Regular Hedge Model Checking

Julien d'Orso[1] and Tayssir Touili[2]

[1] University of Illinois at Chicago. dorso@liafa.jussieu.fr
[2] Liafa, CNRS & Univ. of Paris 7. touili@liafa.jussieu.fr

Abstract. We extend the regular model checking framework so that it can handle systems with arbitrary width tree-like structures. Configurations of a system are represented by trees of arbitrary arities, sets of configurations are represented by regular hedge automata, and the dynamics of a system is modeled by a regular hedge transducer. We consider the problem of computing the transitive closure $\mathcal{T}^+$ of a regular hedge transducer $\mathcal{T}$. This construction is not possible in general. Therefore, we present a *general acceleration* technique for computing $\mathcal{T}^+$. Our method consists of enhancing the termination of the iterative computation of the different compositions $\mathcal{T}^i$ by merging the states of the hedge transducers according to an *appropriate* equivalence relation that preserves the traces of the transducers. We provide a methodology for *effectively* deriving equivalence relations that are appropriate. We have successfully applied our technique to compute transitive closures for some mutual exclusion protocols defined on arbitrary width tree topologies, as well as for an XML application.

1 Introduction

Regular Model Checking has been proposed as a general and uniform framework to analyse infinite-state systems [21, 28, 12, 7]. In this framework, configurations are represented by words or trees, sets of configurations by regular finite word/tree automata, and the transitions of the system by a regular relation described by a word/tree transducer. A central problem in regular model checking is to compute the transitive closure of a regular relation given by a finite-state transducer. Such a representation allows to compute the set of reachable configurations of a system (thus enabling verification of safety properties) as well as to detect loops between configurations if the transformations are structure preserving (thus enabling verification of liveness properties) [12, 6]. However, computing the transitive closure of a transducer is not possible in general since the transition relation of any Turing machine can be represented by a regular word transducer. In fact, the major problem in regular model checking is that a naive computation that consists in iteratively computing the different compositions $\mathcal{T}^i$ of a transducer $\mathcal{T}$ does not terminate in general. Therefore, a main issue in regular model checking is to define *general acceleration* techniques that will force the above iterative procedure to terminate for many practical applications.

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 parametrized systems (i.e., parametrized systems where the processes are arranged in a linear topology) [12, 23, 19, 13, 25, 3, 4], as well as systems that operate on linear unbounded data structures such as lists, integers, reals, and even hybrid automata [5, 11, 6], and programs with pointers [9]. (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 parametrized systems with tree topologies [17, 19, 2, 1], and multithreaded programs [22, 17, 14, 8, 26].

In this paper, we develop the regular model checking paradigm further, and consider the more general case of regular *hedge* model checking, where configurations are represented by trees of arbitrary arities. Indeed, arbitrary width tree-like structures are very common and appear naturally in many modeling and verification contexts. We can mention at least three examples of such contexts:

- XML documents can be modeled by unranked trees whose nodes are labeled with the tags of the document [29, 24]. For example, a document having n pages, where page i has k_i paragraphs can be represented by a tree whose root has n children, and where the i^{th} child has k_i children. Since the number of pages and paragraphs in a document are arbitrary, unranked trees are necessary to represent such documents. Then, transformations on XML documents such as XSLT can be represented by relations on unranked trees.
- Configurations of multithreaded recursive programs can also be represented by unbounded width trees where the leaves are labeled with the control points of the program and the inner nodes with the sequential and the parallel operators $\cdot$ and $\|$. For example, a term $\|(t_1, \ldots, t_n)$ represents a configuration where the terms $t_1, \ldots, t_n$ are in parallel. Since the number of parallel processes can be arbitrarily large, we need unbounded width trees to accurately represent such configurations. Then, actions of the program such as procedure calls, launching of new threads, synchronisation statements, etc, can also be represented by relations on unranked trees [15, 16].
- Many parametrized protocols are defined on tree topologies with unbounded width. Indeed, in the case of tree networks, the number of processes and the topology of the network (including the arities of the different nodes) are not fixed. In this case, labeled trees of arbitrary width and height are needed to represent configurations of tree 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 relations (transformations which preserve the structure of the trees). Examples of such systems are multicast protocols, leader election protocols, mutual exclusion protocols, etc.

We use *hedge automata* [18] to symbolically represent infinite sets of unranked trees, and *hedge transducers* to model transformations on these trees. Then, as in the case of regular *word* and *tree* model checking, the central problem is to compute the transitive closure of a hedge transducer $\mathcal{T}$. Our aim is then 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 this work is the definition of a *general acceleration* technique on relabeling hedge transducers (tranducers that preserve the structure of the trees). Our technique works as follows: To enhance the termination of the iterative computation of the different compositions $\mathcal{T}^i$, we merge equivalent states using an *appropriate* equivalence relation, i.e., an equivalence relation that preserves the traces of the transducers (for which collapsing two states does not add new traces to the transducers). The main problem amounts then to defining and computing appropriate equivalences. We provide a methodology for deriving such equivalence relations. More precisely, we consider equivalence relations induced by two simulation relations, namely a *downward* and an *upward* simulation, both defined on hedge automata. We give sufficent conditions on the simulations that guarantee appropriateness of the induced equivalence. Furthermore, we define *effectively computable downward* and *upward* simulations for which the induced relation is guaranteed to be appropriate. We have successfully applied our technique to compute transitive closures of some mutual exclusion protocols defined on arbitrary width tree topologies. We were also able to handle an XML application. This effort is reported in Section 6.

Related work. There are several works on efficient computation of transitive closures for *word* transducers [12, 19, 25, 5, 11, 6, 4] and *tree* transducers [17, 2, 1]. However, these works only consider trees where the arities are fixed, whereas our framework allows to consider ranked *as well as* unranked trees. In fact, our technique can be seen as an extension of the approach used in [1] to hedge transducers. Note that arbitrary arities make this extension nontrivial. In particular, the transition rules of the collapsed hedge transducer under construction make use of regular languages over classes of tuples, these classes themselves being potentially regular languages. This nesting of languages is delicate to manipulate.

More recently, *hedge automata* have been used to compute reachability sets of some classes of transformations, namely *Process Rewrite Systems* (PRS) [15] and *Dynamic Pushdown Networks* (DPN) [16]. Compared to our work, these algorithms compute the sets of the reachable states of the systems, whereas we consider the more general problem of computing the transitive closure of the system's transducer. Moreover, our technique is *general* and can be *uniformly* applied to all the classes of relabeling transformations, whereas the algorithms of [15, 16] can only be applied to the specific class of PRS or DPN.

