Minimization of Nondeterministic Tree Automata

Ricardo Almeida[1], Lukáš Holík[2], and Richard Mayr[1]

1 **University of Edinburgh, UK**
2 **Brno University of Technology, Czech Republic**

Abstract

We present an efficient algorithm to reduce the size of nondeterministic tree automata, while retaining their language. It is based on new transition pruning techniques, and quotienting of the state space w.r.t. suitable equivalences. It uses criteria based on combinations of downward and upward simulation preorder on trees, and the more general downward and upward language inclusions. Since tree-language inclusion is EXPTIME-complete, we describe methods to compute good approximations in polynomial time.

We implemented our algorithm as a module of the well-known libvata tree automaton library, and tested its performance on a given collection of tree automata from various applications of libvata in regular model checking and shape analysis, as well as on various classes of randomly generated tree automata. Our algorithm consistently outperforms all previous techniques by a wide margin, yielding substantially smaller and sparser automata in most instances.

1998 ACM Subject Classification F.1.1 [Models of Computation]: Automata; D.2.4 [Software Verification]: Model checking

Keywords and phrases Tree automata, Minimization

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Background. Tree automata are a generalization of word automata that accept trees instead of words [12]. They have many applications in model checking [5, 4, 10], term rewriting [13], and related areas of formal software verification, e.g., shape analysis [3, 18, 16]. Several software packages for manipulating tree automata have been developed, e.g., MONA [7], Timbuk [14], Autowrite [13] and libvata [20], on which other verification tools like Forester [21] are based.

For nondeterministic automata, many questions about their languages are computationally hard. The language universality, equivalence and inclusion problems are PSPACE-complete for word automata and EXPTIME-complete for tree automata [12]. However, recently techniques have been developed that can solve many practical instances fairly efficiently. For word automata there are antichain techniques [2], congruence-based techniques [8] and techniques based on generalized simulation preorders [11]. The antichain techniques have been generalized to tree automata in [9, 19] and implemented in the libvata library [20]. Performance problems also arise in computing the intersection of several languages, since the product construction multiplies the numbers of states.

Automata Minimization. Our goal is to make tree automata more computationally tractable in practice. We present an efficient algorithm for the minimization of nondeterministic tree automata, in the sense of obtaining a smaller automaton with the same language, though not necessarily with the absolute minimal possible number of states. (In general, the minimal nondeterministic automaton for a language is not even unique.) The reason to perform minimization is that the smaller minimized automaton is more efficient to handle in a subsequent computation. Thus there is an algorithmic tradeoff between the effort for minimization and the complexity of the problem later considered for this automaton. The main applications of minimization are the following: (1) Helping to solve hard problems like language universality/equivalence/inclusion. (2) If automata undergo a long chain of

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–22

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

manipulations/combinations by operations like union, intersection, projection, etc., then intermediate results can be minimized several times on the way to keep the automata within a manageable size. (3) There are fixed-parameter tractable problems (e.g., in model checking where an automaton encodes a logic formula) where the size of one automaton very strongly influences the overall complexity, and must be kept as small as possible.

Our contribution. We present a minimization algorithm for nondeterministic tree automata. (The tool is available for download [6].) It is based on a combination of new transition pruning techniques for tree automata, and quotienting of the state space w.r.t. suitable equivalences. The pruning techniques are related to those presented for word automata in [11], but significantly more complex due to the fundamental asymmetry between the upward and downward directions in trees.

Transition pruning in word automata [11] is based on the observation that certain transitions can be removed without changing the language, because other 'better' transitions remain. One defines some strict partial order (p.o.) between transitions and removes all transitions that are not maximal w.r.t. this order. A strict p.o. between transitions is called *good for pruning* (GFP) iff pruning w.r.t. it preserves the language of the automaton. Note that pruning reduces not only the number of transitions, but also (indirectly) the number of states, because it may make some states unreachable, which can then be removed. One can obtain computable strict p.o. between transitions by comparing the possible backward- and forward behavior of their source- and target states, respectively. For this, one uses computable relations like backward/forward simulation preorder and approximations of backward/forward trace inclusion via lookahead- or multipebble simulations. Some such combinations of backward/forward trace/simulation orders on states induce strict p.o. between transitions that are GFP, while others do not [11]. However, there is always a symmetry between backward and forward, since finite words can equally well be read in either direction.

