations are encoded as words. These works have been successfully applied to reason about linear parameterized systems (i.e., parameterized systems where the processes are arranged in a linear topology) [3,5,6,14,17,29,40,45,46], as well as systems that operate on linear unbounded data structures such as lists, integers, reals, and even hybrid automata [13,15,16], and programs with pointers [11]. (2) Then, in the case of regular *tree* model checking where configurations are represented by trees of arbitrary sizes (but fixed arities). These works have been applied to the analysis of parameterized systems with tree topologies [4,7,20,29], and multithreaded programs [10,20,21,42,48]; and (3) in the case of *hedge* tree model checking, where the configurations of the system are represented by trees of arbitrary width [32,49].

In this paper, we consider the case of regular tree model checking. Indeed, tree-like structures are very common and appear naturally in many modeling and verification contexts. We consider in this paper two of such contexts: verification of parameterized networks with tree-like topologies, and data flow analysis of multithreaded programs.

Indeed, in the case of parameterized tree networks, labeled trees of arbitrary height represent configurations of networks of arbitrary numbers of processes: each vertex in a tree corresponds to a process, and the label of a vertex is the current control state of its corresponding process. Typically, actions in such parameterized systems are communications between processes and their sons or fathers. These actions correspond in our framework to tree relabeling rules (transformations which preserve the structure of the trees). Examples of such systems are multicast protocols, leader election protocols, mutual exclusion protocols, etc.

In the case of multithreaded programs, trees represent control structures recording the names of the procedures to call, and the sequential/parallel order in which they must be called. These structures are of unbounded size and are transformed dynamically after the execution of each action of the program (e.g., recursive call, launching a new thread, etc.). Such actions correspond in our framework to tree transformations represented as tree transductions (or tree rewriting rules). Note that, in contrast with the previous case (parameterized systems), these tree transformations are not tree relabelings, but transformations which modify the structures of the trees (non structure-preserving transformations).

Therefore, our aim in this work is to provide algorithmic techniques which allow us to compute automatically closures of regular tree languages under regular tree transformations, and when possible, to compute transitive closures of regular tree transformations. Moreover, we want to define *general* techniques which can deal with different classes of relations, and which can be applied *uniformly* in many verification and analysis contexts such as those mentioned above.

The main contribution of our work is the definition of a general acceleration technique on tree automata called *regular tree widening*. Our technique is based on comparing languages (automata) obtained by successive applications of a transformation R to a language L in order to guess automatically the limit $R^*(L)$. The guessing technique we introduce is based on detecting regular growths in the structures of the automata. A test is performed to check automatically whether the guess covers all the reachable configurations, i.e., whether the set $R^*(L)$ is included in the guessed language. The same test ensures in many interesting cases that the guess is exact, i.e., that the guessed language is precisely $R^*(L)$. This technique can also be applied in order to compute iteratively transitive closures of relabeling transducers.

We show that the iterative computation of closures enhanced with regular tree widening yields a quite general and accurate reachability analysis procedure which can be applied uniformly to various analysis problems. We illustrate this by showing the application of this procedure to the analysis of parameterized systems and to the analysis of multithreaded programs. Moreover, we prove that this procedure is powerful and accurate enough to compute precisely the reachability sets for many significant classes of systems, covering several classes for which there exist different specialized algorithms.

First, we consider the case of parameterized networks. We consider a particular class of models based on term rewriting systems called *well-oriented systems*. These models correspond to systems where, typically, information is exchanged between a set of processes (the leaves of the tree) and another process (the root of the tree) through a tree-like network, assuming that the state of each process is modified after the transmission of a message (this corresponds for instance to the fact that paths followed by messages are marked, messages are memorized by routers, etc.). We assume, moreover, that the system has a finite number of ascending and descending phases. These assumptions are quite realistic and many protocols and parallel algorithms running on tree-like topologies (e.g., leader election protocols, mutual exclusion protocols, parallel boolean algorithms, etc.) have these features (for instance, requests are generated by leaves and go up to the root, and then answers or acknowledgements are generated by the root and go down to leaves following some marked paths).

We prove that for every well-oriented system, the transitive closure is regular and we provide a transducer characterizing this closure. Then, we prove that our widening techniques can simulate the direct construction we provide for well-oriented systems.

Then, we address the issue of analyzing multithreaded programs using regular tree model checking. We consider programs with recursive calls, dynamic creation of processes, and communication. We adopt the approach advocated in

[21,22,33] which consists in reducing data flow analysis to reachability analysis problems for process rewriting systems (PRS) [43]. Programs are described as term rewriting rules of the form $t \rightarrow t'$ where t and t' are terms built up from process variables, sequential composition, and asynchronous parallel composition.

We show that reachability analysis with regular tree widening terminates and computes precisely the set of reachable configurations in the case of PA rewriting systems (when all the left-hand-sides of the rules are process variables). Hence, our techniques cover the case considered in [42,35] and can handle programs which are beyond the scope of these algorithms (e.g., a concurrent server). Actually, we prove a more general result. We prove that our procedure terminates and computes the exact reachability set (at least in the case) of any PRS R such that R is well founded. This is for instance the case of the concurrent server. The completeness result for PA follows from the fact that we can transform any PA into an equivalent one (w.r.t. reachability) which has this property.

Finally, we show that *regular tree widening* computes the exact reachability set in the case of well-founded *linear semi-monadic systems* and *ground term rewriting systems*. These systems are already known to preserve regularity [31,26]. In this paper, we show that our widening techniques can compute the reachability sets of these systems if they are well founded.

Related work The idea of widening operation is inspired by works in the domain of abstract interpretation [23] where it is mainly used for systems with numerical data domains (integers or reals) [28]. There are numerous works on tree-like representations of program configurations or schemes. Probably, the first work using tree automata and tree transducers for symbolic reachability analysis of systems (especially parameterized systems) is Kesten et al. [41]. However, no acceleration techniques are provided in that work.

Our regular tree widening technique is an extension of the widening techniques for word automata defined in [14,46]. In [15], widening techniques are also applied to compute transitive closures of *word* transducers. There are mainly three differences between our technique and the one of Boigelot et al. [15]. First, to apply widening, we compare at each step every transducer with its immediate successor, whereas Boigelot et al. [15] compare transducers according to some period. The second difference is that Boigelot et al. [15] use minimal automata and compare the automata languages to detect the increments, whereas our technique compares the structures of the automata. A last difference is that our technique applies to trees, whereas as far as we know, the theory of Boigelot et al. [15] was not extended to trees.

Computing transitive closures of relabeling tree transducers have been considered in [4,7]. The techniques presented in these papers are different from ours: They are based on collapsing equivalent states, whereas our widening techniques are based on detecting growths, and then extrapolating by adding loops on these growths. Moreover, the techniques of these papers can only be applied to *structure preserving* tree transformations, whereas our techniques can also be applied to *non structure preserving* ones.

Genet et al. [8,38] defined a technique that can be applied to compute an over-approximation of the reachability set $R^*(L)$ of a regular language L and a rewriting system R. The technique presented in [8,38] is different from our widening techniques. It is based on applying a saturation procedure to the different tree automata representing $R^i(L)$. This adds new rules and new states to these automata. Thus, termination is not guaranteed. To enforce termination, the authors consider approximation functions that reduce the number of states that can be added. The precision of the over-approximation depends on the approximation function. The main limitation of this technique is that the approximation function has to be provided by the user, whereas our widening techniques are completely automatic and do not necessitate the help of the user.

In [12,13], abstract regular tree model checking is introduced (ARTMC). This technique uses abstract fixpoint computations in some *finite* domain of automata. The abstract fixpoint computation always terminates. If the result is not precise enough, a refinement procedure is applied over another finite domain of automata. This refinement procedure may not terminate. As in our case, the abstraction is done by collapsing automata states. In our widening techniques, states are collapsed if some growth is detected, whereas in [12,13], they are collapsed according to an equivalence relation defined by a finite set of regular predicate languages. We believe that our widening techniques can be more precise than ARTMC since the automata produced by widening do not have to belong to a *finite* domain, whereas the ones computed by ARTMC do.

Dams et al. [29] presents techniques for computing transitive closures for word transducers. The possibility of extending this technique to trees was mentioned in the conclusion of Dams et al. [29]; however, no further details are given.

Our regular tree widening techniques were defined in [20]. The current paper is a more detailed journal version of [20].

2 Preliminaries

2.1 Trees and terms

An alphabet Σ is ranked if it is endowed with a mapping $rank : \Sigma \rightarrow \mathbb{N}$. For $k \geq 0$, Σ_k is the set of elements of rank k. The elements of Σ_0 are called constants.

Let $\mathcal{X}$ be a denumerable set of symbols called variables. The set $T_\Sigma[\mathcal{X}]$ of terms over Σ and $\mathcal{X}$ is defined inductively as follows:

- If $f \in \Sigma_0$, then $f \in T_\Sigma[\mathcal{X}]$,
- If $x \in \mathcal{X}$, then $x \in T_\Sigma[\mathcal{X}]$,
- 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 $x_1, \ldots, x_n$ be the variables occurring in C and 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 $1 \leq i \leq n$.

A term in $T_\Sigma[\mathcal{X}]$ can be viewed as a rooted labeled tree where an internal node with n sons is labeled by a symbol from Σ_n, and the leaves are labeled with variables and constants. Therefore, if t is the term $f(t_1, \ldots, t_k)$, we let $root(t) = f$. Throughout the paper, we shall often make no distinction between a term of $T_\Sigma[\mathcal{X}]$ and the tree that corresponds to it.

2.2 Tree automata and transducers

Let $\mathcal{X}$ be a fixed denumerable set of variables $\{x_1, x_2, \ldots\}$. We shall use x and y to denote arbitrary elements of $\mathcal{X}$.

Definition 1 A **bottom-up tree automaton** (we shall omit 'bottom-up') is a tuple $\mathcal{A} = (Q, \Sigma, F, \delta)$ where Q is a finite 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

$$f\big(q_1, \ldots, q_n\big) \to q \qquad (1)$$

$$a \to q \qquad (2)$$

$$q \to q' \qquad (3)$$

where $a \in \Sigma_0$, $n \geq 1$, $f \in \Sigma_n$, and $q_1, \ldots, q_n, q, q' \in Q$.

Let t be a ground term. A run of $\mathcal{A}$ on t can be done in a bottom-up manner as follows: first, we assign a state to each leaf according to the rules (2), then for each node, we must collect the states assigned to all its children and then associate a state to the node itself according to the rules (1). Formally, 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 $t = C[f\big(q_1(t_1), \ldots, q_n(t_n)\big)]$, and $t' = C[q\big(f(t_1, \ldots, t_n)\big)]$, 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 $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} L_q$. A tree language is regular if it is accepted by a finite tree automaton.

Definition 2 A **bottom-up tree transducer** (we shall omit 'bottom-up') is a tuple $\mathcal{T} = (Q, \Sigma, \Sigma', F, \delta)$ where Q is a finite set of states, Σ and Σ' (the sets of input and output symbols) are ranked alphabets, $F \subseteq Q$ is a set of final states, and δ is a set of rules of the form:

$$f\big(q_1(x_1), \ldots, q_n(x_n)\big) \to q(u),\ u \in T_{\Sigma'}[\{x_1, \ldots, x_n\}] \qquad (4)$$

$$q(x) \to q'(u),\ u \in T_{\Sigma'}[\{x\}] \qquad (5)$$

$$a \to q(u),\ u \in T_{\Sigma'} \qquad (6)$$

where $a \in \Sigma_0, n \geq 1, f \in \Sigma_n, x, x_1, \ldots, x_n \in \mathcal{X}$, and $q_1, \ldots, q_n, q, q' \in Q$.

Given an input term t, $\mathcal{T}$ proceeds as previously: it begins by replacing some leaves according to the rules (6). For instance, if a leaf is labeled a and the rule $a \to q(u)$ is in δ, then a is replaced by $q(u)$. The substitution proceeds then towards the root. If the rule $f\big(q_1(x_1), \ldots, q_n(x_n)\big) \to q(u)$ is in δ, then $\mathcal{T}$ replaces an occurrence of a subtree $f\big(q_1(t_1), \ldots, q_n(t_n)\big)$ by the term $q(u[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n])$, where each occurrence of the variable x_i in t is replaced by t_i. The computation continues until the root of t is reached.

The transducer $\mathcal{T}$ defines the following *regular* relation between trees $R_\mathcal{T} = \{(t, t') \in T_\Sigma \times T_{\Sigma'} \mid t \xrightarrow{*}_\delta q(t'),$ for some $q \in F\}$. Let R be a tree relation over T_Σ, we denote by R^n the composition of R with itself n times. Let $R^{\leq n} = \bigcup_{n \geq j} R^j$. As usual, $R^* = \bigcup_{n \geq 0} R^n$ denotes the reflexive-transitive closure of R. Let $L \subseteq T_\Sigma$ be a tree language. $R(L)$ is the set $\{t' \in T_{\Sigma'} \mid \exists t \in L, (t, t') \in R\}$.

Definition 3 A transducer is **linear** if all the right hand sides of its rules are linear (no variable occurs more than once).

We restrict ourselves to linear tree transducers since they are closed under composition whereas general transducers are not [25,34].

Proposition 1 *Let $\mathcal{T}$ be a linear tree transducer and $\mathcal{L}$ be a regular tree language. Then, $R_\mathcal{T}(\mathcal{L})$ is regular and effectively constructible.*

Particular cases of linear transducers are *relabeling tree transducers*.

Definition 4 A transducer is called a **relabeling** if all its rules are of the form

$$f\big(q_1(x_1), \ldots, q_n(x_n)\big) \to q\big(g(x_1, \ldots, x_n)\big) \qquad (7)$$

$$a \to q(b) \qquad (8)$$

$$q(x) \to q'(x) \qquad (9)$$

where $f, g \in \Sigma_n$ and $a, b \in \Sigma_0$.

Note that relabeling tree transducers preserve the structure of the input tree. A relabeling $(Q, \Sigma, \Sigma', F, \delta)$ can be seen as a tree automaton over the product alphabet $\Sigma \times \Sigma'$. The rules (7) can then be written $f/g(q_1, \ldots, q_n) \to q$, the rules (8) can be written $a/b \to q$, and the rules (9) can be written $q \to q'$.

2.3 Tree automata and hypergraphs

Definition 5 Let V be a set of vertices and Σ be a ranked alphabet. Let $f \in \Sigma_n$ and $v, v_1, \ldots, v_n \in V$; the tuple $(v, f, v_1, \ldots, v_n)$ is a **hyperedge** labeled by f and connecting in order v to the vertices $v_1, \ldots, v_n$. We will write $v \xrightarrow{f} v_1, \ldots, v_n$ for every hyperedge $(v, f, v_1, \ldots, v_n)$, or just $v \xrightarrow{a}$ if $a \in \Sigma_0$. A **hypergraph** is a pair $\mathcal{G} = (V, H)$ where V is a set of vertices and H a set of hyperedges on V.