Outline. In Section 2, we give the definitions of hedge automata and transducers, and show how the i^{th} iterations for a relabeling hedge transducer can be

effectively computed. In Section 3, we describe our general semi-algorithm. In Section 4, we define relations $\sim$ induced by *downward* and *upward* simulations, and give sufficient conditions ensuring that $\sim$ is an appropriate equivalence relation. We provide in Section 5 an effectively computable example of such an equivalence. Finally, in Section 6, we show some examples on which we applied our technique.

2 Hedge automata and transducers

2.1 Terms

Let Σ be an unranked alphabet and $\mathcal{X}$ be a fixed denumerable set of variables $\{x_1, x_2, \ldots\}$. The set $T_\Sigma[\mathcal{X}]$ of terms over $\Sigma \cup \mathcal{X}$ is the smallest set such that:

- $\Sigma \cup \mathcal{X} \subseteq T_\Sigma[\mathcal{X}]$,
- if $f \in \Sigma$, $t_1, \ldots, t_n \in T_\Sigma[\mathcal{X}]$ for some $n \geq 1$, then $f(t_1, \ldots, t_n) \in T_\Sigma[\mathcal{X}]$.

Terms without variables are called *ground terms*. Let T_Σ be the set of ground terms over Σ. A term t in $T_\Sigma[\mathcal{X}]$ is *linear* if each variable occurs at most once in t. A *context* C is a linear term of $T_\Sigma[\mathcal{X}]$. Let $t_1, \ldots, t_n$ be terms of T_Σ, then $C[t_1, \ldots, t_n]$ denotes the term obtained by replacing in the context C the occurrence of the variable x_i by the term t_i, for each $1 \leq i \leq n$.

As usual, a term in $T_\Sigma[\mathcal{X}]$ can be viewed as a rooted labeled tree u where the leaves are labeled with variables or elements in Σ, and every internal node N with a symbol $\lambda(N) \in \Sigma$, where λ is the labeling associated to u.

2.2 Hedge automata

To finitely represent infinite sets of terms, we use *hedge automata* [18]:

Definition 1. *A* **Hedge automaton** *is a tuple $\mathcal{A} = (Q, \Sigma, F, \delta)$ where Q is a finite set of states, Σ is an unranked alphabet, $F \subseteq Q$ is a set of final states, and δ is a set of rules of the form $f(L) \to q$, where $f \in \Sigma$, $q \in Q$, and $L \subseteq Q^*$ is a regular* word *language over Q.*

$\mathcal{A}$ is **deterministic** *if for every $f \in \Sigma$, if δ contains two rules $f(L_1) \to q_1$ and $f(L_2) \to q_2$, then $L_1 \cap L_2 = \emptyset$.*

We define a *move relation* $\to_\delta$ between ground terms in $T_{\Sigma \cup Q}$ as follows: for every two terms t and t', we have $t \to_\delta t'$ iff there exist a context C and a rule $r = f(L) \to q \in \delta$ such that $t = C\Big[f\big(q_1(t_1), \ldots, q_n(t_n)\big)\Big]$, $q_1 \cdots q_n \in L$, and $t' = C\Big[q\big(f(t_1, \ldots, t_n)\big)\Big]$.

Let $\overset{*}{\to}_\delta$ denote the reflexive-transitive closure of $\to_\delta$. A ground term $t \in T_\Sigma$ is accepted by a state q if $t \overset{*}{\to}_\delta q(t)$. Let $L_q = \{t \mid t \overset{*}{\to}_\delta q(t)\}$. A ground term t is accepted by the automaton $\mathcal{A}$ if there is some state q in F such that $t \overset{*}{\to}_\delta q(t)$.

The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of all ground terms accepted by $\mathcal{A}$. A set of terms $\mathcal{L}$ over Σ is *hedge regular* if there exists a hedge automaton $\mathcal{A}$ such that $\mathcal{L} = L(\mathcal{A})$.

Intuitively, given an input term t, a run of $\mathcal{A}$ on t according to the move relation $\rightarrow_\delta$ can be done in a bottom-up manner as follows: first, we assign nondeterministically a state q to each leaf labeled with symbol f if there is in δ a rule of the form $f(L) \rightarrow q$ s.t. $\epsilon \in L$. Then, for each node labeled with a symbol g, and having the terms $t_1, \ldots, t_n$ as children, we must collect the states $q_1, \ldots, q_n$ assigned to all its children, i.e., such that $t_i \stackrel{*}{\rightarrow}_\delta q_i(t_i)$, for $1 \leq i \leq n$, and then associate a state q to the node itself if there exists in δ a rule $r = g(L) \rightarrow q$ such that $q_1 \cdots q_n \in L$. A term t is accepted if $\mathcal{A}$ reaches the root of t in a final state.

Theorem 1. *[18] The class of Hedge automata is effectively closed under determinization and under boolean operations. Moreover, the emptiness problem for Hedge automata is decidable.*

2.3 Relabeling hedge transducers and relations

Definition 2. *A* **Relabeling Hedge Transducer** *is a tuple* $\mathcal{T} = (Q, \Sigma, F, \Delta)$ *where* Q *is a finite set of states,* Σ *is an unranked alphabet,* $F \subseteq Q$ *is a set of final states, and* Δ *is a set of rules of the form* $f(L) \rightarrow q(g)$*, where* $f, g \in \Sigma$*,* $q \in Q$*, and* $L \subseteq Q^*$ *is a regular* word *language over* Q*.*

As for hedge automata, a *relabeling hedge transducer* defines a *move relation* $\rightarrow_\Delta$ between ground terms in $T_{\Sigma \cup Q}$ as follows: for every two terms t and t', we have $t \rightarrow_\Delta t'$ iff there exist a context C and a rule $r = f(L) \rightarrow q(g) \in \Delta$ such that $t = C\Big[f\big(q_1(t_1), \ldots, q_n(t_n)\big)\Big]$, $q_1 \cdots q_n \in L$, and $t' = C\Big[q\big(g(t_1, \ldots, t_n)\big)\Big]$.

Let $\stackrel{*}{\rightarrow}_\Delta$ denote the reflexive-transitive closure of $\rightarrow_\Delta$. The transducer $\mathcal{T}$ defines the following relation between unbounded width trees: $R_\mathcal{T} = \{(t, t') \in T_\Sigma \times T_\Sigma \mid t \stackrel{*}{\rightarrow}_\Delta q(t'),$ for some $q \in F\}$. Note that $R_\mathcal{T}$ is structure preserving, i.e., if $(t, t') \in R_\mathcal{T}$, then t and t' correspond to two different labelings of the same skeleton tree.

