Extrapolating Tree Transformations*

Ahmed Bouajjani and Tayssir Touili

Liafa, Univ. of Paris 7, Case 7014, 2 place Jussieu, F-75251 Paris 5, France.
E-mail: {Ahmed.Bouajjani,Tayssir.Touili}@liafa.jussieu.fr

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 parametrized 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.

1 Introduction

Regular Model Checking has been proposed as a general and uniform framework for reasoning about infinite-state systems [KMM+97,WB98,BJNT00,Bou01]. 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

* This work was supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082)

relations having nonregular transitive closures, but under which subclasses of regular languages are effectively closed (see, e.g., [BMT01]). 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.

During the last three years, several authors addressed this issue, essentially in the case of Regular Word Model Checking where configurations are encoded as words (see, e.g., [ABJN99,JN00,BJNT00,PS00,DLS01,Tou01]). In this paper, we consider the more general 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 parametrized networks with tree-like topologies, and data flow analysis of multithreaded programs.

Indeed, in the case of parametrized 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 parametrized 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 sizes 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). Notice that, in contrast with the previous case (parametrized 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 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 parametrized 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 parametrized networks. We consider a particular class of models based on term rewriting systems called *Well-Oriented Systems*. These models correspond to systems where, typically, informations are 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 correspond 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 system.

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 [EK99] which consists in reducing data flow analysis to reachability analysis problems for Process Rewriting Systems (PRS) [May98]. Programs are described as term rewriting rules of the form $t \to t'$ where t and t' are terms built up from process variables, sequential composition, and asynchronous parallel composition.

We give a construction of a tree automaton recognizing the set of immediate successors/predecessors of any regular tree language by a PRS transition, and then, the reachability sets of PRSs can be computed iteratively using regular tree widening. We illustrate our approach on the example of a concurrent server which can launch an unbounded number of threads.

Then, we show that our techniques are at least as general as the known algorithms in this context. Namely, we show that reachability analysis with regular tree widening terminates and computes precisely the sets of forward/backward 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 [LS98,EP00] and can handle programs which are beyond the scope of these algorithms (e.g., the 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 or R^{-1} is Noetherian. 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 which has this property.

For lack of space we omit the details and refer to the full version of this paper [BT02].

Related work: The idea of widening operation is inspired by works in the domain of abstract interpretation [CC77] where it is mainly used for systems with numerical data domains (integer or reals) [CH78]. 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 parametrized systems) is [KMM$^+$97]. However, no acceleration techniques are provided in that work. In [LS98,EP00], tree automata are used for symbolic reachability analysis of PA processes and its applications in model-checking and static analysis of programs. Works on acceleration techniques for regular model checking concern mainly the case of word automata and transducers [ABJN99,JN00,BJNT00,PS00,Tou01]. Our regular tree widening technique is an extension of the widening techniques for word automata defined in [BJNT00,Tou01]. In [DLS01], techniques are presented for computing transitive closures for tree transducers, but no completeness results are provided. The problem of finding subclasses of term rewriting systems (tree transducers) which effectively preserve regularity has also been considered in the automata-theory community by several authors (see, e.g., [GT95] where it is proved that ground tree rewrite systems preserve regularity).

2 Preliminaries

2.1 Trees and terms

An alphabet Σ is ranked if it is endowed with a mapping $rank : \Sigma \to \mathbb{N}$. For $k \geq 0$, Σ_k is the set of elements of rank k. The elements of Σ_0 are called constants. Let $\mathcal{X}$ be a denumerable set of symbols called variables. Let $T_\Sigma[\mathcal{X}]$ denote the set of terms over Σ and $\mathcal{X}$. T_Σ will stand 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.

As usual, 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.

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*

$$f\big(q_1, \ldots, q_n\big) \rightarrow_\delta q \tag{1}$$

$$a \rightarrow_\delta q \tag{2}$$

$$q \rightarrow_\delta q' \tag{3}$$

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, if during the state assignment process the subterms $t_1, \ldots, t_n$ are labeled with states $q_1, \ldots, q_n$, and if the rule $f\big(q_1, \ldots, q_n\big) \rightarrow_\delta q$ is in δ then the term $f(t_1, \ldots, t_n)$ is labeled with q. A term t is accepted if $\mathcal{A}$ reaches the root of t in a final state.