Given a bottom-up tree automaton $\mathcal{A} = (Q, \Sigma, F, \delta)$, the transition relation δ can be represented by the hypergraph $\mathcal{G}_\delta = (Q, H_\delta)$, where H_δ is defined by:

- $q \xrightarrow{f} q_1, \ldots, q_n \in H_\delta$ for every rule $f(q_1, \ldots, q_n) \rightarrow q$ in δ.
- $q \xrightarrow{a} \in H_\delta$ for every rule $a \rightarrow q$ in δ.
- $q \rightarrow q' \in H_\delta$ for every rule $q \rightarrow q'$ in δ.

All operations on tree automata can be defined on hypergraphs. In the remainder of the paper, a tree automaton (tree relabeling transducer) will be represented by a pair $(\mathcal{G}, F)$, where $\mathcal{G}$ is the hypergraph that represents its transition relation and F is the set of final states.

3 Regular tree model checking

We recall in this section regular tree model checking. Let P be a program. Sets of configurations of P are represented by regular tree languages (sets of terms) and the dynamics of the system is represented by a term rewriting system. Each rewrite rule can be seen as a regular tree transformation and corresponds to a tree transducer. Thus a program P can be represented by a pair $(R = \{R_1, \ldots, R_n\}, L_0)$, where L_0 is a regular tree language representing the set of initial configurations of P, and the R_i's are the rewrite rules corresponding to the different actions of the program. Then, verification problems can be reduced to reachability analysis which consists in computing the set $R^*(L_0)$. Clearly, this problem is undecidable in general (the transition relation of any Turing machine is a regular word relation). Then, the problem we address is to find classes of relations (rewrite systems) R such that $R^*(L_0)$ is effectively computable for any regular language L_0. More generally, we are interested, when possible, in computing transitive closures R^* of regular relations.

Since we cannot have algorithms for computing R^* or $R^*(L_0)$ in the general case (the transition relation of any Turing machine is a regular word transduction), we adopt a semi-algorithmic approach (which is not guaranteed to terminate) based on an iterative computation of closures, and on the use of acceleration techniques in order to help termination. These acceleration techniques consist in computing closures for subrelations of R. Typically, if we are able to compute R_i^* images for some of the rules in R, we can use these partial closure computations to speed up the convergence of the global fixpoint computation of $R^*(L_0)$. The computation of R_i^* images can be done, for instance, using algorithms for special classes of rules. In general, we are interested in defining powerful acceleration techniques allowing to compute precisely the set $R^*(L)$, or accurate upper approximations of it, for wide classes of systems. For this aim, we propose in the next section a general extrapolation technique called *regular tree widening*.

4 Widening techniques on tree automata

We define hereafter an extrapolation technique on tree automata called *regular tree widening* which allows to compute the limit of a sequence of tree sets obtained by iterating tree transformations.

4.1 Principle

The technique we present generalizes the one we have introduced in [14,46] in the case of word automata. The principle proposed in these previous works is based on the detection of growths during the iterative computation of the sequence $L, R(L), R^2(L), \ldots$, in order to guess $R^*(L)$. For instance, if the situation $L = L_1 L_2$ and $R(L) = L_1 \Delta L_2$ occurs, where L_1, L_2 and Δ are regular word languages, then we guess that iterating R will produce $L_1 \Delta^* L_2$. In some cases, it is possible to decide whether our guess is correct. Here, we extend this principle to the case of tree languages. The detection of growths is performed on the hypergraph structures of the tree automata recognizing the computed sequence of languages.

Definition 6 Let $\mathcal{G} = (V, H)$ be a hypergraph and $F \subseteq V$ be a set of accepting vertices. Then, a **hypergraph bisimulation** is a symmetrical binary relation $\rho \subseteq V \times V$ such that, for every $v, v' \in V$, $(v, v') \in \rho$ iff

- $v \in F$ iff $v' \in F$,
- for every hyperedge $v \xrightarrow{f} v_1, \ldots, v_n \in H$, there exists a hyperedge $v' \xrightarrow{f} v'_1, \ldots, v'_n \in H$ such that, for every $i \in \{1, \ldots, n\}$, $(v_i, v'_i) \in \rho$.

We write $v \sim v'$ if there exists a hypergraph bisimulation relating v and v'. Given two tree automata $\mathcal{A} = (\mathcal{G}, F)$ and $\mathcal{A}' = (\mathcal{G}', F')$, we write $\mathcal{A} \sim \mathcal{A}'$ iff every vertex in F is bisimilar to a vertex in F' and vice versa.

Definition 7 Suppose that we are given:

- a sub-hypergraph of $\mathcal{G}$: $\Delta = (V_\Delta, H_\Delta)$ ($V_\Delta \subseteq V$, and $H_\Delta \subseteq H$),

- two subsets of V_Δ: $\mathcal{I}_\Delta$ and $\mathcal{O}_\Delta$ called *entry* and *exit* vertices,
- φ: a partition of $\mathcal{I}_\Delta \cup \mathcal{O}_\Delta$.

Let $\sim_\varphi$ denote the equivalence relation induced by φ. We assume moreover that $(\sim_\varphi \cap \mathcal{O}_\Delta \times \mathcal{O}_\Delta) \subseteq \sim$ (i.e., non bisimilar exit vertices are not $\sim_\varphi$-equivalent).

Then, we define two hypergraphs $\mathcal{G}\backslash_\varphi\Delta$ and $\mathcal{G}[\Delta \leftarrow \Delta^+]$ as follows:

- $\mathcal{G}\backslash_\varphi\Delta$ is the hypergraph (V', H') such that
  - $V' = V\backslash V_\Delta \cup \{[v]_\varphi \mid v \in \mathcal{I}_\Delta \cup \mathcal{O}_\Delta\}$ and
  - $H' = \{v'_0 \xrightarrow{f} v'_1, \ldots, v'_n \mid v_0 \xrightarrow{f} v_1, \ldots, v_n \in H, \forall i. v_i \notin V_\Delta\backslash(\mathcal{I}_\Delta \cup \mathcal{O}_\Delta)$, and if $v_i \in \mathcal{I}_\Delta \cup \mathcal{O}_\Delta$ then $v'_i = [v_i]_\varphi$, otherwise $v'_i = v_i\}$

  where $[v]_\varphi$ denotes the $\sim_\varphi$-equivalence class of the vertex v. Intuitively, $\mathcal{G}\backslash_\varphi\Delta$ is the hypergraph obtained from $\mathcal{G}$ by removing all hyperedges in Δ, and collapsing $\sim_\varphi$-equivalent vertices.
- $\mathcal{G}[\Delta \leftarrow \Delta^+]$ is the hypergraph (V, H'') where: $H'' = \{v'_0 \xrightarrow{f} v'_1, \ldots, v'_n \mid v_0 \xrightarrow{f} v_1, \ldots, v_n \in H$ s.t. $v'_i = v_i$ if $v_i \notin \mathcal{I}_\Delta \cup \mathcal{O}_\Delta$, and $v'_i = [v_i]_\varphi$ otherwise$\}$. Intuitively, $\mathcal{G}[\Delta \leftarrow \Delta^+]$ is obtained by adding loops allowing to iterate Δ (by collapsing entry and exit vertices according to $\sim_\varphi$).

Now, we are able to define the *regular widening* operation on tree automata:

Definition 8 (*Regular tree widening*) Let $\mathcal{A} = (\mathcal{G}, F)$ and $\mathcal{A}' = (\mathcal{G}', F')$ be two tree automata. Then, given a subhypergraph Δ of $\mathcal{G}'$, sets of entry and exit vertices $\mathcal{I}_\Delta$ and $\mathcal{O}_\Delta$, and a partition φ of $\mathcal{I}_\Delta \cup \mathcal{O}_\Delta$ such that $(\sim_\varphi \cap \mathcal{O}_\Delta \times \mathcal{O}_\Delta) \subseteq \sim$, if

$$(\mathcal{G}, F) \sim (\mathcal{G}'\backslash_\varphi\Delta, F') \tag{10}$$

then we define the **widening** operator $\nabla(\mathcal{A}, \mathcal{A}', \Delta, \varphi) = (\mathcal{G}'[\Delta \leftarrow \Delta^+], F')$.

Note that the same widening principle can be applied in the case of relabeling tree transducers (in order to compute iteratively transitive closures of relabeling transducers).

Example 1 Consider the following term rewriting rule: $R = a \rightarrow f(a, b)$ and assume we want to compute $R^*(a)$. Let $(\mathcal{G}_0, \{q_0\})$, $(\mathcal{G}_1, \{q_2\})$, and $(\mathcal{G}_2, \{q_3\})$ be tree automata recognizing a, $R(a)$, and $R^2(a)$. Their corresponding hypergraphs are depicted in Fig. 1.

By comparing $\mathcal{G}_1$ and $\mathcal{G}_2$, we detect a widening situation where Δ is the hypergraph $(\{q_1, q_2, q_3\}, \{q_3 \xrightarrow{f} q_2, q_1\})$, $\mathcal{I}_\Delta = \{q_3\}$, $\mathcal{O}_\Delta = \{q_1, q_2\}$, and $\varphi = \{\{q_1\}, \{q_2, q_3\}\}$. Then, the widening operator ∇ yields an automaton $(\mathcal{G}, \{q_2\})$

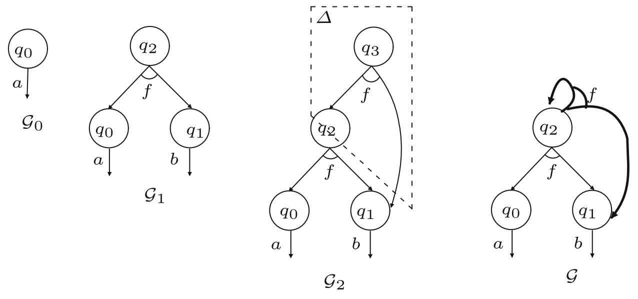

Fig. 1 Illustration of the regular tree widening mechanism

obtained by collapsing the states q_2 and q_3, thus adding the loop drawn by thick lines. This automaton defines precisely $R^{\geq 1}(a)$ (its union with the automata of the previous steps corresponds to $R^*(a)$).

Remark 1 Note that the fact that we do not merge non bisimilar exit vertices (i.e., $\sim_\varphi \cap \mathcal{O}_\Delta \times \mathcal{O}_\Delta \subseteq \sim$) allows to discard many "bad" widening candidates Δ. To illustrate that, let us consider the example of Fig. 2, where the graphs $\mathcal{G}$ and $\mathcal{G}'$ correspond respectively to the regular languages L and $R(L)$ for some rewriting system[1] R and some regular language L. Then if we compare these two graphs, we detect as growth the subgraph Δ drawn inside the dashed box. If we consider $\mathcal{I}_\Delta = \{s_2\}$, $\mathcal{O}_\Delta = \{s'_2, s''_2\}$, and $\varphi = \{s_2, s'_2, s''_2\}$ (this is not possible since s'_2 and s''_2 are two exit vertices which are not bisimilar), then the widening operator ∇ yields the automaton $\mathcal{G}''$, and guesses that this automaton corresponds to $R^*(L)$. But, this is too rough. Indeed, after executing d we cannot execute a, however this is allowed in $\mathcal{G}''$.

Performing a widening operation requires finding a widening situation, i.e., a subgraph Δ and a partition φ satisfying the condition (10).

The detection of candidates Δ can be done effectively by performing a product between the two compared hypergraphs $\mathcal{G}$ and $\mathcal{G}'$, and guessing nondeterministically the entries and the exits of Δ. More precisely, we can build a kind of synchronous product between $\mathcal{G}$ and $\mathcal{G}'$: we progress simultaneously in both $\mathcal{G}$ and $\mathcal{G}'$, and at each step, we can choose nondeterministically to continue progressing in both $\mathcal{G}$ and $\mathcal{G}'$ (if we can), or stop moving in $\mathcal{G}'$: this is the entry of Δ. Then, we guess the exit of Δ whenever it is possible to resume progressing in both $\mathcal{G}$ and $\mathcal{G}'$. Efficient (but incomplete) strategies can be adopted in order to reduce nondeterminism (the number of candidates Δ). For example, one strategy is to consider the entry of Δ only when it is no more possible to progress in both $\mathcal{G}$ and $\mathcal{G}'$.

[1] Consider for example the rewriting system R that corresponds to the rules $ex \rightarrow ecx$, $x \in \Sigma^*(a+b)$ and $ex \rightarrow edx$, $x \in \Sigma^*b$, expressing respectively that e can be rewritten into ec (resp. into ed) if the word on its right side belongs to $\Sigma^*(a+b)$ (resp. to Σ^*b).

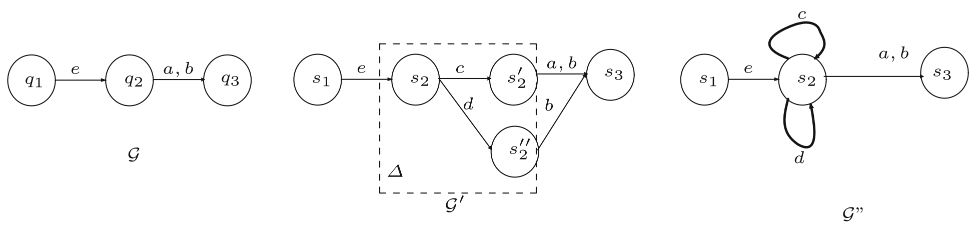

Fig. 2 Importance of bisimilarity of equivalent exit vertices

4.2 Computing transitive closures by widening

Let R be a regular tree relation and L be a regular tree language. We can apply our widening techniques to guess $R^*(L)$ as follows: We compute the sequence $L, R(L), R^2(L), R^3(L)\ldots$,[2] and we compare at each step the hypergraph of the new computed automaton corresponding to $R^i(L)$ to the hypergraphs of the previous automata corresponding to $R^j(L)$, $j < i$, starting from $j = i - 1$ and decrementing j at each step. As soon as we detect a growth between the automata of $R^i(L)$ and $R^k(L)$ for some index k, we extrapolate and apply widening to compute a new automaton A' as described above. By doing so, we are guessing that A' represents the reachability set, or an over-approximation of it. To be sure that our guess is correct, we check whether $L' = R(L') \cup L$, where L' is the language of A'. If this holds, we deduce that L' is an over-approximation of the reachability set (it contains L and is invariant by R), otherwise, we do not extrapolate, and we continue comparing the automata of $R^i(L)$ and $R^j(L)$, for $j < k$ until we reach the index $j = 0$, in which case, we proceed to the next iteration, we compute $R^{i+1}(L)$ and we compare it to its predecessors, etc. We can simplify this procedure by comparing at each step the new computed automaton to its immediate predecessor, i.e., comparing $R^i(L)$ to $R^{i-1}(L)$; if the widening cannot be applied, we go to the next step and compute $R^{i+1}(L)$, etc.

