Abstract Regular Tree Model Checking with Nondeterministic Automata

Lukáš Holík*
Department of Intelligent Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech republic
holik@fit.vutbr.cz

Abstract

Abstract regular tree model checking is an infinite state system verification method that is based on representing possibly infinite sets of states by tree automata. The method was originally defined over deterministic tree automata, involving expensive determinisation steps many times within a single verification run, which significantly limits scalability of the method. To avoid determinisation, we redesign the method on top of nondeterministic tree automata. For this, we develop needed efficient methods for reducing size and checking language inclusion of nondeterministic tree automata. Our experimental results confirm that the version of abstract regular tree model checking based on nondeterministic automata performs much better than the original deterministic automata-based version.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; F.1.2 [**Computation by Abstract Devices**]: Models of Computation—*Alternation and Nondeterminism*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*

Keywords

Finite automata, nondeterministic finite tree automata, simulation, language inclusion, antichain, quotienting, abstract regular tree model checking.

1. Introduction

Abstract regular tree model checking (ARTMC) is a method for verifying systems with unbounded or infinite number of states. Such systems arise often in preactice, for instance in various parametrised protocols or pointer manipulating programs. The basic idea of ARTMC is to represent possibly infinite sets of states by finite automata. In regular model checking (RMC), we start with a finite word automaton (FA) $\mathcal{A}_I$ representing a set of initial configurations I of a system and iteratively apply transition relation τ (symbolically, on the structure of the automaton) until a fixpoint is reached, thus computing an FA representing the set $\tau^*(\mathcal{A}_I)$ of all configurations reachable from the initial configurations. Then, it is checked whether this set satisfies the verified properties. In abstract regular model checking [8], abstraction (together with a counterexample guided refinement) is used to accelerate the computation. Checking the fixpoint condition means to decide whether $\tau^i(A_I) \subseteq \tau^{i+1}(\mathcal{A}_I)$, which requires an efficient language inclusion algorithm. During the computation, the intermediate automata typically grow quickly, therefore it is needed to reduce their size. Tree automata (TA) are used instead of FA when configurations of the system being verified are better represented by trees than by words, e.g., certain parametrised communication protocols, pointer programs manipulating tree-like data structures etc. In that case, we speak about abstract regular tree model checking (ARTMC) [9, 3, 6, 7].

ARTMC was originally based on deterministic tree automata, involving implicit determinisation after each step. However, this exponential step is very costly and significantly limits scalability of the method. In this work, we aim at avoiding the determinisation step by redesigning the method on top of nondeterministic tree automata. To do this, we need two ingredients. Methods for reducing size of nondeterministic tree automata and efficient methods for checking language inclusion of tree automata. Both problems are hard. The language inclusion problem and the minimisation problem for (nondeterministic) tree automata are EXPTIME-complete. A classical approach to cope with these problems is determinisation. TA can be determinised and minimised in a canonical way. Testing language inclusion of deterministic minimal automata is then easy. However, since even the canonical minimal deterministic automaton can still be exponentially larger than the original nondeterministic one, its computation easily becomes a major bottleneck of any automata-based method.

A reasonable and pragmatic approach to the size reduction and language inclusion problem is to consider some relation on states of an automaton that respects language

*Recommended by the thesis supervisor:
Assoc. Prof. Tomáš Vojnar
Defended at Faculty of Information Technology, Brno University of Technology on March 3, 2011.

Holík, L. Abstract Regular Tree Model Checking with Nondeterministic Automata. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 3, No. 2 (2011) 18-25

inclusion on states, but which can be checked efficiently, using a polynomial algorithm. Such a relation can then be used for approximating language inclusion between two automata by checking whether each initial state of one automaton is related to an initial state of other automaton. This method is sound but incomplete in the case when the relation is a proper subset the language inclusion on states. Such a relation can be also used for reducing the size of an automaton by collapsing equivalent states. Here, a natural trade-off between the strength of the considered relation and the cost of its computation arises. In the case of word automata, a relation which is widely considered as a good trade-off in this sense is simulation preorder. It can be checked in polynomial time, and efficient algorithms have been designed for this purpose (see, e.g., [13, 14, 19, 11]). These algorithms make the computation of simulation preorder quite affordable even in comparison with the one of bisimulation equivalence, which is cheaper [16, 18, 21], but which is also stronger, and therefore leads to less significant reductions of automata and also its capability of approximating language inclusion is limited.

As for what concerns language inclusion and universality problem, apart from the classical determinisation-based methods and simulation-based approximation technique, there has recently been proposed the so called antichain universality and inclusion testing method for FA [22]. It is essentially an optimisation of the classical method based on subset construction (i.e., on determinisation), it is still of an exponential worst case complexity, but it behaves very well in practice.

In the case of tree automata, the only methods for size reduction that were previously studied (apart from determinisitc minimization) are based on bisimulation relations [2, 15] and concerning language inclusion testing, the only methods formerly available are the classical ones based on explicit determinisation. However, these methods are not efficient enough. The former ones are rather weak since bisimulation relations are usually relatively sparse and the latter ones suffer from the problem of state space explosion too often.

The lack of efficient methods for reducing size and testing language inclusion of nondeterministic tree automata described above has significantly limited their practical usability not only in ARTMC. There, we aimed at adapting techniques that work well for word automata to tree automata, which in particular concerns the size reduction methods based on simulations and the language inclusion testing algorithms based on the antichain principle [22]. Apart from generalising existing methods from word automata to tree automata, we introduced new types of relations suitable for reducing the size of word as well as tree automata.