The language accepted by the automaton $\mathcal{A}$ is the set of ground terms that it accepts: $\mathcal{L}(\mathcal{A}) = \bigcup\limits_{q \in F} L_q$.

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:*

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

$$q(x) \rightarrow_\delta q'(u), u \in T_{\Sigma'}[\{x\}] \tag{5}$$

$$a \rightarrow_\delta q(u), u \in T_{\Sigma'} \tag{6}$$

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 $a \rightarrow_\delta q(u)$ is in δ, then a is replaced by $q(u)$. The substitution proceeds then towards the root. If the rule $f\big(q_1(x_1), \ldots, q_n(x_n)\big) \rightarrow_\delta q(u)$ 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 $q \in F\}$. We denote by $R^n_\mathcal{T}$ the composition of $R_\mathcal{T}$, n times. As usual, $R^*_\mathcal{T} = \bigcup_{n \geq 0} R^n_\mathcal{T}$ denotes the reflexive-transitive closure of $R_\mathcal{T}$.

Let $L \subseteq T_\Sigma$ be a tree language. Then, we define the set $\mathcal{R}_\mathcal{T}(L) = \{t' \in \Sigma' \mid \exists t \in L, (t, t') \in R_\mathcal{T}\}$.

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 [Eng75,CDG+97].

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

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

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

$$f\big(q_1(x_1), \dots, q_n(x_n)\big) \to_\delta q\big(g(x_1, \dots, x_n)\big) \tag{7}$$

$$a \to_\delta q(b) \tag{8}$$

$$q(x) \to_\delta q'(x) \tag{9}$$

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

Notice 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 $f/g(q_1, \dots, q_n) \to q$, the rules (8) can be written $a/b \to q$, and the rules (9) can be written $q \to q'$.

2.2 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 $v, v_1, \dots, v_n \in V$, the tuple $(v, f, v_1, \dots, v_n)$ is a* **hyperedge** *labeled by f and connecting in order v to the vertices $v_1, \dots, v_n$. We will write $v \xrightarrow{f} v_1, \dots, v_n$ for every hyperedge $(v, f, v_1, \dots, v_n)$, 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:

- $q \xrightarrow{f}_\delta q_1, \dots, q_n \in H_\delta$ for every rule $f\big(q_1, \dots, q_n\big) \to_\delta q$.
- $q \xrightarrow{a}_\delta \in H_\delta$ for every initial rule $a \to_\delta q$.
- $q \to_\delta q' \in H_\delta$ for every rule $q \to_\delta q'$.

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 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.

3.1 Principle

The technique we present generalizes the one we have introduced in [BJNT00,Tou01] 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 $L = L_1L_2$ and $R(L) = L_1\Delta L_2$ occurs, 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*
  - *$H' = H\backslash H_\Delta \cup \{v'_0 \xrightarrow{f} v'_1, \ldots, v'_n \mid v_0 \xrightarrow{f} v_1, \ldots, v_n \in H\backslash H_\Delta$ 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:*
$H'' = H \cup \{v_0 \xrightarrow{f} v'_1, \ldots, v'_n \mid v_0 \xrightarrow{f} v_1, \ldots, v_n \in \Delta$ *and* $\forall i$, *if* $v_i \in \mathcal{O}_\Delta$ *then* $v'_i \in [v_i]_\varphi \cap \mathcal{I}_\Delta$, *otherwise* $v'_i = v_i\}$.
Intuitively, $\mathcal{G}[\Delta \leftarrow \Delta^+]$ *is obtained by adding loops allowing to iterate* Δ *(by going back to entry vertices).*

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 sub-hypergraph* 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*

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

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

Notice 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 Figure 1.