Remark 1. Let f and g be two letters in Σ. We represent the pair (f, g) by f/g. Let t and t' be two terms corresponding to different labelings λ_1 and λ_2 of the same underlying tree u. We define the term t/t' as the labeling λ_3 of u such that for every node N of u, $\lambda_3(N) = \lambda_1(N)/\lambda_2(N)$.

A relabeling hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$ can be seen as a hedge automaton $\mathcal{A} = (Q, \Sigma \times \Sigma, F, \delta)$ over the alphabet $\Sigma \times \Sigma$, where δ is the set of rules $f/g(L) \rightarrow q$ s.t. $f(L) \rightarrow q(g) \in \Delta$. Then it is easy to see that $L(\mathcal{A}) = \{t/t' \mid (t, t') \in R_\mathcal{T}\}$.

A relation R over T_Σ is hedge regular if there exists a relabeling hedge transducer $\mathcal{T}$ such that $R = R_\mathcal{T}$. We denote by $R^n_\mathcal{T}$ the composition of $R_\mathcal{T}$,

n times. As usual, $R_{\mathcal{T}}^{+} = \bigcup_{n \geq 1} R_{\mathcal{T}}^{n}$ denotes the transitive closure of $R_{\mathcal{T}}$. Let $L \subseteq T_{\Sigma}$ be a hedge tree language. Then, we define the set $R_{\mathcal{T}}(L) = \{t' \in \Sigma \mid \exists t \in L, (t, t') \in R_{\mathcal{T}}\}$.

We show in what follows that hedge regular relations are closed under composition, and that they preserve regularity of hedge languages. First, we need to define the product of regular word languages as follows:

Definition 3. *Let $L_1, \ldots, L_n$ be n regular word languages over the alphabet Q. The product $L_1 \otimes \cdots \otimes L_n$ is defined by:*

$$L_1 \otimes \cdots \otimes L_n = \{(q_1^1, \ldots, q_n^1) \cdots (q_1^m, \ldots, q_n^m) \mid q_i^1 \cdots q_i^m \in L_i, 1 \leq i \leq n\}$$

Let $A_1, \ldots, A_n$ s.t. $A_i = (P_i, Q, T_i, P_F^i, P_0^i)$ be n word automata that recognize $L_1, \ldots, L_n$, where P_i are the sets of states, Q is the alphabet, T_i are the transitions, and P_F^i and P_0^i denote respectively the sets of final and initial states. It is easy to see that $L_1 \otimes \cdots \otimes L_n$ is recognized by the automaton $A = A_1 \otimes \cdots \otimes A_n$ defined as follows: $A = (P, Q^n, T, P_F, P_0)$ such that:

- $P = P_1 \times \cdots \times P_n$;
- $P_0 = P_0^1 \times \cdots \times P_0^n$;
- $P_F = P_F^1 \times \cdots \times P_F^n$;
- $T = \{\big((p_1, \ldots, p_n), (q_1, \ldots, q_n), (p'_1, \ldots, p'_n)\big) \mid (p_i, q_i, p'_i) \in T_i\}$.

Let $\mathcal{A} = (Q_1, \Sigma, F_1, \delta_1)$ be a hedge tree automaton and $\mathcal{T} = (Q_2, \Sigma, F_2, \Delta_2)$ be a relabeling hedge transducer. Let $\mathcal{B} = (Q, \Sigma, F, \delta)$ be the hedge tree automaton such that $Q = Q_1 \times Q_2$, $F = F_1 \times F_2$, and δ is the set of rules $g(L) \rightarrow (q_1, q_2)$ such that there exists two rules $f(L_1) \rightarrow q_1 \in \delta_1$ and $f(L_2) \rightarrow q_2(g) \in \Delta_2$ such that $L = L_1 \otimes L_2$.

Then we have the following:

Lemma 1. $L(\mathcal{B}) = R_{\mathcal{T}}\big(L(\mathcal{A})\big)$.

Let $\mathcal{T} = (Q, \Sigma, F, \Delta)$, and let the relabeling hedge transducer $\mathcal{T}_n = (Q_n, \Sigma, F_n, \Delta_n)$ defined as follows: $Q_n = Q^n$, $F_n = F^n$, and Δ_n is the set of rules of the form $f(L) \rightarrow (q_1, \ldots, q_n)(g)$ such that there exist in Δ rules of the form $f_i(L_i) \rightarrow q_i(f_{i+1}), 1 \leq i \leq n$, s.t. $f_1 = f$, $f_{n+1} = g$, and $L = L_1 \otimes \cdots \otimes L_n$.

Then we can show that:

Lemma 2. $R_{\mathcal{T}_n} = R_{\mathcal{T}}^{n}$.

3 Computing transitive closures

Our goal in this work is to compute a relabeling hedge transducer that recognizes the transitive closure $R_{\mathcal{T}}^{+}$ of a regular hedge relation $R_{\mathcal{T}}$. Unfortunately, this is not possible in general since the transitive closures are not necessarily hedge regular. Therefore, our purpose is to propose a *semi-algorithm* that, *in*

case of termination, computes a relabeling hedge transducer that recognizes the transitive closure $R_{\mathcal{T}}^+$.

More precisely, starting from a relabeling hedge transducer $\mathcal{T}$, we derive a transducer, called the *history hedge transducer* that characterizes the transitive closure $R_{\mathcal{T}}^+$. The set of states of the history transducer is infinite. To tackle this issue, we present a method (that is not guaranteed to terminate) for computing a finite-state transducer which is an abstraction of the history transducer, based on a notion of an equivalence relation on the states of the history transducer. The abstract transducer can be generated on-the-fly by a procedure which starts from the original transducer $\mathcal{T}$, and then incrementally adds new states and transition rules, merging equivalent states.

Let us first give the formal definition of the history hedge transducer:

Definition 4. *The* history hedge transducer *of a relabeling hedge transducer* $\mathcal{T} = (Q, \Sigma, F, \Delta)$ *is the (infinite) transducer given by the tuple* $\mathcal{H} = (Q_H, \Sigma, F_H, \Delta_H)$ *such that:* $Q_H = \bigcup_{n\geq 1} Q_n$, $F_H = \bigcup_{n\geq 1} F_n$, *and* $\Delta_H = \bigcup_{n\geq 1} \Delta_n$.

Since $R_{\mathcal{T}}^n = R_{\mathcal{T}_n}$ (Lemma 2), and by definition $R_{\mathcal{H}} = \bigcup_{n\geq 1} R_{\mathcal{T}_n}$, it follows that:

Theorem 2. $R_{\mathcal{T}}^+ = R_{\mathcal{H}}$.