Tree Automata Reduction Methods. Our tree automata reduction methods are build on the notions of downward and upward tree automata simulations (proposed first in [4]) that are the tree automata counterparts the forward and backward FA simulations.

We design efficient algorithms for computing tree automata simulations. A deep examination of the structure of the TA simulations reveals that both upward and downward TA simulations can be computed by the same algorithmic pattern. More specifically, the problems of computing a TA simulation can be reduced to a problem of computing a common FA simulation (a tree automaton is translated into an FA and then a common FA simulation algorithm is used).

We have identified a principle of combining upward and downward TA simulations and forward and backward FA simulations that yields an equivalence, called mediated equivalence, suitable for reducing automata by collapsing their states while preserving the language. Mediated equivalence is coarser than downward resp. forward simulation equivalence and thus gives a better reduction. The principle of mediated minimisation of FA generalises the principle of forward simulation minimisation. Two forward simulation equivalent states can be safely collapsed since they have the same forward languages (symmetrically for backward simulation). In contrary, the property that allow collapsing two mediated equivalent states p and q is the following. Whenever there is a computation under a word u starting in an initial state that ends in a state p, and another computation under a word v starting in a state q and ending in a final state, then there is a computation under uv from an initial to a final state. Therefore, collapsing the two states p, q does not introduce any new behaviour since every word accepted via the new state was accepted also before collapsing. The case of TA mediated equivalence can be explained analogically. It may be seen from the above that unlike simulations, mediated equivalences approximate neither forward nor backward language equivalence on states, and similarly the tree automata mediated equivalence is not compatible with any notion of language of a state of a tree automaton. The combination principle allows to build a mediated equivalence from any downward/backward relation (simulation, bisimulation or identity relation) and any upward/forward relation (simulation, bisimulation, identity). This yields a scale of mediated equivalences offering a fine choice between the computation cost and reduction power, as confirmed by our experimental results.

Language Inclusion Checking for TA. Our universality and language inclusion algorithms for tree automata build on the antichain based method for FA proposed first in [22]. It is a complete method that optimises the classical subset construction based algorithms. We first briefly review its main idea.

Consider a nondeterministic FA $\mathcal{A}$. In the simpler case of universality checking, the method is based on a search for a nonaccepting state of the determinised version $\mathcal{A}'$ of $\mathcal{A}$ reachable from an initial state of $\mathcal{A}'$. Such a state is a counterexample to universality of $\mathcal{A}$. When a counterexample is reached, the algorithm may terminate even before all states of $\mathcal{A}'$ are constructed. The states of $\mathcal{A}'$, called macro-states, have the form of subsets of the set of states of $\mathcal{A}$. The key idea is that some macro-states have a better chance of finding a counterexample than other ones since they have provably smaller languages (in our terminology, we say that they subsume the states with larger languages). Therefore, one can safely continue searching only from the generated macro-states that have minimal languages, and simply discard any generated macro-state that is subsumed by another one. In [22], the subsump-

tion relation is just set inclusion, and already this simple solution gives a fundamental speedup.

We adapt the FA antichain technique for tree automata. The adaptation is quite straightforward, and similarly as in the case of FA, it has a major impact on efficiency of the TA language inclusion and universality tests.

Experimental Evaluation. Based on our new tree automata reduction and language inclusion techniques, we design a version of abstract regular tree model checking method on top of nondeterministic automata. We have implemented the method and compared its performance with the original deterministic automata-based version. The experiments show that the new version greatly outperform the original version.

Outline. In Section 2, we introduce preliminary notions that we build on. Section 3 is devoted to methods for reducing size of nondeterministic TA, Section 4 describes our TA language inclusion checking algorithm, Section 5 presents the nondeterministic automata based version of ARTMC and its experimental evaluation. Section 6 concludes the paper.

More details and full proofs of theorems and lemmas presented may be found in [4, 5, 1].

2. Preliminaries

Here we give preliminaries on relations, labelled transition systems, finite automata, simulations, tree automata, and regular tree model checking that we build on in this work.

Relations. Given a binary relation $R \subseteq X \times X$ on a set X, we often use the infix notation xRy to denote that $(x, y) \in R$. Given a subset Y of X, the relation $R \cap Y \times Y$ is the *restriction* of R to Y. For an equivalence relation $\equiv$ on X, we use $X/\equiv$ to denote the partitioning of X according to $\equiv$, and we call an equivalence class of $\equiv$ a *block*. For two relations R and S, we denote $R \circ S$ their *composition* where $x(R \circ S)y \iff \exists z : xRzSy$.

Labelled Transition Systems and Finite Automata. A (finite) *labelled transition system (LTS)* is a tuple $\mathcal{T} = (\Sigma, Q, \delta)$ where Q is a finite set of states, Σ is a finite set of labels, and $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation. Given two states $q, r \in Q$, we denote by $q \xrightarrow{a} r$ that $(q, a, r) \in \delta$.

Forward and Backward Simulations. A *(forward) simulation* over an LTS $\mathcal{T} = (\Sigma, Q, \delta)$ is a binary relation R on Q such that for any states q, r, q', if qRr and $q \xrightarrow{a} q'$, then there is a state r' with $r \xrightarrow{a} r'$ and $q'Rr'$.