This symmetry does not hold for tree automata, because the tree branches as one goes downward, while it might 'join in' side branches as one goes upward. While downward simulation preorder (resp. downward language inclusion) between states in a tree automaton is a direct generalization of forward simulation preorder (resp. forward language inclusion) on words, the corresponding upward notions do not correspond to backward on words. Comparing upward behavior of states in tree automata depends also on the branches that 'join in' from the sides as one goes upward in the tree. Thus upward simulation/language inclusion is only defined *relative* to a given other relation that compares the downward behavior of states 'joining in' from the sides [1]. So one speaks of "upward simulation *of* the identity relation" or "upward simulation *of* downward simulation". When one studies strict p.o. between transitions in tree automata in order to check whether they are GFP, one has combinations of three relations: the source states are compared by an upward relation $X(Y)$ of some downward relation Y, while the target states are compared w.r.t. some downward relation Z (where Z can be, and often must be, different from Y). This yields a richer landscape, and many counter-intuitive effects.

We provide a complete picture of which combinations of upward/downward simulation/trace inclusions are GFP on tree automata; cf. Figure 4. Since tree-language inclusion is `EXPTIME`-complete, we describe methods to compute good approximations of them in polynomial time, by generalizing lookahead simulations [11] to trees. Finally, we also generalize results on quotienting of tree automata [17] to larger relations, such as approximations of trace inclusion.

We implemented our algorithm [6] as a module of the well-known `libvata` [20] tree automaton library, and tested its performance on a given collection of tree automata from various applications of `libvata` in regular model checking and shape analysis, as well as on various classes of randomly generated tree automata. Our algorithm consistently outperforms all previous techniques by a wide margin, yielding substantially smaller automata in most instances. Moreover, the thus obtained automata are also much sparser (i.e., use fewer transitions per state and less nondeterministic branching) than the originals, which yields additional performance advantages in subsequent computations.

2 Trees and Tree Automata

Trees. A *ranked alphabet* Σ is a set of symbols together with a function $\# : \Sigma \to \mathbb{N}_0$. For $a \in \Sigma$, $\#(a)$ is called the *rank* of a. For $n \geq 0$, we denote by Σ_n the set of all symbols of Σ which have rank n.

We define a *node* as a sequence of elements of $\mathbb{N}$, where ε is the empty sequence. For a node $v \in \mathbb{N}^*$, we define the i-th child of v to be the node vi, for some $i \in \mathbb{N}$. Given a ranked alphabet Σ, a *tree* over Σ is defined as a partial mapping $t : \mathbb{N}^* \to \Sigma$ such that for all $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, if $vi \in dom(t)$ then **(1)** $v \in dom(t)$, and **(2)** $\#(t(v)) \geq i$. In this paper we consider only finite trees. To maintain the parallel with the word automata case, we also consider the existence of the *empty tree*, denoted by ϵ.

Note that the number of children of a node v may be smaller than $\#(t(v))$. In this case we say that the node is *open*. Nodes which have exactly $\#(t(v))$ children are called *closed*. Nodes which do not have any children are called *leaves*. A tree is closed if all its nodes are closed, otherwise it is open. By $\mathbb{C}(\Sigma)$ we denote the set of all closed trees over Σ and by $\mathbb{T}(\Sigma)$ the set of all trees over Σ. A tree t is *linear* iff every node in $dom(t)$ has at most one child.

The *subtree* of a tree t at v is defined as the tree t_v such that $dom(t_v) = \{v' \mid vv' \in dom(t)\}$ and $t_v(v') = t(vv')$ for all $v' \in dom(t_v)$. A tree t' is a prefix of t iff $dom(t') \subseteq dom(t)$ and for all $v \in dom(t')$, $t'(v) = t(v)$. For $t \in \mathbb{C}(\Sigma)$, the *height of a node* v of t is given by the function h: if v is a leaf then $h(v) = 1$, otherwise $h(v) = 1 + max(h(v1)), \ldots, h(v\#(t(v))))$. We define the height of a tree $t \in \mathbb{C}(\Sigma)$ as $h(\varepsilon)$, i.e., as the number of levels of t. By definition, the empty tree ϵ has height 0.