Note that the same procedure can be applied to compute the transitive closure of a regular tree relation.

4.3 Exact widening

We give hereafter a test which allows in some cases to check automatically whether a widening operation computes the *exact* reachability set $R^*(L)$.

Definition 9 A relation R over terms is well founded if there is no infinite sequence of terms $t_0, t_1, \ldots$ such that for every $i \geq 0$, $(t_{i+1}, t_i) \in R$.

Proposition 2 [36] *If R is well founded then $L' = R^*(L)$ iff*

$$L' = R(L') \cup L \tag{11}$$

[2] This sequence can be obtained by composing at each step with the transducer $\mathcal{T}$ of R.

Therefore, when R is well founded, we can use our widening technique to generate automatically transitive closure candidates, and use the test (11) to check automatically that a candidate is indeed equal to $R^*(L)$.

5 Parametrized networks with tree-like topologies

In this section, we show the application of regular tree model checking in the analysis of parameterized networks of identical processes arranged in a tree-like topology.

We model such systems by relabeling tree transducers. Indeed, the set of configurations of a parameterized tree network can be represented by a set of trees (of arbitrary size) where node labels correspond to control locations of processes, and therefore, actions in the network can be seen as transformations which modify the labels in the trees.

Then, given a set of initial configurations represented by a finite tree automaton $\mathcal{A}$ and a finite tree transducer $\mathcal{T}$ representing the dynamics in the network, we can apply reachability analysis with regular widening in order to compute (an upper-approximation of) the set of reachable configurations $R^*_{\mathcal{T}}(\mathcal{L}(\mathcal{A}))$. We can also apply the same procedure in order to compute a finite transducer corresponding to the transitive closure of $R_{\mathcal{T}}$.

5.1 Example: parallel OR algorithm

To illustrate our approach, we show the example of a parallel boolean program, called PERCOLATE [41], which computes the OR of a set of boolean values: we consider an arbitrary number of processes arranged in a binary tree architecture. Each process has a variable *val* ranging over $\{0, 1, \perp\}$. Initially, all the leaves have $val \in \{0, 1\}$, and all the other nodes have $val = \perp$. The purpose of the program is to percolate to the root the value 1 if at least one of the leaves has $val = 1$. A transition of the system consists in assigning 1 to a node if one of its children has $val = 1$, and 0 otherwise. This corresponds to the term rewriting system $R_{\text{percolate}}$ given by the following rewriting rules:

$$\perp(1(x_1, x_2), 1(x_3, x_4)) \to 1(1(x_1, x_2), 1(x_3, x_4))$$
$$\perp(1(x_1, x_2), 0(x_3, x_4)) \to 1(1(x_1, x_2), 0(x_3, x_4))$$
$$\perp(0(x_1, x_2), 1(x_3, x_4)) \to 1(0(x_1, x_2), 1(x_3, x_4))$$
$$\perp(0(x_1, x_2), 0(x_3, x_4)) \to 0(0(x_1, x_2), 0(x_3, x_4))$$

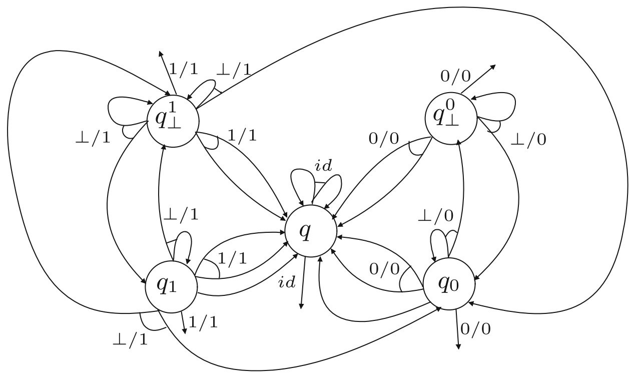

Fig. 3 The transducer of $R^*_{\text{percolate}}$

The property to check is that the root is labeled by 1 if and only if at least one of the leaves is labeled by 1. This property can be represented by a regular tree automaton. Hence, we can check the satisfaction of this property if we are able to compute the set of reachable configurations in the system.

Actually, our approach allows to construct automatically the transitive closure of $R_{\text{percolate}}$ after one iteration. This relation is given by the transducer depicted in Fig. 3 where the final state is q and id stands for $\{\perp/\perp, 0/0, 1/1\}$. Since the order between the children is not important, the automaton has for each rule of the form $f/g(q_1, q_2) \to q$ a symmetrical rule $f/g(q_2, q_1) \to q$, which is not represented in the figure for the sake of clarity. Details about the construction of this transducer are described in Sects. 5.4 and 5.5.

5.2 Well-oriented systems

We prove hereafter that with our widening techniques reachability analysis terminates and computes exactly the transitive closure of (at least) a kind of relabeling transducers, called *well-oriented systems*, which correspond to a significant class of parameterized networks.

It can be observed that many protocols and parallel algorithms which are defined on networks with a tree-like topology satisfy the following features: (1) information goes from leaves upward to the root and vice versa, which means that each node communicates directly either with its children or with its father, (2) there is a finite number of alternating phases of upward and downward information propagation (e.g., requests are sent by leaves, and then answers are sent by the root, and so on), (3) the state of each process is modified after each transmission of information, i.e., at each phase, when a node of the network is crossed, it is marked by a new label. This corresponds, for instance, to marking paths, memorizing sent messages, etc.

We introduce a model to describe the dynamics of such parameterized tree networks which consists of term rewriting systems called *well-oriented systems*. To simplify the presentation, we shall restrict ourselves in this section to binary trees, the general case is similar.

Definition 10 Let $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_n$, where the $\mathcal{S}_i$'s are disjoint finite sets of symbols.

A n-phase *well-oriented system* (n-phase WOS) over $\mathcal{S}$ is a set of rewriting rules of the form:

$$b(a(x_1, x_2), c_1(x_3, x_4)) \to a(b'(x_1, x_2), c_1(x_3, x_4)) \tag{12}$$
$$a(b(x_1, x_2), c_1(x_3, x_4)) \to b'(a(x_1, x_2), c_1(x_3, x_4)) \tag{13}$$
$$a(b(x_1, x_2), c_2(x_3, x_4)) \to b'(a(x_1, x_2), a(x_3, x_4)) \tag{14}$$
$$b(x_1, x_2) \to d(x_1, x_2) \tag{15}$$
$$b(a(x_1, x_2), c_1(x_3, x_4)) \to d(a(x_1, x_2), c_1(x_3, x_4)) \tag{16}$$
$$a(b(x_1, x_2), c_1(x_3, x_4)) \to a(d(x_1, x_2), c_1(x_3, x_4)) \tag{17}$$
$$a(b(x_1, x_2), c_2(x_3, x_4)) \to a(d(x_1, x_2), d(x_3, x_4)) \tag{18}$$

as well as the symmetrical forms of these rules obtained by commuting the children, where $a, b', c_1 \in \mathcal{S}_{i+1}$, $b, c_2 \in \mathcal{S}_i$, and $d \in \mathcal{S}_{i+2}$, such that $0 \leq i \leq n-1$ for the rules (12), (13), and (14), and $0 \leq i \leq n-2$ for the last rules.

In the definition above, the variables x_1, x_2, x_3 and x_4 represent the subtrees hanging under the nodes a and c_1 in the rules (12) and (16), b and c_1 in the rules (13) and (17), and b and c_2 in the rules (14) and (18).

Intuitively, a rule (12) corresponds to the upward propagation of a. When a crosses b, it takes its place and labels its old place with b' in order to mark its path. Similarly, a rule (13) corresponds to the downward propagation of a, and a rule (14) corresponds to the broadcasting of a. Finally, the four last rules allow to pass from one phase to the next one. More precisely, the rules (15) correspond to an unconditional move, and the rules (16) (resp. (17) and (18)) to a conditional move towards a descending (resp. an ascending) phase. These rules correspond to the move from phase $i+1$ to phase $i+2$, i.e., from the propagation of symbols in $\mathcal{S}_{i+1}$ to the propagation of symbols in $\mathcal{S}_{i+2}$.

The rewriting system above induces a relabeling relation $R_{\mathcal{S}}$ over trees defined by: $(t, t') \in R_{\mathcal{S}}$ if there exists a context C, and:

– either a rule $b(x_1, x_2) \to d(x_1, x_2)$ in $\mathcal{S}$, and two terms u_1 and u_2 such that

$$t = C[b(u_1, u_2)] \quad \text{and} \quad t' = C[d(u_1, u_2)],$$

– or a rule $a(b(x_1, x_2), c(x_3, x_4)) \to d(e(x_1, x_2), f(x_3, x_4))$ in $\mathcal{S}$, and four terms u_1, u_2, u_3 and u_4 such that

$$t = C[a(b(u_1, u_2), c(u_3, u_4))] \quad \text{and}$$
$$t' = C[d(e(u_1, u_2), f(u_3, u_4))].$$

In the remainder of this paper, we will not distinguish between a WOS system $\mathcal{S}$ and the relabeling relation $R_{\mathcal{S}}$ it induces.

5.3 Examples of WOSs

Several examples can be modeled using well-oriented systems. For instance, the system $R_{\text{percolate}}$ given above is a 1-phase WOS where $\mathcal{S}_0 = \{\perp\}$, and $\mathcal{S}_1 = \{0, 1\}$. We can also model as a WOS the *Parity Tree* protocol presented in [27] which computes the parity of the number of leaves labeled by 1. A more interesting example is the *asynchronous tree arbiter* mutual exclusion protocol described in [1]: leaf processes can ask for a critical resource and their requests go all the way up in the tree until the root is reached. Then, the permission to access the critical resource propagates downward the tree to at most one of the leaf processes. This process enters then the critical section. The dynamics of this protocol can easily be modeled by a two-phase well-oriented system. Then, the mutual exclusion property to check is that there is at most one leaf at the critical section (this can be represented by a regular tree automaton).

5.4 Analyzing well-oriented systems

In order to prove that regular widening allows to construct transitive closures of WOSs, we give first a direct construction of these transitive closures, and show that regular widening can simulate this construction. This section is dedicated to the proof of this theorem:

Theorem 1 *Let R be a well-oriented system, then R^* is regular and effectively representable by a tree transducer.*

Let R be a n-phase well-oriented system. The set of rules of the form (12) will be called $R^{\uparrow}_{i+1}$ (they correspond to the upward propagation of the letters a of $\mathcal{S}_{i+1}$). The rules of the form (13) and (14) will be called $R^{\downarrow}_{i+1}$ (they correspond to the downward propagation of the letters a of $\mathcal{S}_{i+1}$). We let $R_i = R^{\uparrow}_i \cup R^{\downarrow}_i$ for every $1 \leq i \leq n$. The set R_i corresponds to the phase i of the system since its rules propagate the letters of $\mathcal{S}_i$. Finally, the four last rules allow to pass from one phase to the next one. This set of rules ((15), (16), (17), and (18)) will be called $R_{i+1\to i+2}$ (they correspond to the move from the phase $i+1$ to the phase $i+2$, i.e., from the propagation of the symbols of $\mathcal{S}_{i+1}$ to the propagation of the symbols of $\mathcal{S}_{i+2}$).

Observe that the application of the previous rules always increases the index of the label of any node (to which the rule has been applied) in the tree. This property together with the fact that c_1 is in $\mathcal{S}_{i+1}$ ensure that the phase $i+1$ ($a \in \mathcal{S}_{i+1}$) depends only on the earlier phases $j \leq i+1$. Therefore, it is easy to see that $R^* = R^*_n \circ R^*_{n-1\to n} \circ R^*_{n-1} \circ \cdots \circ R^*_{1\to 2} \circ R^*_1$. Hence, it suffices to compute the closures $R^*_{i\to i+1}$ for i s.t. $1 \leq i < n$, and R^*_i for every i s.t. $1 \leq i \leq n$. We give hereafter transducers corresponding to these relations.

Computing R^*_i

Let t and t' be two terms such that $t' \in R^*_i(t)$. The terms t and t' can be seen as two different labelings of the same underlying tree u. Consider the term t/t' an other labeling of the tree u defined as follows: A node n is labeled with f/g if n is labeled with f (resp. g) in t (resp. in t'). Let $T_{\mathcal{S}/\mathcal{S}}$ be the set of such terms t/t'. If $t'' \in R_i(t')$ and $t' \in R_i(t)$, we write $t/t'' \in R_i(t/t')$. The relation R^*_i can be seen as the set of such terms $\{t/t' \mid t' \in R^*_i(t)\}$.

R^*_i is given by the transducer $\mathcal{A}_i = (Q, \mathcal{S}, \mathcal{S}, F, \delta)$ where $Q = \{q\} \cup \{q_c \mid c \in \mathcal{S}_i\} \cup \{q^a_{b'}, q^b_a \mid a, b' \in \mathcal{S}_i, b \in \mathcal{S}_{i-1}\}$, $F = \{q\}$, and δ is the smallest set of transition rules that contains the following transition rules:

(α_1) $a/a \to q$ and $a/a(q, q) \to q$, for every $a \in \mathcal{S}$,
(α_2) $c/c \to q_c$ and $c/c(q, q) \to q_c$, for every $c \in \mathcal{S}_i$,
(α_3) $q_a \to q$, for every $a \in \mathcal{S}_i$,

and such that:

– if $R^{\uparrow}_i$ contains the rule $b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4))$, where $a, b', c \in \mathcal{S}_i$, and $b \in \mathcal{S}_{i-1}$, then the transition relation δ contains:

  (α_4) $a/b'(q, q) \to q^a_{b'}$, (a is rewritten into b'),
  (α_5) $a/b' \to q^a_{b'}$, (a is a leaf, and is rewritten into b'),
  (α_6) $b/a(q^a_{b'}, q_c) \to q_a$, ($a$ takes the place of the letter b),
  (α_7) $b/b'(q^a_{b'}, q_c) \to q^a_{b'}$ (a crosses successively the same letter b several times in the same path),
  (α_8) $b/d'(q^a_{b'}, q_c) \to q^a_{d'}$, if $R^{\uparrow}_i$ contains the rule $d(a(x_1, x_2), e(x_3, x_4)) \to a(d'(x_1, x_2), e(x_3, x_4))$ (a crosses first the letter b and immediately after, it crosses the letter d),
  (α_9) $a/f'(q^f_{b'}, q_g) \to q^a_{b'}$, if $R^{\downarrow}_i$ contains the rule $b'(f(x_1, x_2), g(x_3, x_4)) \to f'(b'(x_1, x_2), g(x_3, x_4))$ (a is first rewritten into b' and then into f': the letter b' goes down by crossing f),
  (α_{10}) $a/f'(q^f_{b'}, q^g_{b'}) \to q^a_{b'}$, if $R^{\downarrow}_i$ contains the rule $b'(f(x_1, x_2), g(x_3, x_4)) \to f'(b'(x_1, x_2), b'(x_3, x_4))$.