Any given simulation on an LTS can be closed under reflexivity, transitivity and union, and so there is a unique maximal simulation on the given LTS, called the *simulation preorder*. It also holds that, for any given *initial* preorder $I \subseteq Q \times Q$, the set of simulations over $\mathcal{T}$ included in I is closed under union, reflexive and transitive closure, and thus there is a unique maximal simulation included in I on $\mathcal{T}$, which we call *the simulation preorder included in* I.

Backward simulation is a dual notion to forward simulation. A *backward simulation* over an LTS $\mathcal{T} = (\Sigma, Q, \delta)$ is a forward simulation over the LTS $\mathcal{T}^{-1} = (\Sigma, Q, \delta^{-1})$ where $\delta^{-1} = \{(p, a, q) \mid (q, a, p) \in \delta\}$.

Trees and Tree Automata. A *ranked alphabet* Σ is a set of symbols together with a function $\# : \Sigma \to \mathbb{N}$. For $a \in \Sigma$, the value $\#(a)$ is called the *rank* of a. For any $n \geq 0$, we denote by Σ_n the set of all symbols of rank n from Σ. Let ϵ denote the empty sequence. A *tree* t over a ranked alphabet Σ is a partial mapping $t : \mathbb{N}^* \to \Sigma$ that satisfies the following conditions:

- $dom(t)$ is a finite, prefix-closed subset of $\mathbb{N}^*$, and
- for each $p \in dom(t)$, $\#(t(p)) = n \geq 0$ iff $\{i \mid pi \in dom(t)\} = \{1, \ldots, n\}$.

Each sequence $v \in dom(t)$ is called a *node* of t. For a node v, we define the i^{th} *child* of p to be the node pi, and the i^{th} *subtree* of v to be the tree t' such that $t'(v') = t(viv')$ for all $p' \in \mathbb{N}^*$. A *leaf* of t is a node v which does not have any children, i.e., there is no $i \in \mathbb{N}$ with $vi \in dom(t)$. We denote by $T(\Sigma)$ the set of all trees over the alphabet Σ.

A (finite, non-deterministic, bottom-up) *tree automaton* (abbreviated as TA in the following) is a quadruple $\mathcal{A} = (\Sigma, Q, \Delta, F)$ where Q is a finite set of states, $F \subseteq Q$ is a set of final states, Σ is a ranked alphabet, and Δ is a set of transition rules. Each transition rule is a triple of the form $((q_1, \ldots, q_n), a, q)$ where $q_1, \ldots, q_n, q \in Q$, $a \in \Sigma$, and $\#(a) = n$. We use $(q_1, \ldots, q_n) \xrightarrow{a} q$ to denote that that $((q_1, \ldots, q_n), a, q) \in \Delta$. When using this notation, states $q_1, \ldots, q_n, q \in Q$ and symbol $a \in \Sigma$ are often considered to be implicitly existentially quantified. In the special case where $n = 0$, we speak about the so-called *leaf rules*, which we abbreviate as $\xrightarrow{a} q$. Finally, the *rank* of $\mathcal{A}$ denoted by $\hat{r}$ is defined as the greatest $n \in \mathbb{N}$ such that $(q_1, \ldots, q_n) \xrightarrow{a} q$.

A *run* of $\mathcal{A}$ over a tree $t \in T(\Sigma)$ is a mapping $\pi : dom(t) \to Q$ such that, for each node $p \in dom(t)$ where $q = \pi(p)$, if $q_i = \pi(pi)$ for $1 \leq i \leq n$, then Δ contains a rule $(q_1, \ldots, q_n) \xrightarrow{t(p)} q$. We write $t \xRightarrow{\pi} q$ to denote that π is a run of $\mathcal{A}$ over t such that $\pi(\epsilon) = q$. We use $t \Longrightarrow q$ to denote that $t \xRightarrow{\pi} q$ for some run π. The *language accepted at a state* q is defined by $L(q) = \{t \mid t \Longrightarrow q\}$, while the *language* of $\mathcal{A}$ is defined by $L(\mathcal{A}) = \bigcup_{q \in F} L(q)$.

Regular Tree Model Checking. Regular tree model checking (RTMC) [20, 9, 4, 6] is a general and uniform framework for verifying infinite-state systems. In RTMC, configurations of a system being verified are encoded by trees, sets of the configurations by tree automata, and transitions of the verified system by a term rewriting system (usually given as a tree transducer or a set of tree transducers). Then, verification problems based on performing reachability analysis correspond to computing closures of regular languages under rewriting systems, i.e., given a term rewriting system τ and a regular tree language I, one needs to compute $\tau^*(I)$ where τ^* is the reflexive-transitive

closure of τ. This computation is impossible in general. Therefore, the main issue in RTMC is to find accurate and powerful fixpoint acceleration techniques helping the convergence of computing language closures. One of the most successful acceleration techniques used in RTMC is abstraction whose use leads to the so-called *abstract regular tree model checking* (ARTMC) [6], on which we concentrate in this work.