Tree automata, bottom-up. A (finite, non-deterministic) *bottom-up tree automaton* (abbreviated as BUTA in the following) is a quadruple $A = (\Sigma, Q, \delta, F)$ where Q is a finite set of states, $F \subseteq Q$ is a set of final states, Σ is a ranked alphabet, and $\delta \subseteq Q^+ \times \Sigma \times Q$ is a set of transition rules. A BUTA has an unique initial state, which we represent by ψ. The transition rules satisfy that if $\langle \psi, a, q \rangle \in \delta$ then $\#(a) = 0$, and if $\langle q_1 \ldots q_n, a, q \rangle \in \delta$ (with $n > 0$) then $\#(a) = n$. In the related literature, it is common to consider that a BUTA does not have an explicit initial state defined. The transition function is defined as $\delta \subseteq Q^* \times \Sigma \times Q$ and the transition rule for the case of $n = 0$ (i.e., $\langle \psi, a, q \rangle$) is sometimes called an *initial rule* or a *leaf rule* [12, 9, 19]. However, to ease the connection between bottom-up and top-down automata we replace this rule by the special case $\langle \psi, a, q \rangle$.

A *run* of A over a tree $t \in \mathbb{T}(\Sigma)$ (or a t-run in A) is a partial mapping $\pi : \mathbb{N}^* \to Q$ such that $v \in dom(\pi)$ iff either $v \in dom(t)$ or $v = v'i$ where $v' \in dom(t)$ and $i \leq \#(t(v'))$. Further, for every $v \in dom(t)$, there exists either **a)** a rule $\langle \psi, a, q \rangle$ such that $q = \pi(v)$ and $a = t(v)$, or **b)** a rule $\langle q_1 \ldots q_n, a, q \rangle$ such that $q = \pi(v)$, $a = t(v)$, and $q_i = \pi(vi)$ for each $i : 1 \leq i \leq \#(a)$. A *leaf of a run* π on t is a node $v \in dom(\pi)$ such that $vi \in dom(\pi)$ for no $i \in \mathbb{N}$. We call it *dangling* if $v \notin dom(t)$. Intuitively, the dangling nodes of a run over t are all the nodes which are in π but are missing in t due to it being incomplete. Notice that dangling leaves of π are children of open nodes of t. The prefix of depth k of a run π is denoted π_k. Runs are always finite since the trees we are considering are finite.

We write $t \xrightarrow{\pi} q$ to denote that π is a t-run of A such that $\pi(\varepsilon) = q$. We use $t \Longrightarrow q$ to denote that such run π exists. A run π is accepting if $t \xrightarrow{\pi} q \in F$. The *downward language of a state* q in A is defined by $D_A(q) = \{t \in \mathbb{C}(\Sigma) \mid t \Longrightarrow q\}$, while the *language* of A is defined by $L(A) = \bigcup_{q \in F} D_A(q)$. The *upward language* of a state q in A, denoted $U_A(q)$, is then defined as the set of open trees t, such that there exists an accepting t-run π with exactly one dangling leaf v s.t. $\pi(v) = q$. We omit the A subscript notation when it is implicit which automaton we are considering.

Tree automata, top-down. In analogy to automata on words, which read a word from the beginning to the end (i.e., left-to-right), one may also define tree automata reading a tree from its root to the leaves (i.e., top-down). A top-down tree automaton (TDTA) is a quadruple (Σ, Q, δ, I) where $I \subseteq Q$ is the set of initial states and $\psi \in Q$ now denotes the (unique) final state. A TDTA can be obtained from a BUTA by swapping the roles between the initial states and the final states. The direction of the transition rules is reversed to reflect this change, so we let $\delta \subseteq Q \times \Sigma \times Q^+$. A run π is accepting

why a run is not itself a tree?

n>0 !!

it follows that t has exactly one open node...?

why not just introducing TDTA straight away?

if $t \xrightarrow{\pi} q \in I$. The language of a TDTA A is now given by $L(A) = \bigcup_{q \in I} D_A(q)$. See Appendix A for an example of a tree automaton presented in both BUTA and TDTA form.