– if $R^{\downarrow}_i$ contains the rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), c(x_3, x_4))$, where $a, b', c \in \mathcal{S}_i$, and $b \in \mathcal{S}_{i-1}$, then the transition relation δ contains:

  (α_{11}) $b/a \to q^b_a$, (a is a leaf and takes the place of the letter b),
  (α_{12}) $b/a(q, q) \to q^b_a$, (a takes the place of the letter b),
  (α_{13}) $a/b'(q^b_a, q_c) \to q_a$, $a/b'(q^b_a, q_c) \to q_{b'}$, ($a$ is rewritten into b'. This node is annotated either by

q_a or by $q_{b'}$ to memorize that we had an "a" or a "b'" at this position. This is useful if a or b' is a context of some other rule in R_i. This can happen since these letters belong to $\mathcal{S}_i$),

(α_{14}) $b/b'(q_a^b, q_c) \to q_a^b$, ($a$ crosses successively the same letter b several times in the same path),

(α_{15}) $b/d'(q_a^d, q_e) \to q_a^b$, if $R_i^{\downarrow}$ contains the rule $a(d(x_1, x_2), e(x_3, x_4)) \to d'(a(x_1, x_2), e(x_3, x_4))$ (a crosses first the letter b and immediately after, it crosses the letter d),

(α_{16}) $b/f'(q_a^f, q_a^g) \to q_a^b$, if $R_i^{\downarrow}$ contains the rule $a(f(x_1, x_2), g(x_3, x_4)) \to f'(a(x_1, x_2), a(x_3, x_4))$ (a crosses first the letter b and immediately after, it is broadcasted through f and g),

(α_{17}) $a/h'(q_a^b, q_c) \to q_{h'}^{b'}$, if $R_i^{\uparrow}$ contains the rule $h(b'(x_1, x_2), k(x_3, x_4)) \to b'(h'(x_1, x_2), k(x_3, x_4))$ (a is first rewritten into b' and then into h').

– if $R_i^{\downarrow}$ contains the rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), a(x_3, x_4))$, where $a, b' \in \mathcal{S}_i$, and $b, c \in \mathcal{S}_{i-1}$, then the transition relation δ contains:

(α_{18}) $b/a \to q_a^b$, (a is a leaf and takes the place of the letter b),

(α_{19}) $b/a(q, q) \to q_a^b$, (a takes the place of the letter b),

(α_{20}) $c/a \to q_a^c$, (a is a leaf and takes the place of the letter c),

(α_{21}) $c/a(q, q) \to q_a^c$, (a takes the place of the letter c),

(α_{22}) $a/b'(q_a^b, q_a^c) \to q_a$, $a/b'(q_a^b, q_a^c) \to q_{b'}$, ($a$ is rewritten into b'),

(α_{23}) $b/b'(q_a^b, q_a^c) \to q_a^b$, ($a$ crosses successively the same letter b several times in the same path),

(α_{24}) $c/b'(q_a^b, q_a^c) \to q_a^c$, ($c$ is first rewritten into a and then into b')

(α_{25}) $b/d'(q_a^d, q_e) \to q_a^b$, $c/d'(q_a^d, q_e) \to q_a^c$, if $R_i^{\downarrow}$ contains the rule $a(d(x_1, x_2), e(x_3, x_4)) \to d'(a(x_1, x_2), e(x_3, x_4))$ (a crosses first the letter b and immediately after, it crosses the letter d),

(α_{26}) $b/f'(q_a^f, q_a^g) \to q_a^b$, and $c/f'(q_a^f, q_a^g) \to q_a^c$, if $R_i^{\downarrow}$ contains the rule $a(f(x_1, x_2), g(x_3, x_4)) \to f'(a(x_1, x_2), a(x_3, x_4))$,

(α_{27}) $a/h'(q_a^b, q_a^c) \to q_{h'}^{b'}$,if $R_i^{\uparrow}$ contains the rule $h(b'(x_1, x_2), k(x_3, x_4)) \to b'(h'(x_1, x_2), k(x_3, x_4))$ (a is first rewritten into b' and then into h').

Since the order between the children is not important, the automaton has for each rule of the form $f/g(q_1, q_2) \to q$ a symmetrical rule $f/g(q_2, q_1) \to q$ which is not represented above.

Intuition First, let us explain intuitively how does this automaton behave. Let t and t' be two terms such that $t' \in R_i^*(t)$. Any arbitrary node of t/t' which is labeled by f/f, i.e., where no rewriting has occurred, can be annotated by q (rules α_1). A node n in t/t' is annotated by q_a, $a \in \mathcal{S}_i$ if:

– n is labeled by a/a (rules α_2). This means that no rewriting has been done in this place i.e., the label of this node is the same in both t and t'.
– there exists $b \in \mathcal{S}_i$ such that n is labeled by b/a (rules α_6). This means that a rewriting has been done at this node using a rule $b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4))$. The node n is labeled by b in t' and by a in t. The state q_c in the left hand side of the transition rule α_6 imposes that b can cross a only if its brother is c.
– there exists $b' \in \mathcal{S}_i$ such that n is labeled by a/b'. This means that a rewriting has been done at this node using a rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), c(x_3, x_4))$ (rules α_{13}), or a rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), a(x_3, x_4))$ (rules α_{22}). This node can also be annotated by $q_{b'}$ (rules α_{13} and α_{22}).

In all these cases, a node n is annotated by q_a ($a \in \mathcal{S}_i$) if it is labeled by a either in t or in t', i.e., if at this position there "was" or there "is" an "a". We need to annotate such nodes by q_a to know that there is (was) an "a" at this position. This is useful if a is in the context of some rule in R_i (if it is the brother of a node where a rewriting can occur). Note that rules α_6 annotate nodes labeled with b/a only with q_a and not with q_b. This is because $b \in \mathcal{S}_{i-1}$, and thus it cannot be the context of another rule in R_i, therefore, it is useless to memorize that at this position we had a "b".

Let a be a symbol of $\mathcal{S}_i$, and let $\{b_k \in \mathcal{S}_{i-1}\}$ be a set of marking symbols such that there is a rule of the form $b_k(a(x_1, x_2), c(x_3, x_4)) \to a(b'_k(x_1, x_2), c(x_3, x_4))$ in R_i, where c is in $\mathcal{S}_i$. A node in t/t' is annotated by $q_{b'_k}^a$ if its label is a/b'_k, b_k/b'_k, or b_l/b'_k and all its ancestors are annotated by $q_{b'_k}^a$ (we apply the same rule, i.e., a crosses the same letter b_k) until we reach an ancestor annotated by q_a, i.e., labeled by b_k/a (we stop the application of the rule and put a (rules α_6)) or until we reach an ancestor annotated by $q_{b'_{k'}}^a$, for some $k' \neq k$, i.e., labeled by $b_k/b'_{k'}$ (this means that in this node, the label b_k in t was first rewritten into the letter a which was then rewritten using another rule into $b'_{k'}$).

We show that:

Lemma 1 $R_i^* = \mathcal{L}(\mathcal{A}_i)$.

Proof We show that for every $b \in \mathcal{S}_{i-1}$, and every $a, b' \in \mathcal{S}_i$, we have:

(A) $L_q = R_i^*$,
(B) $L_{q_a} = \{u \in R_i^* \mid root(u) \in \{a/a, a/b', b/a \mid b \in \mathcal{S}_{i-1}, b' \in \mathcal{S}_i\}\}$,

(C) $L_{q^a_{b'}} = \{u \in T_{\mathcal{S}/\mathcal{S}}$ s.t. there exists a rule $b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4)) \in R_i^{\uparrow} \mid b/a(u, c/c) \in R_i^*\}$,

(D) $L_{q^b_a} = \{u \in T_{\mathcal{S}/\mathcal{S}}$ s.t. there exists a rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), c(x_3, x_4)) \in R_i^{\downarrow}$ and $a/b'(u, c/c) \in |, R_i^*$, or there exists a rule $a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), a(x_3, x_4)) \in R_i^{\downarrow}$, and $a/b'(u, c/a) \in R_i^*\}$.

The proof of these items can be found in the appendix. □

Computing $R^*_{i\to i+1}$

$R^*_{i\to i+1}$ is given by the transducer $\mathcal{A}_{i\to i+1} = (Q, \mathcal{S}, \mathcal{S}, F, \delta)$ where $Q = \{q\} \cup \{q_a \mid a \in \mathcal{S}_i\} \cup \{q_{b/d} \mid b \in \mathcal{S}_{i-1}, d \in \mathcal{S}_{i+1}\}$, $F = \{q\}$, and δ is the following set of transition rules:

(γ_1) $a/a \to q$ and $a/a(q, q) \to q$, for every $a \in \mathcal{S}$,
(γ_2) $a/a \to q_a$ and $a/a(q, q) \to q_a$, for every $a \in \mathcal{S}_i$,
(γ_3) $q_a \to q$, for every $a \in \mathcal{S}_i$,
(γ_4) $b/d \to q$ and $b/d(q, q) \to q$, for every rule $b \to d$ of $R_{i\to i+1}$,
(γ_5) $b/d(q_a, q_c) \to q$, for every rule $b(a(x_1, x_2), c(x_3, x_4)) \to d(a(x_1, x_2), c(x_3, x_4))$ of $R_{i\to i+1}$,
(γ_6) $b/d(q, q) \to q_{b/d}$, $b/d \to q_{b/d}$,
(γ_7) $a/a(q_{b/d}, q_c) \to q_a$ for every rule $a(b(x_1, x_2), c(x_3, x_4)) \to a(d(x_1, x_2), c(x_3, x_4))$ of $R_{i\to i+1}$,
(γ_8) $a/a(q_{b/d}, q_{c/d}) \to q_a$ for every rule $a(b(x_1, x_2), c(x_3, x_4)) \to a(d(x_1, x_2), d(x_3, x_4))$ of $R_{i\to i+1}$.

Here also, the automaton above has for each rule of the form $f/g(q_1, q_2) \to q$ a symmetrical rule $f/g(q_2, q_1) \to q$ that is not represented. Intuitively, a node is annotated by q_a if it is labeled with a/a. As previously mentioned, these states q_a are useful to know whether the context of some rule holds at some point. Since these contexts only belong to $\mathcal{S}_i$, it suffices to record the places of the symbols $a \in \mathcal{S}_i$. A node is annotated by $q_{b/d}$ if it is labeled with b/d after the application of a rule of the form (17) or (18) corresponding to a move to a descending step. A node annotated by this state must have a father labeled with a/a (transition rules γ_7 and γ_8).

We show that:

Lemma 2 $R^*_{i\to i+1} = \mathcal{L}(\mathcal{A}_{i\to i+1})$.

Proof We show as previously that:

- $L_q = R^*_{i\to i+1}$,
- $L_{q_a} = \{u \in R^*_{i\to i+1} \mid root(u) = a/a\}$,
- $L_{q_{b/d}} = \{u = b/d(u_1, u_2) \mid u_1 \in R^*_{i\to i+1}, u_2 \in R^*_{i\to i+1}\}$.

□

As an example, an equivalent transducer to the one of Fig. 3 is produced by our construction.

5.5 Analyzing well-oriented systems with widening

We show in this section that regular tree widening is able to compute the transitive closure of any well-oriented system (it can emulate the constructions given above).

Theorem 2 *Let R be a well-oriented system, then a tree transducer that represents R^* can be computed using regular tree widening.*

Proof We show that widening allows to compute the relations R_i^* for every $1 \le i \le n$ and $R^*_{i\to i+1}$ for every $1 \le i < n$. We give hereafter the details of the computation of R_i^*, the other construction is similar. Since R_i is well founded, it suffices to show that during the iterative computations of the relations R_i^j, there exists a situation where widening can be applied and can produce R_i^*. The bad widening candidates are rejected by the test (11). To simplify the presentation, we suppose that $R_i = R_i^{\uparrow}$. The case where $R_i^{\downarrow}$ is not empty is similar.

R_i can be described by the following relabeling transducer: $\mathcal{A} = (Q, \mathcal{S}, \mathcal{S}, F, \delta)$ where $Q = \{q, q'\} \cup \{q_c \mid c \in \mathcal{S}_i\} \cup \{q^a_{b'} \mid a, b' \in \mathcal{S}_i\}$, $F = \{q'\}$, and δ is the set of following rules, where every rule of the form $f/g(q_1, q_2) \to q_3$ corresponds to a symmetrical rule of the form $f/g(q_2, q_1) \to q_3$ that is not represented:

(γ_1) $a/a \to q$ and $a/a(q, q) \to q$, for every $a \in \mathcal{S}$,
(γ_2) $c/c \to q_c$, $c/c(q, q) \to q_c$, and $q_c \to q$ for every $c \in \mathcal{S}_i$,

and such that if $R_i^{\uparrow}$ contains the rule $b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4))$, where $a, b', c \in \mathcal{S}_i$, and $b \in \mathcal{S}_{i-1}$, then the transition relation δ contains:

(γ_3) $a/b'(q, q) \to q^a_{b'}$,
(γ_4) $a/b' \to q^a_{b'}$,
(γ_5) $b/a(q^a_{b'}, q_c) \to q'$,
(γ_6) $a/a(q', q) \to q'$, for every $a \in \mathcal{S}$,

The state q recognizes the terms of the form t/t, where no rewriting occurred. The states q_a recognize the terms of the form t/t where $root(t) = a$, $a \in \mathcal{S}_i$. The state q' recognizes the terms of the form t/t' such that $t' \in R_i(t)$. The state $q^a_{b'}$ is an intermediate state that allows to memorize the place where a has been rewritten into b'.

If we compose the relabeling above with itself, we obtain the following relabeling $\mathcal{A}^2$ (that recognizes R_i^2): $\mathcal{A}^2 = (Q', \mathcal{S}, \mathcal{S}, F', \delta')$ where $Q' = Q \times Q$, $F' = F \times F$, and δ' is the set of transition rules defined as follows. If $R_i^{\uparrow}$ contains the rules $b(a(x_1, x_2), c(x_3, x_4)) \rightarrow a(b'(x_1, x_2), c(x_3, x_4))$ and $d(a(x_1, x_2), e(x_3, x_4)) \rightarrow a(d'(x_1, x_2), e(x_3, x_4))$, where $a, b', c, d', e \in \mathcal{S}_i$, and $b, d \in \mathcal{S}_{i-1}$, then δ' contains:

(β_1) $f/f \rightarrow [q_0, q_1]$ and $f/f\big([q, q], [q, q]\big) \rightarrow [q_0, q_1]$ for every $f \in \mathcal{S}$, $q_0, q_1 \in \{q, q_f\}$;
(β_2) $a/b'\big([q, q], [q, q]\big) \rightarrow [q^a_{b'}, q]$, $a/b' \rightarrow [q^a_{b'}, q]$;
(β_3) $a/b'\big([q, q], [q, q]\big) \rightarrow [q, q^a_{b'}]$, $a/b' \rightarrow [q, q^a_{b'}]$;
(β_4) $b/a\big([q^a_{b'}, q], [q_c, q]\big) \rightarrow [q', q]$,
(β_5) $b/a\big([q^a_{b'}, q], [q_c, q]\big) \rightarrow [q', q_a]$,
(β_6) $b/a\big([q, q^a_{b'}], [q, q_c]\big) \rightarrow [q, q']$,
(β_7) $b/a\big([q, q^a_{b'}], [q', q_c]\big) \rightarrow [q', q']$,
(β_8) $b/d'\big([q^a_{b'}, q], [q_c, q]\big) \rightarrow [q', q^a_{d'}]$,
(β_9) $b/a\big([q', q^a_{b'}], [q, q_c]\big) \rightarrow [q', q']$,
(β_{10}) $f/f\big([q', q], [q, q']\big) \rightarrow [q', q']$, $f/f\big([q, q], [q', q']\big) \rightarrow [q', q']$, $f/f\big([q, q], [q', q]\big) \rightarrow [q', q]$, $f/f\big([q, q], [q, q']\big) \rightarrow [q, q']$, for every $f \in \mathcal{S}$,
(β_{11}) $[q_1, q_a] \rightarrow [q_1, q]$ and $[q_a, q_1] \rightarrow [q, q_1]$, where $q_1 \in \{q, q_f, q'\}$.