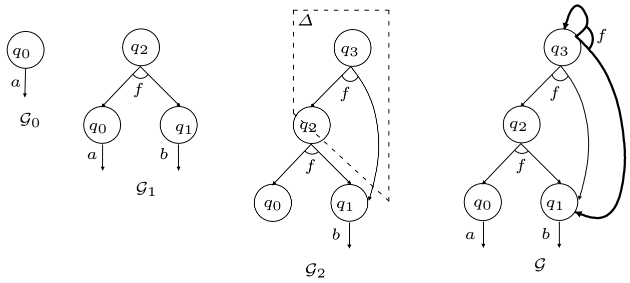

Fig. 1. Illustration of the regular tree widening mechanism

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 $(\mathcal{G}, \{q_3\})$ obtained by adding the loop drawn by thick lines to $\mathcal{G}_2$. This automaton defines precisely $R^{\geq 3}(a)$ (its union with the automata of the previous steps corresponds to $R^*(a)$).

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

Proposition 2. *The problem of finding widening situations is NP-complete.*

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. Efficient (but uncomplete) strategies can be adopted in order to reduce nondeterminism (the number of candidates Δ).

3.2 Exact widening

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

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

Proposition 3. *If R or R^{-1} is nœtherian then $L' = R^*(L)$ iff*

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

Proof: In [FO97], a proof for the case where R^{-1} is nœtherian is given. The proof for the other case can be found in the full paper. □

Therefore, when R or R^{-1} is nœtherian, we can use our widening technique to generate automatically closure candidates, and use the test (11) to check automatically that a candidate is indeed equal to $R^*(L)$.

4 Parametrized Networks with Tree-like Topologies

We show the application of regular tree model checking in the analysis of parametrized 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 parametrized tree network can be represented by a set of trees (of arbitrary size) where nodes correspond to control location 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 $\mathcal{T}^*(\mathcal{A})$. We can also apply the same procedure in order to compute a finite transducer corresponding to the transitive closure of $\mathcal{T}$.

4.1 Example : Parallel OR algorithm

To illustrate our approach, we show the example of a parallel boolean program, called PERCOLATE [KMM+97], 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 others 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 $R_{percolate}$ given by the following rewriting rules:

$$\begin{aligned}
\perp(1(x_1, x_2), 1(x_3, x_4)) &\rightarrow 1(1(x_1, x_2), 1(x_3, x_4))\\
\perp(1(x_1, x_2), 0(x_3, x_4)) &\rightarrow 1(1(x_1, x_2), 0(x_3, x_4))\\
\perp(0(x_1, x_2), 1(x_3, x_4)) &\rightarrow 1(0(x_1, x_2), 1(x_3, x_4))\\
\perp(0(x_1, x_2), 0(x_3, x_4)) &\rightarrow 0(0(x_1, x_2), 0(x_3, x_4))
\end{aligned}$$

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 $R_{percolate}$ after two iterations (see Theorems 1 and 2).

4.2 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 parametrized 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) informations go 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 son 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 parametrized 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. We denote by $\mathcal{S}_{\leq i}$ the set $\bigcup\{\mathcal{S}_j \mid j \leq i\}$.*

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

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

as well as the symmetrical forms of these rules obtained by commuting the children, where $a, b' \in \mathcal{S}_{i+1}$, $b \in \mathcal{S}_i$, $d \in \mathcal{S}_{i+2}$, $c_1 \in \mathcal{S}_{\leq i+1}$, and $c_2 \in \mathcal{S}_{\leq i}$, 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), and b and c_2 in the other rules.

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 rule (15) corresponds to a nonconditionnal passage, and the rules (16) (resp. (17) and (18)) to a conditionnal passage towards a descending (resp. an ascending) phase.

Several examples can be modeled using well-oriented systems. For instance, the system $R_{percolate}$ given above is a 1-phase WOS where $\mathcal{S}_0 = \{\perp\}$, and $\mathcal{S}_1 = \{0, 1\}$. Other examples such as the *Parity Tree* [CGJ95] and the *asynchronous tree arbiter* mutual exclusion protocol [ABH$^+$97] can be found in the full version of the paper [BT02].

4.3 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.