In the rest of this paper we adopt TDTA as the standard representation of a tree automaton in order to make the connection to word automata more explicit. We will consider different types of relations on states of a TDTA which under-approximate language inclusion, where *downward* simulation/trace inclusion corresponds to *direct forward* simulation/trace inclusion in the words case, and *upward* corresponds to *backward* [11]. Note that words are but a special case of trees where every node has only one child, i.e., words are linear trees.

3 Trace Inclusion

We define downward and upward trace inclusions for TDTAs by extending previous definitions of forward and backward trace inclusions for word automata [11]. not formally defined

Downward trace inclusion. Let $A = (\Sigma, Q, I, F, \delta)$ be an automaton on words. For $p, q \in Q$, we say that *forward trace inclusion* holds and write $p \subseteq q$ (this is called *direct trace inclusion* and written $p \subseteq^{di} q$ in [11]), iff for every word $w = \sigma_0\sigma_1 \ldots \sigma_{n-1} \in \Sigma^*$, and for every w-trace $\pi_0 = p \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{n-1}} p_n$, there exists a w-trace $\pi_1 = q \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{n-1}} q_n$, such that $\forall_{i \geq 0}.\, p_i \in F \implies q_i \in F$. Since π_1 is required to preserve the acceptance of the states in π_0, trace inclusion is a strictly stronger notion than language inclusion (see Figure 7 in Appendix A for an example). One way of making downward language inclusion on the states of an automaton coincide with downward trace inclusion is by modifying the automaton to guarantee that **1)** there is one unique final state which has no outgoing transitions, **2)** from any other state, there is a path ending in that final state. Note that in a TDTA these two conditions are automatically satisfied: **1)** since the final state is reached after reading a leaf of the tree, and **2)** because only complete trees are in the language of the automaton. Thus, in a TDTA, downward language inclusion and downward trace inclusion $\subseteq^{dw}$ do coincide. this should be formally defined

this is because in tree languages leaf symbols can be accepted only from final states

Upward trace inclusion. Let $A = (\Sigma, Q, I, F, \delta)$ be an automaton on words. For $p, q \in Q$, we say that *backward trace inclusion* holds, and write $p \subseteq^{bw} q$, iff for every word $w = \sigma_0\sigma_1 \cdots \sigma_{m-1} \in \Sigma^*$, and for every w-trace $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} p_m$, with $p_m = p$, there exists a w-trace $\pi_1 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{m-1}} q_m$, with $q_m = q$, such that $\forall_{i \geq 0}.\, (p_i \in F \to q_i \in F) \wedge (p_i \in I \to q_i \in I)$ [11].

Unlike words, trees are branching as one goes downward, and correspondingly they are 'joining in' side branches as one goes upward. Thus any relation that compares the upward-behavior of states in tree automata is only meaningful *relative* to some previously defined relation R that compares (states in) these joining side branches. Given a TDTA $A = (\Sigma, Q, \delta, I)$, we say that a run π' of A *subsumes* a run π over the same tree w.r.t. a relation ~~s~~ R and a leaf v of π, denoted $\pi \sqsubseteq^v_R \pi'$, iff for all leaves v' of π except v, $\pi(v')\, R\, \pi'(v')$, and $\pi(\varepsilon) \in I \to \pi'(\varepsilon) \in I$. We say that *upward trace inclusion parameterized with relation R* holds, and we write $p \subseteq^{up}(R)\, q$, iff for every linear tree t and every t-run π with a bottom leaf v such that $\pi(v) = p$, there exists a t-run π' such that $\pi'(v) = q$ and the following two conditions hold:

1. $\pi \sqsubseteq^v_R \pi'$, and
2. $\forall_{n \in \mathbb{N}^*}.\, (\pi(n) = \psi \to \pi'(n) = \psi)$
   in dom(π)

Downward trace inclusion is EXPTIME-complete for trees [12] (and PSPACE-complete for words), while the complexity of upward trace inclusion depends on the relation R (e.g., PSPACE-complete for $R = id$). In contrast, downward/upward simulation preorder is computable in polynomial time [1], but typically yields only small under-approximations of the corresponding trace inclusions. In the following section we consider generalized lookahead simulations in the style of [11], in order to obtain larger relations, while maintaining an adequate balance between complexity and size.