As mentioned previously, $\mathcal{H}$ cannot be computed in general since it has an infinite number of states. To sidestep this problem, we will compute an equivalent smaller transducer $\mathcal{H}_\sim$ (that might be finite), obtained by merging the states of $\mathcal{H}$ according to an equivalence $\sim$ on Q_H. This transducer is defined as $\mathcal{H}_\sim = (Q_\sim, \Sigma, F_\sim, \Delta_\sim)$ such that:

- $Q_\sim = \{q_\sim \mid q \in Q_H\}$, where $q_\sim$ denotes the equivalence class of the state q w.r.t. $\sim$;
- $F_\sim = \{q_\sim \mid q \in F_H\}$ is the set of equivalence classes of F_H w.r.t. $\sim$;
- $\Delta_\sim$ is the set of rules $f(L_\sim) \to s_\sim(g)$ such that $f(L) \to s(g)$ is a rule in Δ_H, where $L_\sim$ is obtained from L by substituting each state q by its equivalence class $q_\sim$.

We compute $\mathcal{H}_\sim$ iteratively according to the following *procedure*:

1. We compute successive powers of $\mathcal{T}$: $\mathcal{H}^{\leq 1}$, $\mathcal{H}^{\leq 2}$, $\mathcal{H}^{\leq 3}$,... (where $\mathcal{H}^{\leq i} = \bigcup_{j=1}^{i} \mathcal{T}_j$) while collapsing states according to $\sim$. We obtain the sequence of transducers $\mathcal{H}_\sim^{\leq 1}$, $\mathcal{H}_\sim^{\leq 2}$, $\mathcal{H}_\sim^{\leq 3}$,...
2. If at step i we obtain that $R_{\mathcal{H}_\sim^{\leq i-1}} = R_{\mathcal{H}_\sim^{\leq i}}$, the procedure terminates.

This procedure is not guaranteed to terminate, but if it does, it is clear that the obtained transducer $\mathcal{H}_\sim^{\leq i}$ is equivalent to $\mathcal{H}_\sim$ (i.e. $R_{\mathcal{H}_\sim^{\leq i}} = R_{\mathcal{H}_\sim}$). The problem then amounts to *defining an appropriate equivalence relation* $\sim$ for which $R_{\mathcal{H}_\sim} = R_{\mathcal{H}}$. More generally, since a relabeling hedge transducer can be seen as a hedge automaton (Remark 1), in the next section, we define for every hedge automaton $\mathcal{A}$ an equivalence $\sim$ such that $L(\mathcal{A}_\sim) = L(\mathcal{A})$.

4 Appropriate equivalences for hedge automata

Let $\mathcal{A} = (Q, \Sigma, F, \delta)$ be a hedge automaton. We define in this section an appropriate equivalence $\sim$ on the set of states Q such that $L(\mathcal{A}_\sim) = L(\mathcal{A})$. To do so, we first define two simulation relations, namely a *downward simulation* $\preccurlyeq_{down}$ and an *upward simulation* $\preccurlyeq_{up}$ on Q, and then we show how to generate an appropriate equivalence $\sim$ from these simulations.

4.1 Downward and upward simulations

We introduce here the notion of downward and upward simulation for hedge automata:

Definition 5. [**Downward Simulation**]
A binary relation $\preccurlyeq_{down}$ on Q is a downward simulation iff for any symbol $f \in \Sigma$, for all states $q, r \in Q$, we have:
Whenever $q \preccurlyeq_{down} r$, $f(L) \rightarrow q \in \delta$, then for every states $q_1, \ldots, q_n \in Q$ s.t. $q_1 \cdots q_n \in L$, there exist states $r_1, \ldots, r_n \in Q$ and a rule $f(L') \rightarrow r$ in δ such that $q_1 \preccurlyeq_{down} r_1, \ldots, q_n \preccurlyeq_{down} r_n$, and $r_1 \cdots r_n \in L'$.

It is easy to see that if $q \preccurlyeq_{down} r$, then whenever a term t is accepted by state q (i.e., $t \xrightarrow{*}_\delta q(t)$), it is also accepted by state r.

Lemma 3. *Let $\preccurlyeq_{down}$ be a downward simulation on Q. The reflexive closure and the transitive closure of $\preccurlyeq_{down}$ are both downward simulations. Furthermore, there is a unique maximal downward simulation on Q.*

Definition 6. [**Upward Simulation**]
Given a downward simulation $\preccurlyeq_{down}$ on Q, a binary relation $\preccurlyeq_{up}$ on Q is an upward simulation w.r.t. $\preccurlyeq_{down}$ iff for any symbol $f \in \Sigma$, for all states $q_i, r_i \in Q$, the following holds:
Whenever $q_i \preccurlyeq_{up} r_i$ and $f(L) \rightarrow q \in \delta$, then for every states $q_1, \ldots, q_n \in Q$ s.t. $q_1 \cdots q_i \cdots q_n \in L$, there exist states $r_1, \ldots, r_n, r \in Q$ and a rule $f(L') \rightarrow r$ in δ such that $q_j \preccurlyeq_{down} r_j$, for $j \neq i$, $r_1 \cdots r_n \in L'$, and $q \preccurlyeq_{up} r$.

It is easy to see that whenever $q \preccurlyeq_{up} r$, for every context C and every terms $t_1, \ldots, t_n, t', t$ such that $t = C[t_1, \ldots, t_i, t', t_{i+1}, \ldots, t_n]$ and

$$C[t_1, \ldots, t_i, q(t'), t_{i+1}, \ldots, t_n] \xrightarrow{*}_\delta s(t)$$

for a state s; then there exists a state s', $s \preccurlyeq_{up} s'$ such that:

$$C[t_1, \ldots, t_i, r(t'), t_{i+1}, \ldots, t_n] \xrightarrow{*}_\delta s'(t)$$

Lemma 4. *Let $\preccurlyeq_{down}$ be a reflexive (transitive) downward simulation on Q, and let $\preccurlyeq_{up}$ be an upward simulation w.r.t. $\preccurlyeq_{down}$. The reflexive (transitive) closure of $\preccurlyeq_{up}$ is also an upward simulation w.r.t. $\preccurlyeq_{down}$. Furthermore, there is a unique maximal upward simulation on Q.*

4.2 Induced equivalence

We define an equivalence relation derived from two binary relations:

Definition 7. *Two binary relations $\preceq_1$ and $\preceq_2$ are said to be* independent *iff whenever $q \preceq_1 r$ and $q \preceq_2 r'$, there exists s such that $r \preceq_2 s$ and $r' \preceq_1 s$. Moreover, the relation $\sim$* induced *by $\preceq_1$ and $\preceq_2$ is defined as:*

$$\preceq_1 \circ \preceq_2^{-1} \cap \preceq_2 \circ \preceq_1^{-1} .$$

In [4], Abdulla et al. have shown the following fact:

Lemma 5. *Let $\preceq_1$ and $\preceq_2$ be two binary relations. If $\preceq_1$ and $\preceq_2$ are reflexive, transitive, and independent, then their induced relation $\sim$ is an* equivalence relation*. Moreover, whenever $x \sim y$ and $x \preceq_1 z$, there exists t such that $y \preceq_1 t$ and $z \preceq_2 t$.*

4.3 Defining an appropriate equivalence

Let $\mathcal{A} = (Q, \Sigma, F, \delta)$ be a hedge automaton. Let $\preccurlyeq_{down}$ be a downward simulation, and let $\preccurlyeq_{up}$ be an upward simulation w.r.t. $\preccurlyeq_{down}$. Thanks to Lemmas 3 and 4, we suppose without loss of generality that $\preccurlyeq_{down}$ and $\preccurlyeq_{up}$ are reflexive and transitive. Let $\preceq$ be a reflexive and transitive relation included in $\preccurlyeq_{up}$ such that $\preccurlyeq_{down}$ and $\preceq$ are independent, and let $\sim$ be the relation induced by $\preccurlyeq_{down}$ and $\preceq$. It follows from Lemma 5 that $\sim$ defines an equivalence relation on states of $\mathcal{A}$. Suppose in addition that:

- whenever $x \in F$ and $x \preccurlyeq_{up} y$, then $y \in F$; and that
- if $X \in F_\sim$ and $x \in X$, then $x \in F$.

In this case, we show that $\sim$ is an appropriate equivalence.

Theorem 3. $L(\mathcal{A}_\sim) = L(\mathcal{A})$.

5 An instance of an appropriate equivalence

Let us now come back to our relabeling hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$ and its corresponding history transducer $\mathcal{H} = (Q_H, \Sigma, F_H, \Delta_H)$. We suppose that $\mathcal{T}$ is deterministic (this is not a restriction thanks to Theorem 1 and Remark 1). Recall that our purpose is to *effectively* compute an appropriate equivalence relation $\sim$ on Q_H such that $L(\mathcal{H}_\sim) = L(\mathcal{H})$. We give in this section an example of a *computable* equivalence $\sim$ on Q_H induced by a downward simulation $\preccurlyeq_{down}$, an upward simulation w.r.t. $\preccurlyeq_{down}$, and a relation $\preceq$ satisfying the conditions required in the previous section.

First, we need to introduce the notion of *copying* states:

Definition 8 (Copying States). *Let $q \in Q$ be a state:*

– q *is a* prefix copying *state iff for every term* t*:*

$$t \xrightarrow{*}_{\Delta} q(t') \text{ iff } t = t'$$

– q *is a* suffix copying *state iff for every term* t*, context* C*, and* $q_F \in F$*:*

$$C\big[q(t)\big] \xrightarrow{*}_{\Delta} q_F\big(C'[t]\big) \text{ iff } C = C'$$

Let S be a set in $Q_H \times Q_H$. We define the relation R_S generated by S as the smallest reflexive-transitive relation that contains S and that is a congruence with respect to product, i.e., if $\big((q_1, \ldots, q_n), (q'_1, \ldots, q'_m)\big) \in R_S$, then for any $s_1, \ldots, s_k, s'_1, \ldots, s'_l$ in Q_H,

$$\big((s_1, \ldots, s_k, q_1, \ldots, q_n, s'_1, \ldots, s'_l), (s_1, \ldots, s_k, q'_1, \ldots, q'_m, s'_1, \ldots, s'_l)\big) \in R_S$$

Lemma 6. *If the set* S *is a downward (resp. upward) simulation on* $Q_{\mathcal{H}}$*, then its generated relation* $\preceq_S$ *is also a downward (resp. upward) simulation.*

Let Q_{pref} be the set of prefix copying states of $\mathcal{T}$, and Q_{suff} be the set of suffix copying states of $\mathcal{T}$ that are not in Q_{pref}. Let $\preccurlyeq_{down}$ be the binary relation on $Q_H \times Q_H$ generated by the set

$$\{\big((q,q),q\big), \big(q,(q,q)\big) \mid q \in Q_{pref}\}$$

We show that $\preccurlyeq_{down}$ is a downward simulation:

Lemma 7. $\preccurlyeq_{down}$ *is a downward simulation.*

Let $\preccurlyeq_{up}$ be the binary relation on $Q_H \times Q_H$ generated by the set

$$\{\big((q,q),q\big), \big(q,(q,q)\big) \mid q \in Q_{suff}\}$$

Then we have:

Lemma 8. $\preccurlyeq_{up}$ *is an upward simulation w.r.t.* $\preccurlyeq_{down}$*.*

Let $\preceq = \preccurlyeq_{up}$. Then, we can show that $\preceq$ and $\preccurlyeq_{down}$ are independent:

Lemma 9. $\preceq$ *and* $\preccurlyeq_{down}$ *are independent.*

Let then $\sim$ be the relation induced by $\preccurlyeq_{down}$ and $\preceq$. We show that the conditions of Theorem 3 are satisfied:

Lemma 10. *Whenever* $x \in F_H$ *and* $x \preccurlyeq_{up} y$*, then* $y \in F_H$*. Moreover, if* $X \in F_\sim$ *and* $x \in X$*, then* $x \in F_H$*.*

It follows then from Theorem 3 that:

Theorem 4. $L(\mathcal{H}_\sim) = L(\mathcal{H})$*.*

Remark 2. Note that both $\preccurlyeq_{down}$ and $\preccurlyeq_{up}$ are included in $\sim$ (this is due to the fact that these relations are reflexive and symmetric).

Now, it remains to show how can the equivalence $\sim$ be effectively computed.

For this, we need to compute the sets of copying states Q_{pref} and Q_{suff}. This is described next.

Input:
Hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$, and a state q.
Begin
$d := \{q\}$
Repeat
for each $q_1 \in d$, and for each rule $r = f(L) \to g(q_1)$,
add $\{q_2 \mid L \cap (Q^* q_2 Q^*) \neq \emptyset\}$ to d.
Until No more additions can be made
End
Output:
"Yes" if all rules r encountered were copying (i.e. such that $f = g$).
"No" otherwise.

Fig. 1. Determining whether a state is prefix copying.

5.1 Computing copying states

The algorithm for checking whether a state q is prefix copying is shown in Figure 1. Intuitively, the algorithm works as follows: it tries to explore all rules r useful for computing the language of $\mathcal{T}$ with q as the only accepting state. If all such rules r are of the form $f(L_1) \to f(q_1)$, then q is indeed prefix-copying.