Abstract Regular Tree Model Checking. We briefly recall the basic principles of ARTMC in the way they were introduced in [6]. Let Σ be a ranked alphabet and $\mathbb{M}_\Sigma$ the set of all tree automata over Σ. Let $\mathcal{I} \in \mathbb{M}_\Sigma$ be a tree automaton describing a set of initial configurations, τ a term rewriting system describing the behaviour of a system, and $\mathcal{B} \in \mathbb{M}_\Sigma$ a tree automaton describing a set of bad configurations. The safety verification problem can now be formulated as checking whether the following holds:

$$\tau^*(\mathcal{L}(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) = \emptyset \tag{1}$$

In ARTMC, the precise set of reachable configurations $\tau^*(\mathcal{L}(\mathcal{I}))$ is not computed to solve Problem (1). Instead, its overapproximation is computed by interleaving the application of τ and the union in $\mathcal{L}(\mathcal{I}) \cup \tau(\mathcal{L}(\mathcal{I})) \cup \tau(\tau(\mathcal{L}(\mathcal{I}))) \cup \ldots$ with an application of an abstraction function α. The abstraction is applied on the tree automata encoding the so-far computed sets of reachable configurations.

An abstraction function is defined as a mapping $\alpha : \mathbb{M}_\Sigma \to \mathbb{A}_\Sigma$ where $\mathbb{A}_\Sigma \subseteq \mathbb{M}_\Sigma$ and $\forall \mathcal{A} \in \mathbb{M}_\Sigma : \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\alpha(\mathcal{A}))$. An abstraction α' is called a *refinement* of the abstraction α if $\forall \mathcal{A} \in \mathbb{M}_\Sigma : \mathcal{L}(\alpha'(\mathcal{A})) \subseteq \mathcal{L}(\alpha(\mathcal{A}))$. Given a term rewriting system τ and an abstraction α, a mapping $\tau_\alpha : \mathbb{M}_\Sigma \to \mathbb{M}_\Sigma$ is defined as $\forall \mathcal{A} \in \mathbb{M}_\Sigma : \tau_\alpha(\mathcal{A}) = \hat{\tau}(\alpha(\mathcal{A}))$ where $\hat{\tau}(\mathcal{A})$ is the minimal deterministic automaton describing the language $\tau(\mathcal{L}(\mathcal{A}))$. An abstraction α is *finitary*, if the set $\mathbb{A}_\Sigma$ is finite.

For a given abstraction function α, one can compute iteratively the sequence of automata $(\tau_\alpha^i(\mathcal{I}))_{i \geq 0}$. If the abstraction α is finitary, then there exists $k \geq 0$ such that $\tau_\alpha^{k+1}(\mathcal{I}) = \tau_\alpha^k(\mathcal{I})$. The definition of the abstraction function α implies that $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \supseteq \tau^*(\mathcal{L}(\mathcal{I}))$.

If $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$, then Problem (1) has a positive answer. If the intersection is non-empty, one must check whether a real or a spurious counterexample has been encountered. The spurious counterexample may be caused by the used abstraction (the counterexample is not reachable from the set of initial configurations). Assume that $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$, which means that there is a symbolic path:

$$\mathcal{I},\ \tau_\alpha(\mathcal{I}),\ \tau_\alpha^2(\mathcal{I}), \ldots, \tau_\alpha^{n-1}(\mathcal{I}),\ \tau_\alpha^n(\mathcal{I}) \tag{2}$$

such that $\mathcal{L}(\tau_\alpha^n(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$.

Let $X_n = \mathcal{L}(\tau_\alpha^n(\mathcal{I})) \cap \mathcal{L}(\mathcal{B})$. Now, for each l, $0 \leq l < n$, $X_l = \mathcal{L}(\tau_\alpha^l(\mathcal{I})) \cap \tau^{-1}(X_{l+1})$ is computed. Two possibilities may occur: (a) $X_0 \neq \emptyset$, which means that Problem (1) has a negative answer, and $X_0 \subseteq \mathcal{L}(\mathcal{I})$ is a set of dangerous initial configurations. (b) $\exists m, 0 \leq m < n, X_{m+1} \neq \emptyset \wedge X_m = \emptyset$ meaning that the abstraction function is too rough—one needs to refine it and start the verification process again.

In [6], two general-purpose kinds of abstractions are proposed. Both are based on *automata state equivalences*. Tree automata states are split into several equivalence classes, and all states from one class are collapsed into one state. An abstraction becomes finitary if the number of equivalence classes is finite. The refinement is done by refining the equivalence classes. Both of the proposed abstractions allow for an automatic refinement to exclude the encountered spurious counterexample.

The first proposed abstraction is an *abstraction based on languages of trees of a finite height*. It defines two states equivalent if their languages up to the give height n are equivalent. There is just a finite number of languages of height n, therefore this abstraction is finitary. A refinement is done by an increase of the height n. The second proposed abstraction is an *abstraction based on predicate languages*. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of *predicates*. Each predicate $P \in \mathcal{P}$ is a tree language represented by a tree automaton. Let $\mathcal{A} = (Q, \Sigma, F, q_0, \delta)$ be a tree automaton. Then, two states $q_1, q_2 \in Q$ are equivalent if the languages $\mathcal{L}(\mathcal{A}_{q_1})$ and $\mathcal{L}(\mathcal{A}_{q_2})$ have a nonempty intersection with exactly the same subset of predicates from the set $\mathcal{P}$ provided that $\mathcal{A}_{q_1} = (Q, \Sigma, F, q_1, \delta)$ and $\mathcal{A}_{q_2} = (Q, \Sigma, F, q_2, \delta)$. Since there is just a finite number of subsets of $\mathcal{P}$, the abstraction is finitary. A refinement is done by adding new predicates, i.e. tree automata corresponding to the languages of all the states in the automaton of X_{m+1} from the analysis of spurious counterexample ($X_m = \emptyset$).