4 Simulations

Simulation preorders on bottom-up tree automata were defined co-inductively in [4]. We adapt the definitions to our TDTA representation, and add a condition about the preservation of accepting states.

Downward simulation. Let $A = (\Sigma, Q, \delta, I)$ be a TDTA. A *downward simulation* D is a binary relation on Q such that if $q\ D\ r$, then $(q = \psi \to r = \psi)$ and for any $\langle q, a, q_1 \dots q_n \rangle \in \delta$, there exists $\langle r, a, r_1 \dots r_n \rangle \in \delta$ such that $q_i\, D\, r_i$ for each $i : 1 \leq i \leq n$.

Upward simulation. Let $A = (\Sigma, Q, \delta, I)$ be a TDTA. Given a binary relation R on Q, an *upward simulation* $U(R)$ induced by R is a binary relation on Q such that if $q\ U(R)\ r$, then

1. $(q = \psi \to r = \psi)$ and $(q \in I \to r \in I)$.
2. for any $\langle q', a, q_1 \dots q_n \rangle \in \delta$ with $q_i = q$ (for some $i : 1 \leq i \leq n$), there exists $\langle r', a, r_1 \dots r_n \rangle \in \delta$ such that $r_i = r$, $q'\ U(R)\ r'$ and $q_j\ R\ r_j$ for each $j : 1 \leq j \neq i \leq n$.

Since the set of all downward simulations on A is closed under union and under reflexive and transitive closure (cf. Lemma 4.1 in [17]), it follows that there is a unique maximal downward simulation on A, and that relation is a preorder. We call it *the downward simulation preorder on A* and write $\sqsubseteq^{\mathsf{dw}}$, and we call the equivalence relation $\equiv := \sqsubseteq^{\mathsf{dw}} \cap (\sqsubseteq^{\mathsf{dw}})^{-1}$ *the downward simulation equivalence on A*. Similarly, for any given relation R, there is a unique maximal upward simulation induced by R which is a preorder (cf. Lemma 4.2 in [17]). We call it *the upward simulation preorder on A induced by R* and write $\sqsubseteq^{\mathsf{up}}(R)$, and we call the equivalence relation $\equiv := \sqsubseteq^{\mathsf{up}}(R) \cap (\sqsubseteq^{\mathsf{up}}(R))^{-1}$ *the upward simulation equivalence on A induced by R*.

Downward/upward simulations are computable in polynomial time and under-approximate the corresponding downward/upward trace inclusions, respectively. I.e., $\sqsubseteq^{\mathsf{dw}} \subseteq \subseteq^{\mathsf{dw}}$ and $\sqsubseteq^{\mathsf{up}}(R) \subseteq \subseteq^{\mathsf{up}}(R)$. However, this approximation is poor, since simulations are much smaller than trace inclusions on many automata. For word automata, more general *lookahead simulations* were introduced in [11]. These provide a practically useful tradeoff between the computational effort and the size of the obtained relations. Lookahead simulations can also be seen as a particular restriction of the more general (but less practically useful) *multipebble simulations* [15]; cf. Section 4 in [11]. We generalize lookahead simulations to tree automata in order to compute good approximations of trace inclusions.

Lookahead downward simulation. We say that a tree t is k-bounded iff for all leaves v of t, either **a)** $|v| = k$, or **b)** $|v| < k$ and v is closed. Let $A = (\Sigma, Q, \delta, I)$ be a TDTA. A *k-lookahead downward simulation* $L^{k\text{-}\mathsf{dw}}$ is a binary relation on Q such that if $q\ L^{k\text{-}\mathsf{dw}}\ r$, then $(q = \psi \to r = \psi)$ and the following holds: Let π_k be a run on a k-bounded tree t_k with $\pi(\varepsilon) = q$ s.t. every leaf node of π_k is either at depth k or downward-deadlocked. Then there must exist a run π'_k over a nonempty prefix t'_k of t_k s.t. (1) $\pi'_k(\varepsilon) = r$, and (2) for every leaf v of π'_k, $\pi_k(v)\ L^{k\text{-}\mathsf{dw}}\ \pi'_k(v)$. Since, for given A and $k \geq 0$, lookahead downward simulations are closed under union, there exists a unique maximal one that we call *the k-lookahead downward simulation on A*, denoted by $\sqsubseteq^{k\text{-}\mathsf{dw}}$.