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

Proof (Sketch): Let R be a n-phase well-oriented system. Let us denote by $R^{\uparrow}_{i+1}$ (resp. $R^{\downarrow}_{i+1}$) the set of rules of the form (12) (resp. the set of rules of the form (13) and (14)) corresponding to the upward (resp. downward) propagation of the letters a of $\mathcal{S}_{i+1}$.

We let $R_i = R_i^{\uparrow} \cup R_i^{\downarrow}$ 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 rules (15), (16), (17), and (18) are called $R_{i+1 \rightarrow i+2}$ (they correspond to the passage 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 $\mathcal{S}_{i+2}$).

The main observation is that the application of the previous rules always increases the index of the label of any node in the tree. This property together with the fact that c_1 and c_2 are in $\mathcal{S}_{\leq i+1}$ 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 $R^* = R_n^* \circ R_{n-1 \rightarrow n}^* \circ R_{n-1}^* \circ \cdots \circ R_{1 \rightarrow 2}^* \circ R_1^*$. Moreover, the fact that $c_2 \in \mathcal{S}_{\leq i}$ ensures that during the phase $i+1$, there is no interaction between the ascending rules $R_{i+1}^{\uparrow}$ and the descending ones $R_{i+1}^{\downarrow}$. This infers that $R_i^* = (R_i^{\uparrow})^* \circ (R_i^{\downarrow})^*$. Then, the proof consists in giving direct constructions of the transducers $(R_i^{\uparrow})^*$, $(R_i^{\downarrow})^*$, and $R_{i \rightarrow i+1}^*$ for every $1 \leq i \leq n$ (see [BT02] for details).

□

We show also that regular tree widening is able to compute the transitive closure of any well-oriented system (it can emulate the construction 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.*

5 Multithreaded Programs as Process Rewrite Systems

We show in this section the application of regular tree model checking in 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 are modeled by *Process Rewrite Systems* [May98].

5.1 Process Rewrite Systems

Let $Act = \{a, b, c, \ldots\}$ be a set of actions, $Var = \{X, Y, \ldots\}$ 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, "$\cdot$" (resp. "$||$") denotes sequential composition (resp. parallel composition).

Definition 11 ([May98]). *A* Process Rewriting System *(PRS for short) is a finite set of rules R of the form $t_1 \xrightarrow{a} t_2$, where $t_1, t_2 \in T_p$ and $a \in Act$. A* PA declaration *is a PRS where all the rules have the form $X \xrightarrow{a} t$.*

A PRS induces a transition relation $\xrightarrow{a}_R$ over T_p defined by:

$$\frac{t_1 \xrightarrow{a} t_2 \in R}{t_1 \xrightarrow{a}_R t_2};\ \frac{t_1 \xrightarrow{a}_R t_1'}{t_1||t_2 \xrightarrow{a}_R t_1'||t_2};\ \frac{t_1 \xrightarrow{a}_R t_1'}{t_1.t_2 \xrightarrow{a}_R t_1'.t_2};\ \frac{t_2 \xrightarrow{a}_R t_2'}{t_1||t_2 \xrightarrow{a}_R t_1||t_2'};\ \frac{t_2 \xrightarrow{a}_R t_2'}{t_1.t_2 \xrightarrow{a}_R t_1.t_2'}(t_1 \approx 0)$$

where $t \approx 0$ means that t is a terminated process.

5.2 Example: a concurrent server

The JAVA code below corresponds to a typical concurrent server who 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 rules:

- $R_1 = X \to Y.X$ (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).

5.3 Reachability analysis of PRSs

PRS terms can be naturally represented as trees. Indeed, 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. Therefore, we can apply regular tree model checking to perform reachability analysis of PRSs. We use iterative computation of reachable configurations enhanced with regular tree widening steps.