3. Tree Automata Reduction Methods

Our tree automata reduction methods are build on the notions of downward and upward tree automata simulations (proposed first in [4]) that are the tree automata counterparts the forward and backward FA simulations. Let us fix a tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$. The tree automata simulations are defined as follows.

Downward Simulation. A *downward simulation* D on $\mathcal{A}$ is a binary relation on Q where if qDr and $(q_1, \ldots, q_n) \xrightarrow{a} q$, then $(r_1, \ldots, r_n) \xrightarrow{a} r$ with $q_i D r_i$ for each $i : 1 \leq i \leq n$.

Upward Simulation. Given a preorder D on Q, an *upward simulation U induced by D* is a binary relation on Q such that if qUr, then

1. if $(q_1, \ldots, q_n) \xrightarrow{a} q'$ with $q_i = q$, $1 \leq i \leq n$, then $(r_1, \ldots, r_n) \xrightarrow{a} r'$ with $r_i = r$, $q' U r'$, and $q_j D r_j$ for each $j : 1 \leq j \neq i \leq n$;
2. $q \in F \implies r \in F$.

There is a unique maximal downward simulation D on $\mathcal{A}$ which is a preorder. We call it *the downward simulation preorder on $\mathcal{A}$* and we call the equivalence $D \cap D^{-1}$ *the downward simulation equivalence on $\mathcal{A}$*. Analogically, for a preorder D on Q, there is a unique maximal upward simulation U induced by D which is a preorder. We call it *the upward simulation preorder on $\mathcal{A}$ induced by D* and we call the equivalence $U \cap U^{-1}$ *the upward simulation equivalence on $\mathcal{A}$ induced by D*.

Mediation. Downward simulation can be shown to be compatible with the tree language equivalence. Upward

simulation is not compatible with the tree language equivalence. It is rather compatible with the so-called context language equivalence, where a context of a state q is a tree with a hole on the leaf level such that if we plug a tree in the tree language of q into this hole, we obtain a tree recognised by the automaton. Unlike downward simulation, upward simulation alone cannot be used for reducing tree automata. However, we have identified a principle of combining upward and downward TA simulations simulations that yields an equivalence, called mediated equivalence, suitable for reducing automata by collapsing their states while preserving the language. Mediated equivalence is coarser than downward simulation equivalence and thus gives a better reduction.

For the sake of simplicity, we will explain the combination principle on FA and forward and backward simulation, for which it is also relevant. Mediated minimisation of FA generalises the principle of forward simulation minimisation. Two forward simulation equivalent states can be safely collapsed since they have the same forward languages (symmetrically for backward simulation). In contrary, the property that allow collapsing two mediated equivalent states p and q is the following. Whenever there is a computation under a word u starting in an initial state that ends in a state p, and another computation under a word v starting in a state q and ending in a final state, then there is a computation under uv from an initial to a final state. Therefore, collapsing the two states p, q does not introduce any new behaviour since every word accepted via the new state was accepted also before collapsing.

The case of TA mediated equivalence can be explained analogically. It may be seen from the above that unlike simulations, mediated equivalences approximate neither forward nor backward language equivalence on states, and similarly the tree automata mediated equivalence is not compatible with any notion of language of a state of a tree automaton.

The combination of a downward and an upward simulation (or forward and backward simulation in the case of FA) yielding a mediated equivalence is defined using the following combination operator.

Combination Operator. The *relation combination operator* $\oplus$ is defined such that given two preorders D and U over a set Q, for $x, y \in Q$, $x(D \oplus U)y$ iff (i) $x(D \circ U)y$ and (ii) $\forall z \in Q : yDz \implies x(D \circ U)z$.

Consider a reflexive and transitive downward simulation D on $\mathcal{A}$, and a reflexive and transitive upward simulation U induced by D. We call the relation $M = D \oplus U^{-1}$ a *mediated preorder induced by D and U* and $\equiv_M = M \cap M^{-1}$ a *mediated equivalence induced by D and U*.

We aim at reducing tree automata by quotienting wrt. to mediated equivalence. Informally, quotienting wrt. an equivalence relation means merging equivalent states of an automaton. We define this operation formally as follows.

Quotient Tree Automata. Consider an equivalence relation $\equiv$ on Q. We denote $[q]$ the equivalence class of $\equiv$ containing q. The *quotient* of $\mathcal{A}$ according to $\equiv$ is the tree automaton $\mathcal{A}/{\equiv} = (\Sigma, Q/{\equiv}, \Delta/{\equiv}, \{[q] \mid q \in F\})$ where $\Delta/{\equiv} = \{(([q_1], \ldots, [q_n]), a, [q]) \mid ((q_1, \ldots, q_n), a, q) \in \Delta\}$.

We show that quotienting $\mathcal{A}$ with respect to $\equiv_M$ preserves the language, as stated by the below theorem.

THEOREM 1. $L(\mathcal{A}/{\equiv_M}) = L(\mathcal{A})$.

4. Language Inclusion of Tree Automata

Let $\mathcal{A} = (Q, \Sigma, F, \Delta)$ and $\mathcal{B} = (Q', \Sigma, F', \Delta')$ be two tree automata. We want to check if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. The traditional approach computes the complement of $\mathcal{B}$ and checks if it has an empty intersection with $\mathcal{A}$. This is costly as computing the complement necessitates determinisation. Here we show how to check inclusion without determinisation.