not defined

While $\sqsubseteq^{k\text{-}\mathsf{dw}}$ is trivially reflexive, it is not transitive in general (cf. Appendix B in [11]). Since we only use it as a means to under-approximate the transitive trace inclusion $\subseteq^{\mathsf{dw}}$ (and require a preorder to induce an equivalence), we work with its transitive closure $\preceq^{k\text{-}\mathsf{dw}} := (\sqsubseteq^{k\text{-}\mathsf{dw}})^{+}$.

Lookahead upward simulation. A leaf v_π of a run π on a linear tree t is called a *bottom leaf* of π iff the (only) leaf v_t of t is its prefix (i.e., v_π is either v_t or its child).

how can v_t (which is a leaft) have a child?

Let $A = (\Sigma, Q, \delta, I)$ be a TDTA. A *k-lookahead upward simulation* on A induced by a relation R is a binary relation $\sqsubseteq^{k\text{-}\mathsf{up}}(R)$ on Q s.t. if $q \sqsubseteq^{k\text{-}\mathsf{up}}(R)\ r$, then $(q = \psi \to r = \psi)$ and the following holds: Let π be a run over a linear tree t with $|dom(t)| \leq k$ and $\pi(v) = q$ for some bottom leaf v s.t. either $|dom(t)| = k$ or $\pi(\varepsilon)$ is upward-deadlocked. Then there are $v', w \in t$ such that $v = v'w$ and there is a run π' over t_w s.t. $\pi'(w) = r$, $\pi_w(\varepsilon) \sqsubseteq^{k\text{-}\mathsf{up}}(R)\ \pi'(\varepsilon)$, $\pi_w \sqsubseteq^{v}_{R} \pi'$, and $\pi_w(x) \in I \to \pi'(x) \in I$ for all $x \in dom(\pi_w)$ (where π_w is the restriction of π to t_w, i.e., $\pi_w(x) = \pi(v'x)$).

this notation is not defined

Given an automaton A, $k \geq 1$ and a binary relation R, there is one unique maximal k-lookahead upward simulation induced by R. We call it *the k-lookahead upward simulation induced by R on A* and denote it by $\sqsubseteq^{k\text{-up}}(R)$. Since both R and $\sqsubseteq^{k\text{-up}}(R)$ are not necessarily transitive, we first compute its transitive closure, R^+, and we then compute $\preceq^{k\text{-up}}(R) := (\sqsubseteq^{k\text{-up}}(R^+))^+$, which under-approximates the upward trace inclusion $\subseteq^{\mathsf{up}}(R^+)$.

Simulation games. Simulation between states p_0 and q_0 can be defined in terms of a game between two players, Spoiler and Duplicator. Starting in initial configuration (p_0, q_0), Spoiler chooses transitions from p_0 and Duplicator must imitate them *stepwise* from q_0. This yields new configurations from which the game continues. If one player cannot move the other player wins, and Duplicator wins every infinite game. Simulation holds iff Duplicator wins. In normal simulation, Duplicator only knows Spoiler's very next step, while in k-lookahead simulation Duplicator knows Spoiler's k next steps (unless Spoiler's move ends in a deadlocked state). In a trace inclusion game, Duplicator knows all steps of Spoiler in advance. As k gets larger, k-lookahead simulation approximates trace inclusion better and better. For every fixed k, k-lookahead simulation is computable in polynomial time, though the complexity rises quickly in k (doubly exponential for downward- and single exponential for upward lookahead simulation). A crucial trick (described in detail in [11], and included in our definitions above) makes it possible to practically compute it for nontrivial k: Spoiler's moves are built incrementally, and Duplicator is not required to respond to all of Spoiler's announced k next steps, but only to a prefix of them (after which he requests fresh information).