Input:
Hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$, and a state q.
Begin
$up := \{q\}$, $side := \emptyset$
Repeat
for each $q_1 \in up$, and for each rule $r = f(L) \to g(q_2)$
such that $L \cap (Q^* q_2 Q^*) \neq \emptyset$, then
add q_2 to up, and add $\{q' \mid L \cap (Q^* q' Q^*) \neq \emptyset \wedge q' \neq q\}$ to $side$.
Until No more additions can be made
End
Output:
"Yes" if all rules r encountered were copying (i.e. such that $f = g$)
and all states in $side$ are prefix-copying and there is a final state in up.
"No" otherwise.

Fig. 2. Determining whether a state is suffix copying.

The algorithm for checking whether a state q is suffix copying is shown in Figure 2. Intuitively, the algorithm explores all rules r leading from state q to a final state according to the move relation for $\mathcal{T}$. We must first check that all rules r encountered are copying rules. However, the test performed until now only checks what lies on the path from q up to the root of an accepted context. Therefore, we need to also check what's happening to the child nodes along this root path. This is the purpose of the variable *side*. Any subtree attached to a

child of the root path is accepted by some state in *side*. Hence, we require that all states in *side* are prefix copying.

6 Applications

In this section, we give the results of applying the procedure of Section 3 to the analysis of two mutual exclusion protocols defined on arbitrary width tree-like networks, and of an XML application.

6.1 The unranked simple token protocol

We consider the example of the *unranked simple token protocol*, which is a mutual exclusion protocol defined on arbitrary width tree-like networks. Each process stores a single bit which reflects whether the process has a token or not. The process that has the token has the right to enter the critical section. In this system, the token can move from a leaf upward to the root in the following fashion: any process that currently has the token can release it to its parent. Initially, the system contains exactly one token, located anywhere.

More formally, the passing of the token upward the tree can be represented by the following relabeling hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$ where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{n, t\}$, $F = \{q_2\}$, and Δ contains the rules:

$$n(q_0^*) \to q_0(n)\ (1) \qquad t(q_0^*) \to q_1(n)\ (2)$$
$$n(q_0^* q_1 q_0^*) \to q_2(t)\ \ (3) \qquad n(q_0^* q_2 q_0^*) \to q_2(n)\ (4)$$

The intuition behind the states of the transducer is the following.

- State q_0 is meant to accepts all "pairs" of identical trees where the token doesn't appear. This is a prefix-copying state.
- State q_1 is an intermediate state meaning that the current node released the token. Its parent then acquires the token.
- State q_2 is the final state of the transducer. It accepts all "pairs" of trees in which the token has moved one step upward. This is a suffix-copying state.

According to the algorithm of Figure 1, we get $Q_{pref} = \{q_0\}$, and with the algorithm of Figure 2, we get $Q_{suff} = \{q_3\}$.

Let us now apply the algorithm described in Section 3. We will compute the different iterations $\mathcal{H}_{\simeq}^{\leq 1}, \ldots, \mathcal{H}_{\simeq}^{\leq i}$. We terminate at step i if $R_{\mathcal{H}_{\simeq}^{\leq i-1}} = R_{\mathcal{H}_{\simeq}^{\leq i}}$.

Computing $\mathcal{H}_{\simeq}^{\leq 1}$: Take each rule in $\mathcal{T}$ and substitute each occurence of a state q with its equivalence class $q_\sim$ w.r.t. $\sim$.

$$n(q_{0\sim}^*) \to q_{0\sim}(n)\ (1) \qquad t(q_{0\sim}^*) \to q_{1\sim}(n)\ (2)$$
$$n(q_{0\sim}^* q_{1\sim} q_{0\sim}^*) \to q_{2\sim}(t)\ \ (3) \qquad n(q_{0\sim}^* q_{2\sim} q_{0\sim}^*) \to q_{2\sim}(n)\ (4)$$

Computing $\mathcal{H}_{\simeq}^{\leq 2}$: Take $\mathcal{H}_{\simeq}^{\leq 1}$ and add rules

$$t(q_{0\sim}^*) \to (q_1, q_0)_\sim(n)\ (5) = (2) \otimes (1)$$
$$n(q_{0\sim}^*(q_1, q_0)_\sim q_{0\sim}^*) \to (q_2, q_1)_\sim(n)\ (6) = (3) \otimes (2)$$
$$n(q_{0\sim}^*(q_2, q_1)_\sim q_{0\sim}^*) \to q_{2\sim}(t)\ \ (7) = (4) \otimes (3)$$

For example, rule (5) is obtained by composing rules (2) and (1). The resulting product is the rule $t((q_0,q_0)^*) \to (q_1,q_0)(n)$ (denoted (2)⊗(1) above). Since $(q_0,q_0) \preccurlyeq_{down} q_0$ ($q_0 \in Q_{pref}$) and $\preccurlyeq_{down} \subseteq \sim$ (Remark 2), we get that $(q_0,q_0) \sim q_0$. Therefore, merging w.r.t. $\sim$, we get rule (5).

Note that rule (7) has been simplified. Indeed, performing the product of the rules (4) and (3) yields the rule $n(L) \to q_{2\sim}(t)$, where L is the following regular word language: $(q_0,q_0)^*(q_2,q_0)(q_0,q_0)^*(q_0,q_1)(q_0,q_0)^* + (q_0,q_0)^*(q_0,q_1)(q_0,q_0)^*(q_2,q_0)(q_0,q_0)^* + (q_0,q_0)^*(q_2,q_1)(q_0,q_0)^*$. For the sake of brevity, We omit the first part of L since the states (q_2,q_0) and (q_0,q_1) are not reachable.

Computing $\mathcal{H}_{\approx}^{\leq 3}$: Take $\mathcal{H}_{\approx}^{\leq 2}$ and add the following rules obtained as described previously:

$n(q_{0\sim}^*(q_1,q_0)_\sim q_{0\sim}^*) \to (q_2,q_1,q_0)_\sim(n)$ (8) = (3)⊗(5) = (6)⊗(1)
$n(q_{0\sim}^*(q_2,q_1,q_0)_\sim q_{0\sim}^*) \to (q_2,q_1)_\sim(n)$ (9) = (7)⊗(2) = (4)⊗(6)

Computing $\mathcal{H}_{\approx}^{\leq 4}$: Take $\mathcal{H}_{\approx}^{\leq 3}$ and add the following rule:

$n(q_{0\sim}^*(q_2,q_1,q_0)_\sim q_{0\sim}^*) \to (q_2,q_1,q_0)_\sim(n)$ (10) = (9)⊗(1) = (4)⊗(8)

The procedure terminates at step 4, since subsequent iterations do not change the accepted language.

6.2 The unranked two-way token protocol

This mutual exclusion protocol is similar to the *Simple Token Protocol* above, with the following difference: the node that currently owns the token can release it to its parent neighbor, or it can release it to one of its child neighbors. Thus, the token can move upward, as well as downward inside the tree of processes.