If we compare the hypergraphs $\mathcal{G}$ (hypergraph of $\mathcal{A}$) and $\mathcal{G}'$ (hypergraph of $\mathcal{A}^2$), we can detect a widening candidate where Δ is defined by the rules (β_4), (β_5), (β_6), (β_8), (β_{11}) and (β_{10}); $\mathcal{I}_\Delta = \{[q', q_a], [q_a, q'], [q', q], [q, q'], [q', q'], [q', q^a_{b'}]\}$; $\mathcal{O}_\Delta = \{[q, q_a], [q_a, q], [q, q], [q^a_{b'}, q], [q, q^a_{b'}]\}$; and φ is defined by: $[q_a] = \{[q', q_a], [q_a, q'], [q, q_a], [q_a, q]\}$ for every $a \in \mathcal{S}_i$, $[q^a_{b'}] = \{[q', q^a_{b'}], [q^a_{b'}, q], [q, q^a_{b'}]\}$, and $[q] = \{[q, q], [q', q], [q, q'], [q', q']\}$. By collapsing the states of Δ according to the partition φ, we obtain a relabeling equivalent to the one of R_i^* described in the previous section. Intuitively, the states $[q]$, $[q_a]$, and $[q^a_{b'}]$ correspond respectively to the states q, q_a, et $q^a_{b'}$ of the construction given in the previous section.

Thus, after collapsing these states, the rules (β_1) get transformed into rules (α_1) and (α_2) of the previous construction; the rules (β_2) and (β_3) give the rules (α_4) and (α_5); rules (β_5) give the rules (α_6); the rules (β_8) give the rules (α_7) and (α_8); the rules (β_{10}) produce the rules (α_1); and the rules (β_{11}) produce the rules (α_3). □

6 Analysis of multithreaded programs

We show in this section the application of regular tree model checking and our widening techniques to the analysis of multithreaded programs modeled as term rewriting systems. We consider here multithreaded programs with recursive calls, dynamic creation of parallel processes, and communication. These programs can be modeled by *process rewrite systems* [21,22,43,47].

6.1 Process rewrite systems

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:

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

Intuitively, "0" is the null process, "·" (resp. "||") denotes sequential composition (resp. parallel composition). There is a structural equivalence between process terms defined by the associativity and commutativity of the parallel composition "||", associativity of the sequential composition "·", and the neutrality of "0".

Definition 11 ([43]) A PRS is a finite set of rules R of the form $t_1 \rightarrow t_2$, where $t_1, t_2 \in T_p$. A *PA system* is a PRS where all the rules have the form $X \rightarrow t$.

A PRS induces a transition relation $\rightarrow_R$ over T_p defined by:

$$\frac{t_1 \rightarrow t_2 \in R}{t_1 \rightarrow_R t_2};\ \frac{t_1 \rightarrow_R t_1'}{t_1||t_2 \rightarrow_R t_1'||t_2};\ \frac{t_1 \rightarrow_R t_1'}{t_1.t_2 \rightarrow_R t_1'.t_2};$$
$$\frac{t_2 \rightarrow_R t_2'}{t_1||t_2 \rightarrow_R t_1||t_2'};\ \frac{t_2 \rightarrow_R t_2'}{t_1.t_2 \rightarrow_R t_1.t_2'}(t_1 \sim_0 0)$$

where $t \sim_0 0$ means that t is a terminated process, whose leaves are all labeled by 0. Let $\xrightarrow{*}_R$ be the reflexive transitive closure of $\rightarrow_R$. Let t and t' be two terms in T_p, we write $t' \in R(t)$ (resp. $t' \in R^*(t)$) iff $t \rightarrow_R t'$ (resp. $t \xrightarrow{*}_R t'$). These definitions are extended to sets of terms in the obvious manner.

Given a PRS R, we denote by $Sub(R)$ the set of subterms of the left hand sides and the right hand sides of the rules of R. We say that a PRS is in *normal form* if every element t in $Sub(R)$ has a size smaller or equal to two (i.e., t is of the form X, $X||Y$, or $X \cdot Y$, where X and Y are process variables). We can show that every PRS can be transformed into an equivalent one in normal form [21]. Therefore, we can assume w.l.o.g. that we are dealing with a PRS in normal form.

It is well known that PRS subsumes pushdown systems, PA systems and Petri nets [43]. In [33,35], it is shown that many data flow analysis problems can be reduced to reachability analysis of pushdown and PA systems. The same approach can be applied to reduce the analysis of more complex programs to reachability analysis of PRSs. To model a program in this framework, process variables are used to represent control points in the program, rules of the form $X \rightarrow X_1 \cdot X_2$ are used to represent sequential recursive calls, whereas rules of the form $X \rightarrow X_1||X_2$ are used to model dynamic creation of parallel processes. Moreover, communication between sequential processes and synchronization between parallel processes can be modeled using rules of the forms $X_1 \cdot X_2 \rightarrow X$ and $X_1||X_2 \rightarrow X$.

As in [21,35,42], we apply regular tree model checking to analyse PRSs. Indeed, PRS terms can be naturally represented as trees: the set T_p can be seen as T_Σ where $\Sigma_0 = \{0\} \cup Var$ and $\Sigma_2 = \{\cdot, ||\}$. Thus, we can use finite tree automata to represent regular sets of PRS configurations. Following [21,35,42], we do not take into account the structural equivalence between terms defined by the properties of neutrality of 0 w.r.t. "·" and "||", associativity and commutativity of "||", and associativity of "·". Indeed, introducing this equivalence makes the set of reachable configurations nonregular [37]. Moreover, since terms represent program control structures, it may be legitimate to ignore structural equivalence since for instance informations about the hierarchy between procedures are lost when reasoning modulo associativity.

We apply regular tree model checking to perform reachability analysis of PRSs. We use iterative computation of reachable configurations enhanced with regular tree widening steps. We show in this section that our widening techniques allow to perform the reachability analysis of PRS systems.