5 Transition Pruning Techniques

We define pruning relations on a TDTA $A = (\Sigma, Q, \delta, I)$. The intuition is that certain transitions may be deleted without changing the language, because 'better' transitions remain. We perform this pruning (i.e., deletion) of transitions by comparing their endpoints over the same symbol $\sigma \in \Sigma$. Given two binary relations R_u and R_d on Q, we define the following relation to compare transitions.

$$P(R_\mathsf{u}, R_\mathsf{d}) = \{(\langle p, \sigma, r_1 \cdots r_n\rangle, \langle p', \sigma, r'_1 \cdots r'_n\rangle) \mid p\, R_\mathsf{u}\, p' \text{ and } (r_1 \cdots r_n)\, \hat{R}_\mathsf{d}\, (r'_1 \cdots r'_n)\},$$

where $\hat{R}_\mathsf{d}$ results from lifting $R_\mathsf{d} \subseteq Q \times Q$ to $\hat{R}_\mathsf{d} \subseteq Q^n \times Q^n$. The function P is monotone in the two arguments. If $t\, P\, t'$ then t may be pruned because t' is 'better' than t. We want $P(R_\mathsf{u}, R_\mathsf{d})$ to be a strict partial order (p.o.), i.e., irreflexive and transitive. Consequently, $P(R_\mathsf{u}, R_\mathsf{d})$ is acyclic. There are three possible cases in which $P(R_\mathsf{u}, R_\mathsf{d})$ is guaranteed to be a strict p.o.: **1)** if R_u is some strict p.o. $<_\mathsf{u}$ and $\hat{R}_\mathsf{d}$ is the lifting $\hat{\leq}_\mathsf{d}$ of some p.o. $\leq_\mathsf{d}$ to tuples, **2)** if R_u is some p.o. $\leq_\mathsf{u}$ and $\hat{R}_\mathsf{d}$ is the lifting $\hat{<}_\mathsf{d}$ of some strict p.o. $<_\mathsf{d}$ to tuples, or **3)** if both relations are strict p.o. . The transitions in each pair of $P(<_\mathsf{u}, \leq_\mathsf{d})$ depart from states which must be different and therefore they are necessarily different, while the transitions in each pair of $P(\leq_\mathsf{u}, <_\mathsf{d})$ must go to different tuples of states and thus are different. Since for two tuples $(r_1 \cdots r_n)$ and $(r'_1 \cdots r'_n)$ to be different it suffices that $r_i \neq r'_i$ for some $1 \leq i \leq n$, we define $\hat{<}_\mathsf{d}$ as a binary relation such that $(r_1 \cdots r_n) \hat{<}_\mathsf{d} (r'_1 \cdots r'_n)$ iff **1)** $\forall_{1 \leq i \leq n}.\, r_i \leq_\mathsf{d} r'_i$, and **2)** $\exists_{1 \leq i \leq n}.\, r_i <_\mathsf{d} r'_i$. To define the relation $\hat{\leq}_d$ the first condition is sufficient.

Let $A = (\Sigma, Q, \delta, I)$ be a TDTA and let $P \subseteq \delta \times \delta$ be a strict partial order. The pruned automaton is defined as $\mathit{Prune}(A, P) = (\Sigma, Q, \delta', I)$ where $\delta' = \{(p, \sigma, r) \in \delta \mid \nexists (p', \sigma, r') \in \delta.\, (p, \sigma, r)\, P\, (p', \sigma, r')\}$. Note that the pruned automaton $\mathit{Prune}(A, P)$ is unique. The transitions are removed without requiring the re-computation of the relation P, which could be expensive. Since removing transitions cannot introduce new words in the language, $L(\mathit{Prune}(A, P)) \subseteq L(A)$. If the reverse inclusion holds too (so that the language is preserved), we say that P is *good for pruning* (GFP), i.e., P is GFP iff $L(\mathit{Prune}(A, P)) = L(A)$.