The idea is to find at least one tree accepted by $\mathcal{A}$ and not by $\mathcal{B}$. For that, we simultaneously simulate the runs of the two automata using pairs (p, s) with $p \in Q$ and $s \subseteq Q'$ where p memorises the run of $\mathcal{A}$ and s all the possible runs of $\mathcal{B}$. If t is a tree accepted by $\mathcal{A}$ and not by $\mathcal{B}$, the simultaneous run of the two automata on t reaches the root of t at a pair of the form (p, s) with $p \in F$ and $s \subseteq \overline{F'}$. Notice that s must represent *all* the possible runs of $\mathcal{B}$ on t to make sure that no run of $\mathcal{B}$ can accept the tree t. Therefore, s must be a set of states.

Formally, an *antichain* over $Q \times 2^{Q'}$ is a set $\mathcal{S} \subseteq Q \times 2^{Q'}$ such that for every $(p, s), (p', s') \in \mathcal{S}$, if $p = p'$, then $s \not\subset s'$. We denote by L_I the set of all antichains over $Q \times 2^{Q'}$. Given a set $\mathcal{S} \in Q \times 2^{Q'}$, an element $(p, s) \in \mathcal{S}$ is *minimal* if for every $s' \subset s$, $(p, s') \notin \mathcal{S}$. We denote by $\lfloor \mathcal{S} \rfloor$ the set of minimal elements of $\mathcal{S}$. Given two antichains $\mathcal{S}$ and $\mathcal{S}'$, we define the order $\sqsubseteq_I$, the least upper bound $\sqcup_I$, and the greatest lower bound $\sqcap_I$ as follows: $\mathcal{S} \sqsubseteq_I \mathcal{S}'$ iff for every $(p, s') \in \mathcal{S}'$, there is $(p, s) \in \mathcal{S}$ s.t. $s \subseteq s'$; $\mathcal{S} \sqcup_I \mathcal{S}' = \lfloor \{(p, s \cup s') \mid (p, s) \in \mathcal{S} \wedge (p, s') \in \mathcal{S}'\} \rfloor$; and $\mathcal{S} \sqcap_I \mathcal{S}' = \lfloor \{(p, s) \mid (p, s) \in \mathcal{S} \vee (p, s) \in \mathcal{S}'\} \rfloor$. These definitions can be extended to arbitrary sets in the usual way leading to the operators $\bigsqcup_I$ and $\bigsqcap_I$, yielding a complete lattice.

Given $f \in \Sigma_n, n \geq 0$, we define

$$IPost_f\big((p_1, s_1), \ldots, (p_n, s_n)\big) = \{(p, s) \,|\, f(p_1, \ldots, p_n) \to_\Delta p \wedge s = Post_f^{\Delta'}(s_1, \ldots, s_n)\}.$$

Let $\mathcal{S}$ be an antichain over $Q \times 2^{Q'}$. Then, let

$$IPost(\mathcal{S}) = \lfloor \{IPost_f\big((p_1, s_1), \ldots, (p_n, s_n)\big) \mid n \geq 0, (p_1, s_1), \ldots, (p_n, s_n) \in \mathcal{S}, f \in \Sigma_n\} \rfloor.$$

Let $IPost_0(\mathcal{S}) = \mathcal{S}$ and let

$$IPost_i(\mathcal{S}) = IPost\big(IPost_{i-1}(\mathcal{S})\big) \sqcap_I \mathcal{S}.$$

We can show that $\forall S \in L_I \ \forall i \geq 0 : IPost_{i+1}(S) \sqsubseteq_I IPost_i(S)$, and that for every antichain $\mathcal{S}$, there exists a J such that $IPost_{J+1}(\mathcal{S}) = IPost_J(\mathcal{S})$. Let $IPost^*(\mathcal{S}) = IPost_J(\mathcal{S})$. Note that $IPost_a(\emptyset) = \{(q, Post_a^{\Delta'}(\emptyset)) \mid a \to_\Delta q\}$ for $a \in \Sigma_0$, and $IPost_f(\emptyset) = \emptyset$ for $f \in \Sigma_n$, $n \geq 1$. Then, we can also the following two lemmas.

LEMMA 4.1. *Let $\mathcal{A} = (Q, \Sigma, F, \Delta)$, $\mathcal{B} = (Q', \Sigma, F', \Delta')$ be two tree automata, and let t be a tree over Σ. Let $p \in Q$*

such that $t \xrightarrow{*}_{\Delta} p$, *and* $s = \{q \in Q' \mid t \xrightarrow{*}_{\Delta'} q\}$. *Then,* $IPost^*(\emptyset) \sqsubseteq_I \{(p, s)\}$.

LEMMA 4.2. *Let* $\mathcal{A} = (Q, \Sigma, F, \Delta)$, $\mathcal{B} = (Q', \Sigma, F', \Delta')$ *be two tree automata, and let* $(p, s) \in IPost^*(\emptyset)$. *Then there is a tree* t *over* Σ *s.t.* $t \xrightarrow{*}_{\Delta} p$ *and* $s = \{q \mid t \xrightarrow{*}_{\Delta'} q\}$.

Then, we can decide inclusion *without determinising the automata* as follows:

THEOREM 2. *Let* $\mathcal{A} = (Q, \Sigma, F, \Delta)$, $\mathcal{B} = (Q', \Sigma, F', \Delta')$ *be two tree automata. Then,* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ *iff for every* $(p, s) \in IPost^*(\emptyset)$, $p \in F \Rightarrow s \not\subseteq F'$.