6.2 Example: a concurrent server

The JAVA code below corresponds to a typical concurrent server that launches a new thread to deal with 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(runnableService(socket));
       t.start();
   }
}
```

Let us model this program by a PRS system. An instance of the procedure server() is represented by the process variable X, the instruction try is represented by the variable Y, and an instance of t.start() is represented by the variable Z. The variables T and F correspond to the booleans true and false meaning that the try instruction (represented by Y) succeeded or failed, respectively. The program is modeled by the following PRS system R:

- $R_1 = X \to Y.X$ (the procedure starts by executing Y),
- $R_2 = Y \to T$ (Y returns true),
- $R_3 = Y \to F$ (Y returns false),
- $R_4 = T.X \to X||Z$ (if Y returns true, then a new thread is launched),
- $R_5 = F \to 0$ (otherwise, the request is ignored after failure).

Let us show on this example how widening allows to compute the reachability set $R^*(X)$. We compute the following sequence of sets of terms:

$$\{X\} \overset{R_1}{\to} \{Y.X\} \overset{R_2+R_3}{\to} \{T.X, F.X\} \overset{R_4+R_5}{\to} \{X||Z, 0.X\}$$
$$\overset{R_1}{\to} \{(Y.X)||Z, 0.Y.X\}$$
$$\overset{R_2+R_3}{\to} \{(F.X)||Z, (T.X)||Z, 0.T.X, 0.F.X\} \to \cdots$$

These sets are recognized by the tree automata which are represented by the hypergraphs of Fig. 4. After these iterations, we can detect a widening situation by comparing the hypergraphs $\mathcal{G}$ and $\mathcal{G}'$, where Δ is the subhypergraph inside the dashed rectangle, $\mathcal{I}_\Delta = \{q_X^{T^3}\}$, $\mathcal{O}_\Delta = \{q_X^T\}$, and $\varphi = \{\{q_X^T, q_X^{T^3}\}\}$. The result of the widening is given in Fig. 4. It corresponds precisely to the set $R^*(X)$.

6.3 Analyzing PRS by widening

We prove that when applied to PA systems, our widening technique yields the termination of forward reachability analysis and produces the exact sets of all reachable successors. This shows that our approach is at least as general as the existing algorithms for PA processes, and of course it is more general than them since it can also deal with non PA processes (as shown in the previous section).

Actually, we prove a more general result : our technique allows to construct the exact reachability set for each PRS system R such that R is well founded. For instance, it can be seen that the system corresponding to the concurrent server defined in the previous section is well founded. Then, the completeness result concerning PA follows from the fact that we can transform any PA system to an equivalent well-founded PA (w.r.t. reachability).

Theorem 3 *For every PRS system R, and every regular tree language L, $R^*(L)$ is effectively computable using regular tree widening, provided that we are given a test that checks whether some language is equal to $R^*(L)$.*

Proof Let R be a PRS. Let $Q_R = \{q_t, q_t^{\mathrm{nil}}, q_t^T \mid t \in Sub(R)\}$, and let δ_R be the transition relation containing the following rules:

- $X \to q_X$ for every $X \in Sub(R)$,
- $0 \to q_0$ if $0 \in Sub(R)$,
- $||(q_X, q_Y) \to q_{X||Y}$ if $X||Y \in Sub(R)$,
- $\cdot(q_X, q_Y) \to q_{X \cdot Y}$ if $X \cdot Y \in Sub(R)$.

By these rules, for every $t \in Sub(R)$, q_t recognizes the term t. Let L be a regular tree language defined by the

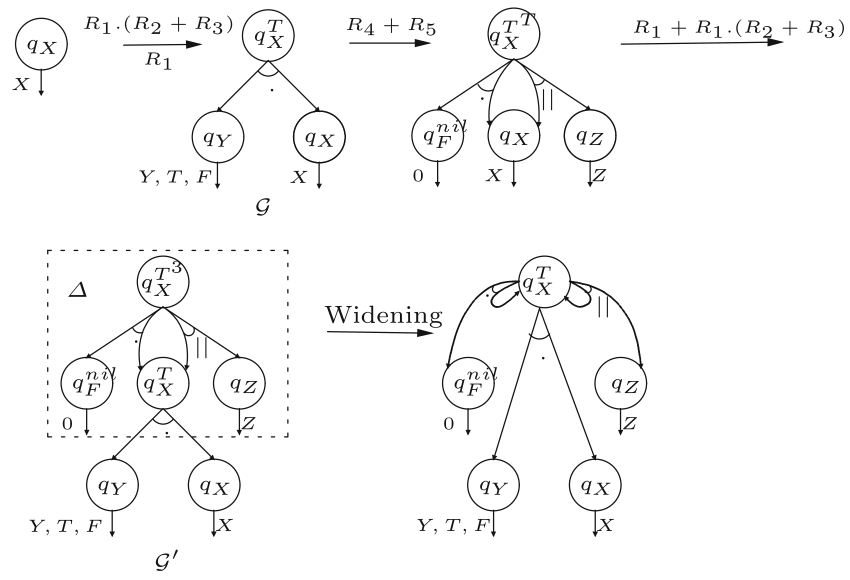

Fig. 4 Reachability analysis of the concurrent server

automaton $\mathcal{A} = (Q, \Sigma, F, \delta)$. It was shown in [21] that $R^*(L)$ can be described by the following automaton: $\mathcal{A}^*_R = (Q', \Sigma, F', \delta')$ such that:

- $Q' = \{q, q^{\text{nil}}, q^T \mid q \in Q \cup Q_R\}$. We denote by $\tilde{q}$ every state in $\{q, q^T, q^{\text{nil}}\}$.
- $F' = \{q, q^{\text{nil}}, q^T \mid q \in F\}$,
- δ' is the smallest set of rules containing $\delta \cup \delta_R$ and such that 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 $0 \to q^{\text{nil}} \in \delta'$,
3. if $t_1 \to t_2 \in R$, and there exists 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) $q^{\text{nil}}_{t_2} \to q^{\text{nil}} \in \delta'$, and
   (c) $q^T_{t_2} \to q^{\text{nil}} \in \delta'$ if $t_1 = 0$,
4. if $\cdot(q_1, q_2) \to q \in \delta \cup \delta_R$, then:
   (a) $\cdot(q_1^{\text{nil}}, \tilde{q}_2) \to q^T \in \delta'$,
   (b) $\cdot(q_1^{\text{nil}}, q_2^{\text{nil}}) \to q^{\text{nil}} \in \delta'$,
   (c) $\cdot(q_1^T, q_2) \to q^T \in \delta'$,
5. if $||(q_1, q_2) \to q \in \delta \cup \delta_R$, then:
   (a) $||(q_1^{\text{nil}}, q_2^{\text{nil}}) \to q^{\text{nil}} \in \delta'$,
   (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 $q^{\text{nil}} \to q'^{\text{nil}} \in \delta'$.

Let us explain the intuition behind this construction. More details can be found in [21,47]. For every $q \in Q \cup Q_R$, the state q^T accepts the successors of L_q (i.e., $L_{q^T} = R^*(L_q)$) and the state q^{nil} accepts the successors u of L_q that have been obtained from null successors of L_q (i.e., $L_{q^{\text{nil}}} = R^*\big(R^*(L_q) \cap \{u \in T_p \mid u \sim_0 0\}\big)$). In particular this means that for every $t \in Sub(R)$, $L_{q_t^T} = R^*(t)$, and $L_{q_t^{\text{nil}}} = R^*\big(R^*(t) \cap \{u \in T_p \mid u \sim_0 0\}\big)$. Rules (1) express that $L_q \subseteq R^*(L_q)$. Rules (2) mark the null leaves with the superscript $^{\text{nil}}$, indicating that they are null. Rules (3) express that if a term t in $Sub(R)$ 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 $u_1 \in R^*(L_{q_1})$, and $u_2 \in R^*(L_{q_2})$, then $u_1||u_2 \in R^*(L_q)$ 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}. Finally, the rules (6) express that if $L_q \subseteq L_{q'}$, then $R^*(L_q) \subseteq R^*(L_{q'})$.

Following the above construction, $R^{\leq j}(L)$ can be described by the following automaton $\mathcal{A}^j = (Q^j, \Sigma, F^j, \delta^j)$ defined as follows:

- $Q^0 = Q \cup Q_R$, $\delta^0 = \delta \cup \delta_R \cup \{0 \to q^{\text{nil}} \mid 0 \xrightarrow{*}_\delta q\}$;
- $Q^j = Q^{j-1} \cup \{q^T \mid q \in Q^{j-1}\} \cup \{q^{\text{nil}} \mid q \in Q^{j-1}\}$. We introduce the notations q^{T^j} defined by: $q^{T^0} = q$ and $q^{T^{j+1}} = (q^{T^j})^T$; and q^{nil^j} defined by: $q^{\text{nil}^1} = q^{\text{nil}}$ and $q^{\text{nil}^{j+1}} = \{(q^{\text{nil}^j})^T, (q^{\text{nil}^j})^{\text{nil}}, (q^{T^j})^{\text{nil}}\}$.
- $F^j = \{q^T \mid q \in F^{j-1}\}$;
- δ^j is the smallest set of rules containing δ^{j-1} and such that for every state $q \in Q^0$ and $q_1, \ldots, q_n \in Q^{j-1}$:

  1. if $t_1 \to t_2 \in R$, and there exists a state $q \in Q \cup Q_R$ such that $t_1 \xrightarrow{*}_{\delta^{j-1}} q^{T^k}$, $k < j$, then if $l + k + 1 = j$ we have:
     (a) $q^{T^l}_{t_2} \to q^{T^j} \in \delta^j$,

 (b) $q_{t_2}^{\mathrm{nil}^l} \to q^{\mathrm{nil}^j} \in \delta^j$, and
 (c) $q_{t_2}^{T^l} \to q^{\mathrm{nil}\, j} \in \delta^j$ if $t_1 = 0$,
2. if $\cdot(q_1, q_2) \to q \in \delta \cup \delta_R$, then :
 (a) $\cdot(q_1^{\mathrm{nil}^k}, q_2^{T^l}) \to q^{T^j} \in \delta^j$, if $k + l = j$;
 (b) $\cdot(q_1^{\mathrm{nil}^k}, q_2^{\mathrm{nil}^l}) \to q^{\mathrm{nil}^j} \in \delta^j$, if $k + l = j$
 (c) $\cdot(q_1^{T^j}, q_2) \to q^{T^j} \in \delta'$;
3. if $||(q_1, q_2) \to q \in \delta \cup \delta_R$, then:
 (a) $||(q_1^{\mathrm{nil}^k}, q_2^{\mathrm{nil}^l}) \to q^{\mathrm{nil}^j} \in \delta^j$, if $k + l = j$;
 (b) $||(q_1^{T^k}, q_2^{T^l}) \to q^{T^j} \in \delta^j$, if $k + l = j$;
4. if $q \to q' \in \delta^{j-1}$, then $q^T \to q'^T \in \delta^j$, and $q^{\mathrm{nil}} \to q'^{\mathrm{nil}} \in \delta^j$.

Then for every $q \in Q \cup Q_R$, $L'_{q^{T^j}} = R^j(L_q)$ and

$$L'_{q^{\mathrm{nil}^j}} = \bigcup_{l+k=j} R^k\big(R^l(L_q) \cap \{u \in T_p \mid u \sim_0 0\}\big)$$

where L_q is the language recognized by the state q in the automaton $\mathcal{A}$ and $L'_{q'}$ is the language recognized by the state q' in the automaton $\mathcal{A}^j$. Intuitively, a state s^k, where s is of the form q^T or q^{nil} is meant to recognize terms that are successors after at most k rewriting steps of terms recognized by the state q.

It is clear that for every rule α of the form $q_t^T \to q^T$, $q_t^{\mathrm{nil}} \to q^{\mathrm{nil}}$, or $q_t^T \to q^{\mathrm{nil}}$ added by the rules (3) of the construction of $\mathcal{A}_R^*$, there is an index j such that δ^j contains rules of the form $q_t^{T^l} \to q^{T^j} \in \delta^j$, $q_t^{\mathrm{nil}^l} \to q^{\mathrm{nil}^j} \in \delta^j$, or $q_t^{T^l} \to q^{T^{j-1}\mathrm{nil}} \in \delta^j$. Let $m(\alpha)$ be the minimum of such indices j. Let $m = \max m(\alpha)$, the maximum being taken over all the rules α of δ' that were added by the inference rules (3) of $\mathcal{A}_R^*$.

We can show that if we compare the hypergraphs $\mathcal{G}$ (hypergraph of $\mathcal{A}^1$) and $\mathcal{G}'$ (hypergraph of $\mathcal{A}^m$), we can detect a widening candidate such that Δ is defined by the rules of $\delta^m \backslash \delta^1$; $\mathcal{I}_\Delta = \{q^{T^i} \mid q \in Q^0, 2 \le i \le m\} \cup \{q^{\mathrm{nil}^i} \mid q \in Q^0, 2 \le i \le m\}$; $\mathcal{O}_\Delta = \{q^T \mid q \in Q^0\} \cup \{q^{\mathrm{nil}} \mid q \in Q^0\}$; and φ is the partition $\varphi = \big\{\{q^{T^i} \mid i > 0\}, \{q^{\mathrm{nil}^i} \mid i > 0\} \mid q \in Q \cup Q_R\big\}$; i.e., for every state q in $Q \cup Q_R$ there are classes $[q^T] = [q^{T^i}]$, and $[q^{\mathrm{nil}}] = [q^{\mathrm{nil}^i}]$ for every $i > 0$. Then, the hypergraph computed by widening corresponds to the automaton $\mathcal{A}_R^*$. □

An immediate consequence of the previous theorem is:

Corollary 1 *For every PRS system R, if R is well founded then for every regular tree language L, the set $R^*(L)$ is effectively computable using regular tree widening.*

Theorem 4 *For every PA system R, and every regular tree language L, the set $R^*(L)$ is effectively computable using regular tree widening.*

Proof Let R be a PA system. We can suppose w.l.o.g. that R is well founded, in which case Corollary 1 infers that $R^*(L)$ is effectively computable using regular tree widening. Indeed, if R is not well founded, we can construct a well-founded PA system which is equivalent to R (w.r.t. reachability) [47]. □

7 Reachability analysis of linear semi-monadic relations

We show in this section that our widening techniques allow to compute $R^*(L)$ for every regular tree language L and every well-founded linear semi-monadic relation R. Linear semi-monadic relations are defined as follows:

Definition 12 (*Linear semi-monadic rewriting systems* [26]) A rewriting system S over an alphabet Σ endowed with an arity function is *linear semi-monadic* if, for every rule $l(x_1, \ldots, x_k) \to r$ in S, the right hand side part r is either a variable, a linear term $f(r_1, \ldots, r_n)$ where $f \in \Sigma_n$, and for every i, $1 \le i \le n$, r_i is a variable in $\{x_1, \ldots, x_k\}$ or a ground term. A rewriting system is ground if all its rules are of the form $l \to r$ where l and r are ground terms.

A linear semi-monadic rewriting system S induces a *linear semi-monadic relation* R_S over the terms of T_Σ defined by the set of pairs of terms (t, t') such that there exists a context C, terms $u_1, \ldots, u_k$ in T_Σ such that $t = C\big[l[x_i \leftarrow u_i]\big]$ and $t' = C\big[r[x_i \leftarrow u_i]\big]$; where $l[x_i \leftarrow u_i]$ (resp. $r[x_i \leftarrow u_i]$) is the term obtained by substituting in l (resp. in r) every occurrence of the variables x_i by the term u_i, for every i, $1 \le i \le k$. In the following, we will not distinguish between a linear semi-monadic rewriting system S and the relation R_S it induces.

It is well known that ground rewrite systems effectively preserve recognizability [19,30]. This result has been extended to linear semi-monadic relations by Coquidé, Dauchet, Gilleron, and Vágvölgyi:

Theorem 5 [26] *Let R be a linear semi-monadic rewriting system over Σ and L be a regular tree language over T_Σ, then $R^*(L)$ is a regular tree language and can be effectively computed.*

Proof We recall the construction of [26]. Let R be a linear semi-monadic rewriting system. Let $Sub(R)$ be the set of all the ground subterms of the right hand sides of the rules of R. Let $Q_R = \{q_t \mid t \in Sub(R)\}$ and δ_R be the following set of transition rules:

- $a \to q_a$ for every $a \in Sub(R) \cap \Sigma_0$,
- $f(q_{t_1}, \ldots, q_{t_n}) \to q_{f(t_1,\ldots,t_n)}$ if $f(t_1, \ldots, t_n) \in Sub(R)$.

It is clear that with δ_R, for every term $t \in Sub(R)$, the state q_t accepts $\{t\}$ ($L_{q_t} = \{t\}$). Let $\mathcal{A} = (Q, \Sigma, F, \delta)$ be a finite tree automaton. We define the automaton $\mathcal{A}^* = (Q', \Sigma, F', \delta')$ as follows:

- $Q' = Q \cup Q_R$.
- $F' = F$,
- δ' is the smallest set of rules containing $\delta \cup \delta_R$ and such that for every states $q, q_1, \ldots, q_n \in Q \cup Q_R$:

  (α_1) if $l(x_1, \ldots, x_k) \rightarrow f(r_1, \ldots, r_{k'}) \in R$ and $l(q_1, \ldots, q_k) \xrightarrow{*}_{\delta'} q$, then $f(q'_1, \ldots, q'_{k'}) \rightarrow q$ is in δ' with $q'_j = q_t$ if $r_j = t \in T_\Sigma$, and $q'_j = q_i$ if $r_j = x_i$ for $j \in \{1, \ldots, k'\}$,

  (α_2) if $l(x_1, \ldots, x_k) \rightarrow x_j \in R$ and $l(q_1, \ldots, q_k) \xrightarrow{*}_{\delta'} q$, then $q_j \rightarrow q$ is in δ'.

Then, $L(\mathcal{A}^*) = R^*(L)$. More precisely, we can show that for every state q, $L'_q = R^*(L_q)$ where L_q is the language recognized by the state q in the automaton $\mathcal{A}$, and L'_q is the language recognized by the state q in the new automaton $\mathcal{A}^*$. □

We show in the following that widening allows to compute this automaton.

Theorem 6 *Let R be a linear semi-monadic rewriting system over Σ and L be a regular tree language over T_Σ. If there exists a test that allows to decide whether a given language L' is equal to $R^*(L)$, then $R^*(L)$ can be computed by widening.*