Formally, these transformations can be represented by the following relabeling hedge transducer $\mathcal{T} = (Q, \Sigma, F, \Delta)$, where $Q = \{q_0,q_1,q_2,q_3\}$, $\Sigma = \{n,t\}$, $F = \{q_3\}$, and Δ contains the rules:

$n(q_0^*) \to q_0(n)$ (1) $n(q_0^*) \to q_1(t)$ (2) $t(q_0^*) \to q_2(n)$ (3)
$t(q_0^*q_1q_0^*) \to q_3(n)$ (4) $n(q_0^*q_2q_0^*) \to q_3(t)$ (5) $n(q_0^*q_3q_0^*) \to q_3(n)$ (6)

The intuition behind the states of the transducer is as follows:

- State q_0 accepts all "pairs" of identical trees where the token never appears. This state is prefix-copying.
- State q_1 is the intermediate state denoting that the current node just acquired the token. Its parent neighbor releases the token.
- State q_2 is also an intermediate state. It means that the current node releases the token. The parent node acquires the token.
- State q_3 is the final state. It accepts all "pairs" of trees in which the token has moved one step upward or downward. This state is suffix-copying.

Computing $\mathcal{H}_{\approx}^{\leq 1}$: Take $\mathcal{T}$ and replace occurences of a state in a rule of Δ with its equivalence class w.r.t. $\sim$.

$n(q_{0\sim}^*) \to q_{0\sim}(n)$ (1) $n(q_{0\sim}^*) \to q_{1\sim}(t)$ (2)
$t(q_{0\sim}^*) \to q_{2\sim}(n)$ (3) $t(q_{0\sim}^*q_{1\sim}q_{0\sim}^*) \to q_{3\sim}(n)$ (4)
$n(q_{0\sim}^*q_{2\sim}q_{0\sim}^*) \to q_{3\sim}(t)$ (5) $n(q_{0\sim}^*q_{3\sim}q_{0\sim}^*) \to q_{3\sim}(n)$ (6)

Computing $\mathcal{H}_{\sim}^{\leq 2}$: Take $\mathcal{H}_{\sim}^{\leq 1}$ and add rules

$$\begin{aligned}
n(q_{0\sim}^*) &\to (q_0, q_1)_\sim(t) && (7) = (1)\otimes(2)\\
n(q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*) &\to (q_1, q_3)_\sim(n) && (8) = (2)\otimes(4)\\
t(q_{0\sim}^*(q_1, q_3)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (9) = (4)\otimes(6)\\
t(q_{0\sim}^*) &\to (q_2, q_0)_\sim(n) && (10) = (3)\otimes(1)\\
n(q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*) &\to (q_3, q_2)_\sim(n) && (11) = (5)\otimes(3)\\
n(q_{0\sim}^*(q_3, q_2)_\sim q_{0\sim}^*) &\to q_{3\sim}(t) && (12) = (6)\otimes(5)\\
t(q_{0\sim}^*) &\to (q_2, q_1)_\sim(t) && (13) = (3)\otimes(2)\\
n(q_{0\sim}^*(q_2, q_1)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (14) = (5)\otimes(4)\\
n(q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (14) = (5)\otimes(4)\\
n(q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (14) = (5)\otimes(4)\\
n(q_{0\sim}^*) &\to (q_1, q_2)_\sim(n) && (15) = (2)\otimes(3)\\
t(q_{0\sim}^*(q_1, q_2)_\sim q_{0\sim}^*) &\to q_{3\sim}(t) && (16) = (4)\otimes(5)
\end{aligned}$$

Computing $\mathcal{H}_{\sim}^{\leq 3}$: Take $\mathcal{H}_{\sim}^{\leq 2}$ and add rules

$$\begin{aligned}
n(q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*) &\to (q_0, q_1, q_3)_\sim(n) && (17) = (1)\otimes(8)\\
n(q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*) &\to (q_1, q_3)_\sim(n) && (18) = (2)\otimes(9)\\
n(q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*) &\to (q_3, q_2, q_0)_\sim(n) && (19) = (11)\otimes(1)\\
n(q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*) &\to (q_3, q_2)_\sim(n) && (20) = (12)\otimes(3)\\
n(q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (21) = (14)\otimes(6)\\
n(q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*(q_2, q_0)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (21) = (14)\otimes(6)\\
n(q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (22) = (6)\otimes(14)\\
n(q_{0\sim}^*(q_0, q_1)_\sim q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (22) = (6)\otimes(14)
\end{aligned}$$

Computing $\mathcal{H}_{\sim}^{\leq 4}$: Take $\mathcal{H}_{\sim}^{\leq 3}$ and add rules

$$\begin{aligned}
n(q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*) &\to (q_0, q_1, q_3)_\sim(n) && (23) = (1)\otimes(18)\\
n(q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*) &\to (q_3, q_2, q_0)_\sim(n) && (24) = (6)\otimes(19)\\
n(q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (25) = (6)\otimes(21)\\
n(q_{0\sim}^*(q_0, q_1, q_3)_\sim q_{0\sim}^*(q_3, q_2, q_0)_\sim q_{0\sim}^*) &\to q_{3\sim}(n) && (26) = (22)\otimes(6)
\end{aligned}$$

The procedure terminates at step 4, since subsequent iterations have the same language.

Note that some rules have been omitted if they contain unreachable states. Some redundant rules have been omitted as well, for the sake of simplicity.

6.3 An XML application

Figure 3 represents an XML document that stores the informations about the clients of a store and the items they bought. Each client has four fields: name, address, the different items that were bought, and the status of the order, i.e., whether the order is treated or not. `status` is 1 if the order is being treated, 0 if it has not been treated yet, and 2 if its treatment is finished. Initially, the first client has `status` 1, and the others 0. This document can be represented by the tree of Figure 4. Note that we need here arbitrary-width trees since the number of clients and the number of bought items are arbitrary.

```
<clients>
    <client>
          <name> Philipp </name>
          <address> ··· </address>
          <status> 1 </status>
          <items>
             <item> bed </item>
             <item> chair </item>
             ...
             <item> fridge </item>
          </items>
    </client>
    <client>
          <name>Maria </name>
          <address> ··· </address>
          <status> 0 </status>
          <items>
             <item> TV </item>
             <item> radio </item>
             ...
             <item> closet </item>
          </items>
    </client>
    ...
</clients>
```