As in [LS98,EP00], 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 [LS98]. 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 prove that when applied to PA systems, our widening technique yields the termination of forward and backward reachability analysis and produces the exact sets of all reachable successors and predecessors. Moreover, we prove that our technique is applicable beyond the PA case (e.g., for the server above). Furthermore, we prove that our technique allows to construct the exact reachability set for each PRS system R such that R or R^{-1} is Noetherian. For instance, it can be seen that the system corresponding to the concurrent server defined in Section 5.2 is such that R^{-1} is Noetherian. Then, the completeness result concerning PA follows from the fact that we can transform any PA system to an equivalent one having this property.

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 $R^*(L)$.*

An immediate consequence of this theorem is:

Corollary 1. *For every PRS system R, if R or R^{-1} is nœtherian then for every regular tree language L, the sets $R^*(L)$ and $(R^{-1})^*(L)$ are effectively computable using regular tree widening.*

Theorem 4. *For every PA system R, and every regular tree language L, the sets $R^*(L)$ and $(R^{-1})^*(L)$ are effectively computable using regular tree widening.*

Let us mention that Theorem 3 holds also for the class of *ground term rewrite* (GTR) systems which is known to preserve regularity [GT95]. Actually, PRS systems are sets of *ground* term rewriting rules (contrary to, e.g., WOSs used to model parametrized systems in Section 4.2). However, semantically, PRSs are not standard GTR systems due to the semantics of the operator "·" which imposes a particular rewriting strategy on the trees. To establish our results for this class, we proceed as for PRS: we provide a new direct construction of the reachability sets and we show that the widening technique allow to simulate this construction. The direct construction we provide constitutes an alternative and actually simpler proof of the result in [GT95].

6 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 [Mon02,GL00,CCM01] 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 systems). In [Tou01], it has already been shown that regular *word* widening (defined on word automata) can simulate existing constructions such as those in [ABJN99,BMT01]. We can actually show that regular widening simulates many other constructions such as, e.g., those in [BEM97,ABJ98] 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.

References

[ABH+97] Rajeev Alur, Robert K. Brayton, Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification*, pages 340–351, 1997.

[ABJ98] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *CAV'98*. LNCS 1427, 1998.

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

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

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV'00*. LNCS, 2000.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation Rewriting and Algorithmic Verification. In *LICS'01*. IEEE, 2001.

[Bou01] A. Bouajjani. Languages, Rewriting systems, and Verification of Infinte-State Systems. In *ICALP'01*. LNCS 2076, 2001. invited paper.

[BT02] A. Bouajjani and T. Touili. Extrapolating tree transformations. Technical report, LIAFA, May 2002. http://verif.liafa.jussieu.fr/∼touili.

[CC77] P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Recursive Procedures. In *IFIP Conf. on Formal Description of Programming Concepts*. North-Holland Pub., 1977.

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *ICALP'2001*. LNCS 2076, 2001.

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

[CGJ95] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterised networks using abstraction and regular languages. *Lecture Notes in Computer Science*, 962:395–407, 1995.

[CH78] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78*. ACM, 1978.

[DLS01] Dennis Dams, Yassine Lakhnech, and Martin Steffen". Iterating transducers. In *CAV'01*. LNCS, 2001.

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

[Eng75] Joost Engelfriet. Bottom-up and top-down tree transformations – a comparison. In *Mathematical Systems Theory*, volume 9(3), 1975.

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

[FO97] L. Fribourg and H. Olsen. Reachability sets of parametrized rings as regular languages. In *Infinity'97*. volume 9 of Electronical Notes in Theoretical Computer Science. Elsevier Science, 1997.

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *15th IPDPS 2000 Workshops*. LNCS 1800, 2000.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. In *Fundamenta Informaticae*, volume 24, pages 157–175, 1995.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00*. LNCS, 2000.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proc. CAV'97*, volume 1254 of *LNCS*, pages 424–435. Springer, 1997.

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

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

[Mon02] D. Monniaux. Abstracting cryptographic protocols with tree automata. *Science of Computer Programming*, 2002.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parametrized verification. In *CAV'00*. LNCS, 2000.

[Tou01] T. Touili. Widening Techniques for Regular Model Checking. In *Vepas Workshop*. Volume 50 of Electronic Notes in TCS, 2001.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular stae spaces. In *CAV'98*. LNCS 1254, 1998.