Proof Since we have a test that allows to decide whether a given language L' is equal to $R^*(L)$, we will show that widening allows to compute the automaton $\mathcal{A}^*$ described in the proof above, we will not be worried about widening candidates that are not exact because they will be rejected by the test. We reconsider the same notations as in the previous proof. It is easy to see that $R^{\leq j}(L)$ is recognized by the automaton $\mathcal{A}^j = (Q^j, \Sigma, F^j, \delta^j)$ defined as follows (these automata are obtained by composing at each step with a transducer $\mathcal{T}$ representing R):

- $Q^0 = Q \cup Q_R$, $\delta^0 = \delta \cup \delta_R$;
- $Q^j = Q^{j-1} \cup \{q^T \mid q \in Q^{j-1}\}$. We introduce the notation q^{T^j} defined by: $q^{T^0} = q$ and $q^{T^{j+1}} = (q^{T^j})^T$;
- $F^j = \{q^T \mid q \in F^{j-1}\}$;
- δ^j is the smallest set of rules containing δ^{j-1} and such that for every states $q \in Q^0$ and $q_1, \ldots, q_n \in Q^{j-1}$:

  1. if $q_1 \rightarrow q_2 \in \delta^{j-1}$ then $q_1^T \rightarrow q_2^T \in \delta^j$;
  2. if $f(q_1, \ldots, q_n) \rightarrow q^{T^{j-1}} \in \delta^{j-1}$ then for every i, $1 \leq i \leq n$,
  $$f(q_1, \ldots, q_{i-1}, q_i^T, q_{i+1}, \ldots, q_n) \rightarrow q^{T^j} \in \delta^j,$$
  3. if $l(x_1, \ldots, x_k) \rightarrow f(r_1, \ldots, r_{k'}) \in R$ and $l(q_1, \ldots, q_k) \xrightarrow{*}_{\delta^{j-1}} q^{T^{j-1}}$, then $f(q'_1, \ldots, q'_{k'}) \rightarrow q^{T^j}$ is in δ^j with $q'_h = q_t$ if $r_h = t \in T_\Sigma$, and $q'_h = q_i$ if $r_h = x_i$ for $h \in \{1, \ldots, k'\}$,
  4. if $l(x_1, \ldots, x_k) \rightarrow x_h \in R$ and $l(q_1, \ldots, q_k) \xrightarrow{*}_{\delta^{j-1}} q^{T^{j-1}}$, then $q_h \rightarrow q^{T^j}$ is in δ^j.

For every $q \in Q \cup Q_R$, $L'_{q^{T^j}} = R^j(L_q)$ where L_q is the language recognized by the state q in the automaton $\mathcal{A}$, and $L'_{q^{T^j}}$ is the language recognized by the state q^{T^j} in the automaton $\mathcal{A}^j$. We explain in the following the intuition behind the rules (1)–(4):

- Rules (1) express that if $R^{j-1}(L_{q_1}) \subseteq R^{j-1}(L_{q_2})$, then $R^j(L_{q_1}) \subseteq R^j(L_{q_2})$;
- Rules (2) express that if $f(t_1, \ldots, t_n) \in R^{j-1}(L_q)$, and $t'_i \in R(t_i)$, then $f(t_1, \ldots, t'_i, \ldots, t_n) \in R^j(L_q)$;
- Rules (3) express that if $l(x_1, \ldots, x_k) \rightarrow f(r_1, \ldots, r_{k'}) \in R$ and $l(t_1, \ldots, t_k) \in R^{j-1}(L_q)$, then $r(u_1, \ldots, u_{k'}) \in R^j(L_q)$ where $u_h = t$ if $r_h = t \in T_\Sigma$, and $u_h = t_i$ if $r_h = x_i$ for $h \in \{1, \ldots, k'\}$;
- Similarly, rules (4) express that if $l(x_1, \ldots, x_k) \rightarrow x_h \in R$ and $l(t_1, \ldots, t_k) \in R^{j-1}(L_q)$, then $t_h \in R^j(L_q)$.

From the constructions of the rules (3) and (α_1) ((α_1) are the rules added in the proof of Theorem 5), for every rule α of the form $f(q_1, \ldots, q_n) \rightarrow q$ added by (α_1), there exists an index j and $i_1, \ldots, i_n$ such that $f(q_1^{T^{i_1}}, \ldots, q_n^{T^{i_n}}) \rightarrow q^{T^j}$ is a rule of δ_j. Let $m(\alpha)$ be the minimum of such indices j.

Similarly, for every rule α of the form $q_i \rightarrow q$ added by (α_2), there exist indices j and $k \leq j$ such that $q_i^{T^k} \rightarrow q^{T^j}$ is a rule of δ_j. Let $m(\alpha)$ be the minimum of such indices j.

Let $m = \max m(\alpha)$ be the maximum taken over all the rules α of δ' that were added by the rules (α_1) and (α_2). It is clear that for every rule of the form $f(q_1, \ldots, q_n) \rightarrow q$ added by (α_1), there exists a rule of the form $f(q_1^{T^{i_1}}, \ldots, q_n^{T^{i_n}}) \rightarrow q^{T^j}$ in δ^m; and for every rule of the form $q_i \rightarrow q$ added by (α_2), there exists a rule of the form $q_i^{T^k} \rightarrow q^{T^j}$ in δ^m.

We can show that if we compare the hypergraphs $\mathcal{G}$ (hypergraph of $\mathcal{A}^0$) and $\mathcal{G}'$ (hypergraph of $\mathcal{A}^m$), we can detect a widening candidate such that Δ is defined by the rules $\delta^m \backslash \delta^0$; $\mathcal{I}_\Delta = \{q^{T^i} \mid q \in Q^0, 1 \leq i \leq m\}$; $\mathcal{O}_\Delta = Q^0$; and φ is defined by $\varphi = \big\{\{q^{T^i} \mid i \geq 0\} \mid q \in Q \cup Q_R\big\}$. If we apply widening according to this partition, we obtain a new hypergraph that is equivalent to the automaton $\mathcal{A}^*$. □

As an immediate consequence of this theorem, we obtain:

Corollary 2 *Let R be a well-founded linear semi-monadic rewriting system over Σ and let L be a regular tree language over T_Σ. Then $R^*(L)$ can be effectively computed by widening.*

Since ground rewriting systems are linear semi-monadic rewriting system, the above corollary implies that widening allows to obtain the result of [19,30] for ground rewriting systems if they are well founded:

Corollary 3 *Let R be a well-founded ground rewriting system over Σ and let L be a regular tree language over T_Σ. Then, $R^*(L)$ can be effectively computed by widening.*

8 Conclusion

We have defined a general framework for reasoning about many kinds of infinite-state systems. Indeed trees are very common data structures and can be used to encode configurations of many classes of systems.

In this paper, we have considered the case of parametrized tree networks and the case of multithreaded programs modeled as transformers of tree control structures. Of course many other cases can be considered since we can consider all systems modeled as term rewriting systems, e.g., systems manipulating abstract data types, logic programs, process calculi, etc. In particular, our algorithmic techniques could be applied in the analysis of cryptographic protocols following the approach in [24,38,39,44] where such systems are represented as term rewriting systems and sets of configurations of such protocols are represented by means of tree automata.

We have defined an acceleration technique (regular tree widening) based on detecting regular growths in sequences of tree sets. Hence, this technique can be applied uniformly regardless from the class of tree transformations since it is based on comparing hypergraph structures of tree automata. In particular, it can be used for structure-preserving as well as for non structure-preserving transformations. We have also shown that this technique is accurate and powerful enough to emulate existing specialized algorithms for symbolic reachability analysis (such as the one for PA and PRS systems). In [46], it has already been shown that regular *word* widening (defined on word automata) can simulate existing constructions such as those in [3,17]. We can actually show that regular widening simulates many other constructions such as, e.g., those in [2,9] concerning pushdown systems and lossy fifo-channel systems.

Finally, the widening principle we have defined here on trees can be extended easily to graphs using graph grammars. This would allow to deal with systems having more complex control or data structures. However, the problem is then to determine a class of graph grammars having nice closure and decision properties, which can be used as symbolic representation structures.

Appendix A: Proof of Lemma 1

Proof We show that for every $b \in \mathcal{S}_{i-1}$, and every $a, b' \in \mathcal{S}_i$, we have:

(A) $L_q = R_i^*$,

(B) $L_{q_a} = \{u \in R_i^* \mid root(u) \in \{a/a, a/b', b/a \mid b \in \mathcal{S}_{i-1}, b' \in \mathcal{S}_i\}\}$,

(C) $L_{q^a_{b'}} = \{u \in T_{\mathcal{S}/\mathcal{S}}$ s.t. there exists a rule $b(a(x_1, x_2), c(x_3, x_4)) \rightarrow a(b'(x_1, x_2), c(x_3, x_4)) \in R_i^\uparrow \mid b/a(u, c/c) \in R_i^*\}$,

(D) $L_{q^b_a} = \{u \in T_{\mathcal{S}/\mathcal{S}}$ s.t. there exists a rule $a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), c(x_3, x_4)) \in R_i^\downarrow$ and $a/b'(u, c/c) \in R_i^*$, or there exists a rule $a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), a(x_3, x_4)) \in R_i^\downarrow$, and $a/b'(u, c/a) \in R_i^*\}$.

First, we introduce the notation $\xrightarrow{k}_\delta$ defined as follows: Given t and t' two terms of $T_{\Sigma \cup Q}$, then $t \xrightarrow{0}_\delta t$, and $t \xrightarrow{k}_\delta t'$ iff there exists t'' such that $t \xrightarrow{k-1}_\delta t''$ and $t'' \rightarrow_\delta t'$.

Let us show the inclusions from left to right first. Let then u be such that $u \xrightarrow{k}_\delta q$, $u \xrightarrow{k}_\delta q_a$, $u \xrightarrow{k}_\delta q^a_{b'}$, or $u \xrightarrow{k}_\delta q^b_a$, respectively. We will show the inclusions by induction on k.

- For $k = 1$:
  - if $u \rightarrow_\delta q$, then u is a leaf labeled with b/b for some b in $\mathcal{S}$ (rules α_1), i.e., $u \in R_i^*$,
  - if $u \rightarrow_\delta q_a$, then u is a leaf labeled with a/a (rules α_2), i.e., $u \in R_i^*$, and $root(u) = a/a$,
  - if $u \rightarrow_\delta q^a_{b'}$, then u is a leaf labeled with a/b' and $b(a(x_1, x_2), c(x_3, x_4)) \rightarrow a(b'(x_1, x_2), c(x_3, x_4))$ is a rule of $R_i^\uparrow$ (rules α_5). Then $b/a(u, c/c) \in R_i^*$,
  - if $u \rightarrow_\delta q^b_a$, then u is a leaf labeled with b/a and $a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), c(x_3, x_4))$ is a rule of $R_i^\downarrow$ (rules α_{11}) (resp. $a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), a(x_3, x_4))$ is a rule of $R_i^\downarrow$ (rules α_{18} and α_{20})). Then $a/b'(u, c/c) \in R_i^*$ (resp. $a/b'(u, c/a) \in R_i^*$).
- Let $k > 1$, and let u be such that:
  - $u \xrightarrow{k}_\delta q$, then necessarily there exists $a \in \mathcal{S}_i$ such that:
    1. $u \xrightarrow{k-1}_\delta q_a \rightarrow_\delta q$. By induction, we have that $u \in L_{q_a} \in R_i^*$, or
    2. $u = a/a(u_1, u_2) \xrightarrow{k-1}_\delta a/a(q, q) \rightarrow_\delta q$ (rules α_1). By induction we have that $u_1, u_2 \in R_i^*$, and hence $u = a/a(u_1, u_2) \in R_i^*$.
  - $u \xrightarrow{k}_\delta q_a$. There are five cases:
    1. $u = a/a(u_1, u_2) \xrightarrow{k-1}_\delta a/a(q, q) \rightarrow_\delta q_a$ (rules α_2). As previously mentioned, it can be seen by induction that $u \in R_i^*$. Moreover, $root(u) = a/a$.
    2. There is a rule $r = b(a(x_1, x_2), c(x_3, x_4)) \rightarrow a(b'(x_1, x_2), c(x_3, x_4))$ in $R_i^\uparrow$, and $u = b/a(u_1, u_2) \xrightarrow{k-1}_\delta b/a(q^a_{b'}, q_c) \rightarrow_\delta q_a$(rules α_6). By induc-

tion, since $u_1 \in L_{q^a_{b'}}$ and $u_2 \in L_{q_c}$, we have that $b/a(u_1, c/c) \in R_i^*, u_2 \in R_i^*$, and the root n of u_2 is labeled with $c/c, c/d'$, or d/c, for $d \in \mathcal{S}_{i-1}$, and $d' \in \mathcal{S}_i$. This implies that $u = b/a(u_1, u_2) \in R_i^*$, since from the fact that $b/a(u_1, c/c) \in R_i^*$ and $u_2 \in R_i^*$, it follows that $b/a(u_1, u_2) \in R_i^*$ if there was a "c" in the node n (which is used as a constraint that allows the application of the rule r). This is ensured by the fact that n is labeled with $c/c, c/d'$, or d/c.

3. There is a rule $r = a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), c(x_3, x_4))$ in $R_i^{\downarrow}$, and $u = a/b'(u_1, u_2) \xrightarrow{k-1}_{\delta} a/b'(q^b_a, q_c) \to_{\delta} q_a$ (rules α_{13}). By induction, since $u_1 \in L_{q^b_a}$, and $u_2 \in L_{q_c}$, we have that $a/b'(u_1, c/c) \in R_i^*, u_2 \in R_i^*$, and the root n of u_2 is labeled with $c/c, c/d'$, or d/c, for $d \in \mathcal{S}_{i-1}$, and $d' \in \mathcal{S}_i$. As previously mentioned, this implies that $u = a/b'(u_1, u_2) \in R_i^*$.
4. The case where there is a rule $r = b(d(x_1, x_2), c(x_3, x_4)) \to d'(b(x_1, x_2), c(x_3, x_4))$ in $R_i^{\downarrow}$ such that $d' = a$, and $u = b/a(u_1, u_2) \xrightarrow{k-1}_{\delta} b/a(q^d_b, q_c) \to_{\delta} q_a$ (rules α_{13}) is similar to the previous ones.
5. The case where the last applied rule is a α_{22} rule is similar to the previous cases.

- $u \xrightarrow{k}_{\delta} q^a_{b'}$. Let $r = b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4))$ in $R_i^{\uparrow}$ such that one of these cases holds:

  1. $u = a/b'(u_1, u_2) \xrightarrow{k-1}_{\delta} a/b'(q, q) \to_{\delta} q^a_{b'}$ (rules α_4). It is clear that $b/a(u, c/c) \in R_i^*$ since by induction $u_1, u_2 \in R_i^*$. Indeed, it suffices to apply r to $b/b\big((a/a(u_1, u_2), c/c\big)$.
  2. $u =|, b/b'(u_1, u_2) \xrightarrow{k-1}_{\delta} b/b'(q^a_{b'}, q_c) \to_{\delta} q^a_{b'}$ (rules α_7). By induction, it follows that $b/a(u_1, c/c) \in R_i^*$, and hence $b/a(u_1, u_2) \in R_i^*$, since $u_2 \in L_{q_c}$. It follows that $b/a(u, c/c) \in r\Big(b/b\big(b/a(u_1, u_2), c/c\big)\Big) \in R_i^*$.
  3. $u = d/b'(u_1, u_2) \xrightarrow{k-1}_{\delta} d/b'(q^a_{d'}, q_e) \to_{\delta} q^a_{b'}$ (rules α_8), and there exists a rule $r' = d(a(x_1, x_2), e(x_3, x_4)) \to a(d'(x_1, x_2), e(x_3, x_4))$ in $R_i^{\uparrow}$. By induction, it follows that $d/a(u_1, e/e) \in R_i^*$, and hence $d/a(u_1, u_2) \in R_i^*$, since $u_2 \in L_{q_e}$. It follows that $b/a(u, c/c) \in r\Big(b/b\big(d/a(u_1, u_2), c/c\big)\Big) \in R_i^*$.
  4. The other cases can be dealt with similarly.

- $u \xrightarrow{k}_{\delta} q^b_a$. Let then $r = a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), c(x_3, x_4))$ in $R_i^{\uparrow}$ such that one of these cases holds (the case where $r = a(b(x_1, x_2), c(x_3, x_4)) \to b'(a(x_1, x_2), a(x_3, x_4))$ is similar):

  1. $u = b/a(u_1, u_2) \xrightarrow{k-1}_{\delta} b/a(q, q) \to_{\delta} q^b_a$ (rules α_{12}). Then, it follows by induction hypothesis that $a/b'(u, c/c) \in R_i^*$,
  2. $u = b/b'(u_1, u_2) \xrightarrow{k-1}_{\delta} b/b'(q^b_a, q_c) \to_{\delta} q^b_a$ (rules α_{14}). Then it follows by induction hypothesis that $a/b'(u_1, u_2) \in R_i^*$, and thus $a/b'(b/b'(u_1, u_2), c/c) \in R_i^*$ since $a/b'(b/a, c/c) \in R_i^*$.
  3. The cases where we apply the other rules α_i are similar.

For the other direction, we show by induction on k that if $u \in R_i^k$ and $root(u) \in \{a/a, a/b', b/a \mid b \in \mathcal{S}_{i-1}, b' \in \mathcal{S}_i\}$, then $u \in L_{q_a}$. This implies thanks to rules α_3 that $u \in L_q$.