We note that this algorithm can be further improved as presented in [1] by interconnecting the antichain principle with a use of simulations over states of the two automata.

5. Abstract Regular Tree Model Checking with Nondeterministic Automata

As is clear from the definition of $\hat{\tau}$ in Section 2, ARTMC was originally defined for and tested on *minimal deterministic* tree automata (DTA). However, the various experiments done showed that the determinisation step is a significant bottleneck. To avoid it and to implement ARTMC using nondeterministic tree automata (TA), we need the following operations over TA: (1) application of the transition relation τ, (2) union, (3) abstraction and its refinement, (4) intersection with the set of bad configurations, (5) emptiness, and (6) inclusion checking (needed for testing if the abstract reachability computation has reached a fixpoint). Finally, (7) a method to reduce the size of the computed TA is also desirable—$\hat{\tau}(\mathcal{A})$ is then redefined to be the reduced version of the TA obtained from an application of τ on an TA $\mathcal{A}$. We note that the method would in theory work without reduction methods too. However, often hundreds of the steps (1) to (6) are performed within a single verification run, and most of them increases the size of automata[1]. Therefore, good reduction techniques are in fact crucial since the size of automata tends to explode which reduces scalability of the method.

An implementation of Points (1), (2), (4), and (5) is easy. Moreover, concerning Point (3), the abstraction mechanisms of [6] can be lifted to work on nondeterministic TA in a straightforward way while preserving their guarantees to be finitary, overapproximating, and the ability to exclude spurious counterexamples. Furthermore, Section 3 gives efficient algorithms for reducing TA based on computing suitable simulation equivalences on their states, which covers Point (7). Hence, the last obstacle for implementing nondeterministic ARTMC was Point (6), i.e., the need to efficiently check inclusion on TA. We have solved this problem by our algorithm presented in Section 4, which allowed us to implement a nondeterministic ARTMC framework in a prototype tool and test it on suitable examples. Below, we present the first very encouraging results that we have achieved.

[1]Some abstraction methods reduce the size of automata too, however, not sufficiently enough to outweigh the increase of size caused by the other steps.

Experiments with Nondeterministic ARTMC.
We have implemented the version of ARTMC framework based on nondeterministic tree automata using the Timbuk tree automata library [12] and compared it with an ARTMC implementation based on the same library, but using DTA. In particular, the deterministic ARTMC framework uses determinisation and minimisation after computing the effect of each forward or backward step to try to keep the automata as small as possible and to allow for easy fixpoint checking: The fixpoint checking on DTA is not based on inclusion, but identity checking on the obtained automata (due to the fact that the computed sets are only growing and minimal DTA are canonical). For TA, the tree automata reduction from Section 3 that we use does not yield canonical automata, and so the antichain-based inclusion checking is really needed.

We have applied the framework to verify several procedures manipulating dynamic tree-shaped data structures linked by pointers. The trees being manipulated are encoded directly as the trees handled in ARTMC, each node is labelled by the data stored in it and the pointer variables currently pointing to it. All program statements are encoded as (possibly non-structure preserving) tree transducers. The encoding is fully automated. The only allowed destructive pointer updates (i.e., pointer manipulating statements changing the shape of the tree) are tree rotations [10] and addition of new leaf nodes.

We have in particular considered verification of the depth-first tree traversal and the standard procedures for rebalancing red-black trees after insertion or deletion of a leaf node [10]. We have verified that the programs do not manipulate undefined and null pointers in a faulty way. For the procedures on red-black trees, we have also verified that their result is a red-black tree (without taking into account the non-regular balancedness condition). In general, the set of possible input trees for the verified procedures as well as the set of correct output trees were given as tree automata. In the case of the procedure for rebalancing red-black trees after an insertion, we have also used a generator program preceding the tested procedure which generates random red-black trees and a tester program which tests the output trees being correct. Here, the set of input trees contained just an empty tree, and the verification was reduced to checking that a predefined error location is unreachable. The size of the programs ranges from 10 to about 100 lines of pure pointer manipulations.

The results of our experiments on an Intel Xeon processor at 2.7GHz with 16GB of available memory are summarised in Table 1. The predicate abstraction proved to give much better results (therefore we do not consider the finite-height abstraction here). The abstraction was either applied after firing each statement of the program ("full abstraction") or just when reaching a loop point in the program ("restricted abstraction"). The results we have obtained are very encouraging and show a significant improvement in the efficiency of ARTMC based on nondeterministic tree automata. Indeed, the ARTMC framework based on deterministic tree automata has either been significantly slower in the experiments (up to 25-times) or has completely failed (a too long running time or a lack of memory)—the latter case being quite frequent.

Table 1: Running times (in sec.) of det. and nondet. ARTMC applied for verification of various tree manipulating programs (× denotes a too long run or a failure due to a lack of memory)

	DFT		RB-delete (null,undef)		RB-insert (null,undef)	
	det.	nondet.	det.	nondet.	det.	nondet.
full abstr.	5.2	2.7	×	×	33	15
restricted abstr.	40	3.5	×	60	145	5.4

	RB-delete (RB preservation)		RB-insert (RB preservation)		RB-insert (gen., test.)	
	det.	nondet.	det.	nondet.	det.	nondet.
full abstr.	×	×	×	×	×	×
restricted abstr.	×	57	×	89	×	978

6. Conclusions and Future Directions

We have designed efficient methods for reducing size of tree automata based on tree automata simulations and algorithm for checking language inclusion of tree automata. Based on these techniques, we implemented a version of abstract regular tree model checking method on top of nondeterministic tree automata instead of on deterministic ones. This greatly improved efficiency and scalability of the method as confirmed by our experimental results.