Fig. 3. Part of a document containing information about the clients of a store

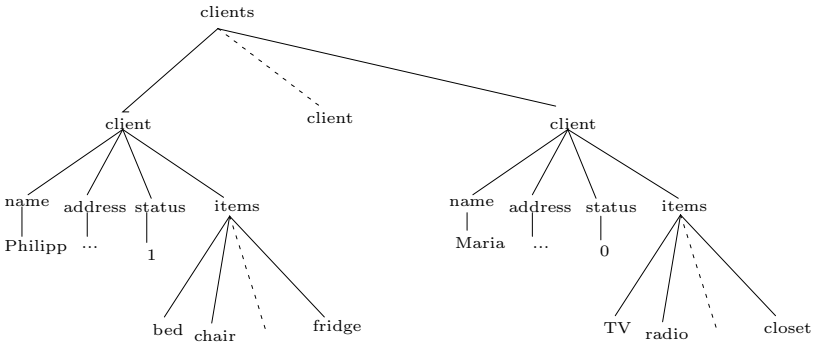

Fig. 4. The previous XML document as a tree

The store has a software that treats the clients in the order they appear in the XML document. The effect of one action of the software consists in changing the status of the current client (resp. the next one) to 2 (resp. to 1) to

express that the treatment of the current client is over, and that now we moved to the treatment of the next client. This transformation can be represented by the following relabeling hedge transducer $\tau = (Q, \Sigma, F, \Delta)$, where $Q = \{q, q_1, q_1', q_1'', q_2, q_2', q_2'', q_f\}$; $F = \{q_f\}$; $\Sigma = \Sigma' \cup \{name, address, item, items, status, client, clients, 1, 0, 2\}$, where Σ' is a finite alphabet that corresponds to the names, addresses, etc, and that is not relevant for us in this application; and Δ contains the following rules:

- For every $f \in \Sigma$, $f(q^*) \rightarrow q(f)$;
- $1(\epsilon) \rightarrow q_2(2)$: 1 is changed to 2;
- $status(q_2) \rightarrow q_2'(status)$;
- $client(q^* q_2' q^*) \rightarrow q_2''(client)$;
- $0(\epsilon) \rightarrow q_1(1)$: 0 is changed to 1;
- $status(q_1) \rightarrow q_1'(status)$;
- $client(q^* q_1' q^*) \rightarrow q_1''(client)$;
- $clients(q^* q_2'' q_1'' q^*) \rightarrow q_f$: we make sure that the client whose "1" has been changed into "2" is adjacent in the document (and therefore in the tree) to the client whose "0" has been changed into "1".

In order to check the behavior of this software, we need to compute the transitive closure τ^+. Our technique terminates in this example and computes τ^+. We skip here the details since they are similar to the previous examples.

7 Conclusion

In this paper, we have extended the regular model checking framework so that it can handle systems with arbitrary width tree-like structures. Since the central problem in regular model checking is the computation of transitive closures of transducers, the main contribution of this paper is a *general acceleration* technique that computes the transitive closure of a given hedge transducer. The technique is based on defining and *effectively* computing an equivalence relation used to collapse the states of the transitive closure of the hedge transducer. We have successfully applied our technique to compute transitive closures for (1) some mutual exclusion protocols defined on arbitrary width tree topologies; and (2) XML document transformations.

As future work, it would be interesting to see if one can extend our technique to handle non-structure preserving transducers. It would also be of interest to see if we can combine our simulation-based technique with other regular model checking techniques such as abstraction [11, 10] or learning [27, 20].

References

1. P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Simulation-based iteration of tree transducers. *Proceedings of TACAS'05*, 2005.

2. Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso. Regular tree model checking. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 555–568, 2002.
3. Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002,* 13^{th} *Int. Conf. on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130, 2002.
4. Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Algorithmic improvements in regular model checking. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 236–248, 2003.
5. Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.
6. Bernard Boigelot, Axel Legay, and Pierre Wolper. Omega regular model checking. In *Proc. TACAS '04,* 10^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 561–575, 2004.
7. A. Bouajjani. Languages, Rewriting systems, and Verification of Infinte-State Systems. In *ICALP'01*. LNCS 2076, 2001. invited paper.
8. A. Bouajjani, J. Esparza, and T. Touili. Reachability Analysis of Synchronised PA systems. In *INFINITY'04*. ENTCS, 2004.
9. A. Bouajjani, P. Habermehl, P. Moro, and T. Vojnar. Verifying programs with dynamic 1-selector-linked structures in regular model checking. *Proceedings of TACAS'05*, 2005.
10. A. Bouajjani, P. Habermehl, P. Moro, and T. Vojnar. Verifying programs with dynamic 1-selector-linked structures in regular model checking. In *TACAS05*, Lecture Notes in Computer Science, pages 13–29. Springer, 2005.
11. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV04*, Lecture Notes in Computer Science, pages 372–386, Boston, July 2004. Springer-Verlag.
12. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In Emerson and Sistla, editors, *Proc.* 12^{th} *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer Verlag, 2000.
13. A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. In *Proc. LICS' 01* 17^{th} *IEEE Int. Symp. on Logic in Computer Science*. IEEE, 2001.
14. A. Bouajjani and T. Touili. Reachability analysis of process rewrite systems. In *FSTTCS03*, Lecture Notes in Computer Science, pages 73–87, 2003.
15. A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In *Proc.* 16^{th} *Int. Conf. on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *Lecture Notes in Computer Science*, April 2005.
16. Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR'05*, LNCS, 2005.
17. Ahmed Bouajjani and Tayssir Touili. Extrapolating Tree Transformations. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 539–554, 2002.
18. A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Research report, 2001.

19. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297, 2001.
20. P. Habermehl and T. Vojnar. Regular model checking using inference of regular languages. In *Proc. of 6th International Workshop on Verification of Infinite-State Systems—Infinity'04*, pages 61–72, Sept. 2004.
21. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.
22. 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.
23. A. Pnueli and E. Shahar. Liveness and acceleration in parametrized verification. In *CAV'00*. LNCS, 2000.
24. H. Seidl, Th. Schwentick, and A. Muscholl. Numerical Document Queries. In *PODS'03*. ACM press, 2003.
25. T. Touili. Regular Model Checking using Widening Techniques. *Electronic Notes in Theoretical Computer Science*, 50(4), 2001. Proc. Workshop on Verification of Parametrized Systems (VEPAS'01), Crete, July, 2001.
26. T. Touili. Dealing with communication for dynamic multithreaded recursive programs. In *1st VISSAS workshop*, 2005. Invited Paper.
27. A. Vardhan, K.Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for FIFO automata. In *FSTTCS04*, Lecture Notes in Computer Science, pages 494–505, 2004.
28. Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, Vancouver, July 1998. Springer Verlag.
29. Silvano Dal Zilio and Denis Lugiez. Xml schema, tree logic and sheaves automata. In *RTA'03*, 2003.