- if $k = 0$, then u is a tree labeled by a/a's for letters $a \in \mathcal{S}$. The rules α_1 and α_2 imply that $u \xrightarrow{*}_{\delta} q_a$.
- if $k > 0$. Let then $u' \in R_i^{k-1}$, t_1, t_2, and t_3 such that $u' = t_1/t_2, u = t_1/t_3$, and $t_3 \in R_i(t_2)$. Let then r be a rule of R_i such that $t_3 \in r(t_2)$. There are three cases depending on the nature of the rule r:
  1. $r = b(a(x_1, x_2), c(x_3, x_4)) \to a(b'(x_1, x_2), c(x_3, x_4))$. Let then C_1 and C_2 be two contexts, g_1, g_2, and g_3 be three symbols, and $v_1, v_2, v_3, v_4, v'_1, v'_2, v'_3$, and v'_4 be eight terms such that
  $t_1 = C_1\Big(g_1\big(g_2(v_1, v_2), g_3(v_3, v_4)\big)\Big)$,
  $t_2 = C_2\Big(b\big(a(v'_1, v'_2), c(v'_3, v'_4)\big)\Big)$,
  and
  $t_3 = C_2\Big(a\big(b'(v'_1, v'_2), c(v'_3, v'_4)\big)\Big)$.
  Thus, $g_1 = b$ since $b \in \mathcal{S}_{i-1}$; hence, at this place, no rewriting occurred by the rules of R_i. This implies that $a(v'_1, v'_2) \in R_i^{k-1}\big(g_2(v_1, v_2)\big)$ and that $c(v'_3, v'_4) \in R_i^{k-1}\big(g_3(v_3, v_4)\big)$. Thus, $g_2/a(v_1/v'_1, v_2/v'_2) \in R_i^{k-1}$, and $g_3/c(v_3/v'_3, v_4/v'_4) \in R_i^{k-1}$. Hence, by induction, we have that
  $g_2/a(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_{\delta} q_a \to_{\delta} q$
  and that
  $g_3/c(v_3/v'_3, v_4/v'_4) \xrightarrow{*}_{\delta} q_c \to_{\delta} q$.
  Suppose that the root of t_1 or the root of t_2 is equal to h. Then, by the induction hypothesis, and by applying the rules α_1 to $b/b(q, q)$, it follows that:
  $$\begin{aligned} &C_1/C_2\Big[b/b\big(g_2/a(v_1/v'_1, v_2/v'_2), g_3/c(v_3/v'_3, v_4/v'_4)\big)\Big] \\ &\xrightarrow{*}_{\delta} C_1/C_2[b/b(q, q)] \\ &\to_{\delta} C_1/C_2[q] \xrightarrow{*}_{\delta} q_h. \end{aligned}$$
  Let us denote this derivation by $(\star)$. Note that to annotate $b/b(q, q)$, only α_1 can be applied, α_2 cannot be applied because $b \in \mathcal{S}_{i-1}$. We want to

show that
$u = C_1/C_2\Big[b/a\big(g_2/b'(v_1/v'_1, v_2/v'_2),$

$g_3/c(v_3/v'_3, v_4/v'_4)\big)\Big] \xrightarrow{*}_\delta q_h$.
For this, it suffices to show that

$$b/a\big(g_2/b'(v_1/v'_1, v_2/v'_2), g_3/c(v_3/v'_3, v_4/v'_4)\big) \xrightarrow{*}_\delta$$
$$b/a\big(g_2/b'(v_1/v'_1, v_2/v'_2), q_c\big) \xrightarrow{*}_\delta q_a \rightarrow_\delta q.$$

Thus, by applying the rules α_6, it suffices to show that
$g_2/b'(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_\delta q^a_{b'}$.
This depends on the nature of the letter g_2:

* $g_2 = a$, i.e., there was no rewriting at this place between t_1 and t_2. Thus, v_1/v'_1 and v_2/v'_2 are in R_i^{k-1}. It follows by induction that they can be annotated by q, and thus, by applying the rules α_4 we obtain that:
$(a/b'(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_\delta a/b'(q, q) \rightarrow_\delta q^a_{b'}$
* $g_2 = d$. There are three cases depending on the last rule that has been used to annotate $d/a(v_1/v'_1, v_2/v'_2)$ by q_a during the derivation ($\star$):

(a) Either it is a rule α_6 that has been applied. In this case, there exists necessarily a rule
$r' = d(a(x_1, x_2), e(x_3, x_4))$
$\rightarrow a(d'(x_1, x_2), e(x_3, x_4))$
in $R_i^\uparrow$ such that $v_1/v'_1 \xrightarrow{*}_\delta q^a_{d'}$ and $v_2/v'_2 \xrightarrow{*}_\delta q_e$. The rules α_8 allow to have:$d/b'(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_\delta d/b'(q^a_{d'}, q_e) \rightarrow_\delta q^a_{b'}$

(b) Either it is a rule α_{13} that has been applied. In this case, there exists necessarily a rule
$r'' = d(f(x_1, x_2), e(x_3, x_4))$
$\rightarrow f'(d(x_1, x_2), e(x_3, x_4))$
in $R_i^\downarrow$ such that $f' = a$ and $v_1/v'_1 \xrightarrow{*}_\delta q^f_d$ and $v_2/v'_2 \xrightarrow{*}_\delta q_e$. The rules α_{17} allow to have:$d/b'(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_\delta d/b'(q^f_d, q_e) \rightarrow_\delta q^a_{b'}$.

(c) The case where rules α_{22} have been applied is similar to the previous case.

2. $r = a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), c(x_3, x_4))$. Let then C_1 and C_2 be two contexts, g_1, g_2, and g_3 be three symbols, $v_1, v_2, v_3, v_4, v'_1, v'_2, v'_3$, and v'_4 be eight terms such that
$t_1 = C_1\Big(g_1\big(g_2(v_1, v_2), g_3(v_3, v_4)\big)\Big)$,
$t_2 = C_2\Big(a\big(b(v'_1, v'_2), c(v'_3, v'_4)\big)\Big)$,
and
$t_3 = C_2\Big(b'\big(a(v'_1, v'_2), c(v'_3, v'_4)\big)\Big)$.

It follows that $g_2 = b$ since $b \in \mathcal{S}_{i-1}$, and thus, at this place, no rewriting occurred by the rules of R_i. It follows that $b/b(v_1/v'_1, v_2/v'_2) \xrightarrow{*}_\delta q$ during the annotation of u', since $b \in \mathcal{S}_{i-1}$, thus, the only possible rule to annotate a node labeled by b/b is the rule α_1: $b/b(q, q) \rightarrow q$. Thus, for the same reason,
$g_1/a\big(b/b(v_1/v'_1, v_2/v'_2), g_3/c(v_3/v'_3, v_4/v'_4)\big)$
$\xrightarrow{*}_\delta g_1/a(q, q') \rightarrow_\delta q''$,
where $q' = q$ and q'' is a state of the automaton. It follows that $g_3/c(v_3/v'_3, v_4/v'_4) \in R_i^*$, and more precisely, we have that $g_3/c(v_3/v'_3, v_4/v'_4) \in R_i^{k-1}$, and thus, by induction, we obtain: $g_3/c(v_3/v'_3, v_4/v'_4) \xrightarrow{*}_\delta q_c$.
Let us consider the annotation of u'. We have
$u' \xrightarrow{*}_\delta C_1/C_2[q''] \xrightarrow{*}_\delta q_h$
for some symbol h. Since the label of the root of u is the same as the one of u', we have to show that $u \xrightarrow{*}_\delta q_h$.
By applying the rules α_{12}, we obtain:
$g_1/b'\big(b/a(v_1/v'_1, v_2/v'_2), g_3/c(v_3/v'_3, v_4/v'_4)\big)$
$\xrightarrow{*}_\delta g_1/b'(q^b_a, q_c)$.
We will show that $g_1/b'(q^b_a, q_c) \rightarrow_\delta q''$, in which case, we obtain:
$u \xrightarrow{*}_\delta C_1/C_2[q''] \xrightarrow{*}_\delta q_h$.
There are three cases depending on the nature of g_1:

(a) $g_1 = a$. In this case $q'' = q_a$ or $q'' = q$ ($a/a(q, q) \rightarrow_\delta q_a$ or $a/a(q, q) \rightarrow_\delta q$), and by α_{13} and α_3, we obtain $a/b'(q^b_a, q_c) \rightarrow_\delta q_a \rightarrow_\delta q$.

(b) $g_1 = b$. Then $q'' = q^b_a$ ($b/a(q, q) \rightarrow_\delta q^b_a$), and by α_{14}, we obtain $b/b'(q^b_a, q_c) \rightarrow_\delta q^b_a$.

(c) $g_1 = d$. Then $q'' = q^d_a$ ($d/a(q, q) \rightarrow_\delta q^d_a$), and by α_{15}, we obtain $d/b'(q^b_a, q_c) \rightarrow_\delta q^d_a$.

3. The case where $r = a(b(x_1, x_2), c(x_3, x_4)) \rightarrow b'(a(x_1, x_2), a(x_3, x_4))$ is similar to the previous case. □

References

1. Alur, R., Brayton, R.K., Henzinger, T.A., Qadeer, S., Rajamani, S.K.: Partial-order reduction in symbolic state space exploration. In: Computer Aided Verification, pp. 340–351 (1997)
2. Abdulla, P., Bouajjani, A., Jonsson, B.: On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In: CAV'98. LNCS, vol. 1427 (1998)
3. Abdulla, P.A., Bouajjani, A., Jonsson, B., Nilsson, M.: Handling global conditions in parametrized system verification. Lect. Notes Comput. Sci. **1633**, 134–150 (1999)
4. Abdulla, P.A., Jonsson, B., Mahata, P., d'Orso, J.: Regular tree model checking. In: Proceedings of the 14th International

Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2404, pp. 555–568 (2002)
5. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Proceedings CONCUR 2002 of the 13th International Conference on Concurrency Theory. Lecture Notes in Computer Science, vol. 2421, pp. 116–130 (2002)
6. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Algorithmic improvements in regular model checking. In: Proceedings of the 15th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2725, pp. 236–248 (2003)
7. Abdulla, P.A., Legay, A., d'Orso, J., Rezine, A.: Simulation-based iteration of tree transducers. In: Proceedings of TACAS'05 (2005)
8. Balland, E., Boichut, Y., Genet, T., Moreau, P.-E.: Towards an efficient implementation of tree automata completion. In: AMAST, pp. 67–82 (2008)
9. Bouajjani, A., Esparza, J., Maler, O.: Reachability Analysis of Pushdown Automata: Application to Model Checking. In: CONCUR'97. LNCS, vol. 1243 (1997)
10. Bouajjani, A., Esparza, J., Touili, T.: Reachability Analysis of Synchronised PA systems. In: INFINITY'04. ENTCS (2004)
11. Bouajjani, A., Habermehl, P., Moro, P., Vojnar, T.: Verifying programs with dynamic 1-selector-linked structures in regular model checking. In: Proceedings of TACAS'05 (2005)
12. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular tree model checking. Electr. Notes Theor. Comput. Sci. **149**(1), 37–48 (2006)
13. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: CAV04. Lecture Notes in Computer Science. pp. 372–386. Springer, Boston (2004)
14. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV'00. LNCS (2000)
15. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Proceedings of the 15th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2725, pp. 223–235 (2003)
16. Boigelot, B., Legay, A., Wolper, P.: Omega regular model checking. In: Proceedings TACAS '04 of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science. pp. 561–575 (2004)
17. Bouajjani, A., Muscholl, A., Touili, T.: Permutation Rewriting and Algorithmic Verification. In: LICS'01. IEEE, New York (2001)
18. Bouajjani, A.: Languages, Rewriting systems, and Verification of Infinte-State Systems. In: ICALP'01. LNCS, vol. 2076 (2001) invited paper
19. Brained, W.S.: Tree generating regular systems. Inf. Control **14**, 217–231 (1969)
20. Bouajjani A., Touili, T.: Extrapolating Tree Transformations. In: Proceedings of the 14th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2404, pp. 539–554 (2002)
21. Bouajjani, A., Touili, T.: Reachability analysis of process rewrite systems. In: Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03). LNCS, vol. 2914 (2003)
22. Bouajjani, A., Touili, T.: On Computing Reachability Sets of Process Rewrite Systems. In: RTA'05. LNCS (2005)
23. Cousot, P., Cousot, R.: Static Determination of Dynamic Properties of Recursive Procedures. In: IFIP Conf. on Formal Description of Programming Concepts. North-Holland, Amsterdam (1977)
24. Comon, H., Cortier, V., Mitchell, J.: Tree automata with one memory, set constraints and ping-pong protocols. In: ICALP'2001. LNCS, vol. 2076 (2001)
25. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata (1997)
26. Coquid, J.L., Dauchet, M., Gilleron, R., Vágvlgyi, S.: Bottom-up tree pushdown automata and rewrite systems. Theor. Comput. Sci. **127**(1), 69–98 (1994)
27. Clarke, E.M., Grumberg, O., Jha, S.: Verifying parameterised networks using abstraction and regular languages. Lect. Notes Comput. Sci. 962:395–407 (1995)
28. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL'78. ACM, New York (1978)
29. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: CAV'01. LNCS (2001)
30. Dauchet, M., Tison, S.: The theory of ground rewrite systems is decidable. In: Proceedings of the IEEE Conference on Logic in Computer Science (LICS), IEEE Computer Society Press, New York, pp. 242–248 (1990)
31. Dauchet, M., Tison, S.: The theory of ground rewrite systems is decidable. In: Proceedings of 5th IEEE Symposium on Logic in Computer Science, Philadelphia. pp. 242–248 (1990)
32. d'Orso, J., Touili, T.: Regular hedge model checking. In: 4th IFIP International Conference on Theoretical Computer Science (TCS 2006). IFIP, vol. 209, pp. 213–230. Springer, Berlin (2006)
33. Esparza, J., Knoop, J.: An automata-theoretic approach to interprocedural data-flow analysis. In: FOSSACS'99. LNCS, vol. 1578 (1999)
34. Engelfriet, J.: Bottom-up and top-down tree transformations—a comparison. Math. Syst. Theory, 9(3) (1975)
35. Esparza, J., Podelski, A.: Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In: Symposium on Principles of Programming Languages. pp. 1–11 (2000)
36. Fribourg, L., Olsen, H.: Reachability sets of parametrized rings as regular languages. In: Infinity'97. Electronical Notes in Theoretical Computer Science, vol. **9**, Elsevier, Amsterdam (1997)
37. Gilleron, R., Deruyver, A.: The reachability problem for ground trs and some extensions. In: Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'89). LNCS, vol. 351, pp. 227–243 (1989)
38. Genet, T., Klay, F.: Rewriting for cryptographic protocol verification. In: In Proceedings of the 17th International Conference on Automated Deduction, Pittsburgh (Pen., USA). Lecture Notes in Artificial Intelligence, vol. 1831, Springer, Berlin (2000)
39. Goubault-Larrecq, J.: A method for automatic cryptographic protocol verification. In: 15th IPDPS 2000 Workshops. LNCS, vol. 1800 (2000)
40. Jonsson, B., Nilsson, M.: Transitive closures of regular relations for verifying infinite-state systems. In: TACAS'00. LNCS (2000)
41. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E. : Symbolic model checking with rich assertional languages. In: Grumberg, O. (ed.) Proc. CAV'97, LNCS, vol. 1254, pp. 424–435. Springer, Berlin (1997)
42. Lugiez, D., Schnoebelen, Ph.: The regular viewpoint on PA-processes. In: Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98), Nice, France, September 1998. vol. 1466, pp. 50–66. Springer, Berlin (1998)
43. Mayr, R.: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. Phd. thesis, TUM (1998)
44. Monniaux, D.: Abstracting cryptographic protocols with tree automata. Sci. Comput. Program. (2002)
45. Pnueli, A., Shahar, E.: Liveness and acceleration in parametrized verification. In: CAV'00. LNCS (2000)
46. Touili, T.: Regular Model Checking using Widening Techniques. In: Vepas Workshop. Electronic Notes in TCS, vol. 50 (2001)

47. Touili, T.: 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)
48. Touili, T.: Dealing with communication for dynamic multithreaded recursive programs. In: 1st VISSAS workshop (2005) Invited Paper
49. Touili, T.: Computing transitive closures of hedge transformations. In: VECOS. Electronic Workshops in Computing series (eWiC), British Computer Society, London (2007)
50. Wolper, P., Boigelot, B.: Verifying systems with infinite but regular stae spaces. In: CAV'98. LNCS, vol. 1254 (1998)