There is a number of interesting directions of further work. The mediation principle itself can be further elaborated. We already have some preliminary results suggesting that it is possible to define a hierarchy of coarser and coarser relations similar to the mediated equivalence (and suitable for quotienting automata), where a mediated relation of level i is used to induce a mediated relation of level $i + 1$. We are also working towards applying our methods in practice. We are developing an efficient BDD based library that would provide procedures for handling nondeterministic tree automata (in the style of [17]). This work includes also a development of BDD versions of our algorithms, which is itself an interesting problem. We are also working on an ARTMC based method for verification of pointer manipulating programs that will make use of our TA reduction and language inclusion checking techniques.

References

[1] P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When Simulation Meets Antichains (on Checking Language Inclusion of NFAs). In *TACAS'10*, volume 6015 of *LNCS*, pages 158–174. Springer, 2010.

[2] P. A. Abdulla, J. Högberg, and L. Kaati. Bisimulation Minimization of Tree Automata. *Int. J. Found. Comput. Sci.*, 18(4):699–713, 2007.

[3] P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular Tree Model Checking. In *CAV'02*, volume 2404 of *LNCS*, pages 555–568. Springer, 2002.

[4] P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Simulation-Based Iteration of Tree Transducers. In *TACAS*, volume 3440 of *LNCS*, pages 30–44. Springer, 2005.

[5] A. Bouajjani, P. Habermehl, L. Holík, T. Touili, and T. Vojnar. Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. In *CIAA'08*, volume 5148 of *LNCS*, pages 57–67. Springer, 2008.

[6] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular Tree Model Checking. *Electr. Notes Theor. Comput. Sci.*, 149(1):37–48, 2006.

[7] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *SAS'06*, pages 52–70, 2006.

[8] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *CAV'04*, volume 3114 of *LNCS*, pages 372–386. Springer, 2004.

[9] A. Bouajjani and T. Touili. Extrapolating Tree Transformations. In *CAV'02*, volume 2404 of *LNCS*, pages 539–554. Springer, 2002.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[11] S. Crafa, F. Ranzato, and F. Tapparo. Saving Space in a Time Efficient Simulation Algorithm. In *ACSD'09*, pages 60–69. IEEE, 2009.

[12] T. Genet, V. Viet, and T. Tong. Timbuk: A Tree Automata Library. `http://www.irisa.fr/lande/genet/timbuk`, 2003.

[13] R. Gentilini, C. Piazza, and A. Policriti. From Bisimulation to Simulation: Coarsest Partition Problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[14] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *FOCS'95*, pages 453–462, Washington, DC, USA, 1995. IEEE.

[15] J. Högberg, A. Maletti, and J. May. Backward and Forward Bisimulation Minimisation of Tree Automata. In *CIAA'07*, volume 4783 of *LNCS*, pages 109–121. Springer, 2007.

[16] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.

[17] N. Klarlund and A. Møller. MONA Version 1.4 User Manual, 2001. BRICS, Department of Computer Science, University of Aarhus, Denmark.

[18] R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[19] F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *LICS'07*, pages 171–180. IEEE, 2007.

[20] E. Shahar. *Tools and Techniques for Verifying Parameterized Systems*. PhD thesis, Faculty of Mathematics and Computer Science, The Weizmann Inst. of Science, Rehovot, Israel, 2001.

[21] A. Valmari. Bisimilarity Minimization in $\mathcal{O}(m \log n)$ Time. In *Petri Nets*, volume 5606 of *LNCS*, pages 123–142. Springer, 2009.

[22] M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A New Algorithm for Checking Universality of Finite Automata. In *CAV'06*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.

Selected Papers by the Author

A. Bouajjani, P. Habermehl, L. Holík, T. Touili, and Tomáš Vojnar. Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. In *CIAA'08*, volume 5148 of *LNCS*, pages 57–67. Springer, 2008.

P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Computing Simulations over Tree Automata: Efficient Techniques for Reducing Tree Automata. In *TACAS'08*, volume 4963 of *LNCS*, pages 93–108. Springer, 2008.

P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Composed Bisimulation for Tree Automata. *Int. J. Found. Comput. Sci.*, 20(4):685–700, 2009.

P. A. Abdulla, Y.-F. Chen, L. Holík, and T. Vojnar. Mediating for Reduction (on Minimizing Alternating Büchi Automata). In *FSTTCS'09*, volume 4 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

P. A. Abdulla, L. Holík, L. Kaati, and T. Vojnar. A Uniform (Bi-)Simulation-Based Framework for Reducing Tree Automata. *Electr. Notes Theor. Comput. Sci.*, 251:27–48, 2009.

P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When Simulation Meets Antichains (on Checking Language Inclusion of NFAs). In *TACAS'10*, volume 6015 of *LNCS*, pages 158–174. Springer, 2010.

P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, Chih-Duo Hong, Richard Mayr, and T. Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *CAV'10*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.

L. Holík and A. Rogalewicz. Counterexample Analysis in Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *MEMICS'07*, pages 59–66, 2007.

L. Holík and J. Šimáček. Optimizing an LTS-Simulation Algorithm. In *MEMICS'09*, pages 93–101. Faculty of Informatics MU, 2009. An extended version accepted at Computing and Informatics.