We now provide a complete picture which combinations of simulation and trace inclusion relations are GFP. Recall that simulations are denoted by square symbols $\sqsubseteq$ while trace inclusions are denoted by round symbols $\subseteq$. For every partial order R, the corresponding strict p.o. is defined as $R \backslash R^{-1}$.

This notation is inconsistent with the above def. of P(...,...). Maybe one can define P(Ru, Rd), where Rd is already the lifting to tuples of some relation

acyclic is not defined. isn't it the same as irreflexive for transitive relations?

of

except for the empty relation

$P(\subset^{\mathsf{bw}}, \subset^{\mathsf{di}})$ is not GFP for automata over words (see Fig. 2(a) in [11] for a counterexample). As mentioned before, words correspond to linear trees. Thus $P(\subset^{\mathsf{up}}(R), \subset^{\mathsf{dw}})$ is not GFP for tree automata (regardless of the relation R). Figure 1 presents several more counterexamples. For word automata, $P(\subset^{\mathsf{bw}}, \sqsubseteq^{\mathsf{di}})$ and $P(\sqsubseteq^{\mathsf{bw}}, \subset^{\mathsf{di}})$ are not GFP, even though $P(\subseteq^{\mathsf{bw}}, \sqsubset^{\mathsf{di}})$ and $P(\sqsubset^{\mathsf{bw}}, \subseteq^{\mathsf{di}})$ are (cf. [11]). Thus $P(\subset^{\mathsf{up}}(R), \sqsubseteq^{\mathsf{dw}})$ and $P(\sqsubseteq^{\mathsf{up}}(R), \subset^{\mathsf{dw}})$ are not GFP for tree automata (regardless of the relation R). For automata over trees, $P(\sqsubset^{\mathsf{up}}(\sqsubset^{\mathsf{dw}}), id)$ and $P(\sqsubset^{\mathsf{up}}(\subset^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$ are not GFP. Moreover, a complex counterexample (see Fig. 8; App. A) is needed to show that $P(\sqsubset^{\mathsf{up}}(\sqsubset^{\mathsf{dw}}), \subset^{\mathsf{dw}})$ is not GFP.

The following theorems and corollaries provide several relations which are GFP.

▶ **Theorem 1.** *For every strict partial order $R \subset \subseteq^{\mathsf{dw}}$, it holds that $P(id(id), R)$ is GFP.*

▶ **Corollary 2.** *By Theorem 1, $P(id(id), \subset^{\mathsf{dw}})$ and $P(id(id), \sqsubset^{\mathsf{dw}})$ are GFP.*

▶ **Theorem 3.** *For every strict partial order $R \subset \subseteq^{\mathsf{up}}(id)$, it holds that $P(R, id)$ is GFP.*

▶ **Corollary 4.** *By Theorem 3, $P(\subset^{\mathsf{up}}(id), id)$ and $P(\sqsubset^{\mathsf{up}}(id), id)$ are GFP.*

▶ **Definition 5.** Given a tree automaton A, a binary relation W on its states is called a *downup-relation* iff the following condition holds: If $p\ W\ q$ then for every tree $t \in \mathbb{T}(\Sigma)$ and accepting t-run π from p there exists an accepting t-run π' from q such that $\forall_{v \in \mathbb{N}^*}\ \pi(v) \sqsubseteq^{\mathsf{up}}(W)\ \pi'(v)$.

we are looking for a downward simulation W that preserves upward simulation parametrised with W

in dom(π)

▶ **Lemma 6.** *Any relation V satisfying **1)** V is a downward simulation, and **2)** $id \subseteq V \subseteq \sqsubseteq^{\mathsf{up}}(V)$ is a downup-relation. In particular, id is a downup-relation, but $\sqsubseteq^{\mathsf{dw}}$ and $\sqsubseteq^{\mathsf{up}}(id)$ are not.*

▶ **Theorem 7.** *For every downup-relation W, it holds that $P(\sqsubset^{\mathsf{up}}(W), \subseteq^{\mathsf{dw}})$ is GFP.*

Proof. Let $A' = Prune(A, P(\sqsubset^{\mathsf{up}}(W), \subseteq^{\mathsf{dw}}))$. We show $L(A) \subseteq L(A')$. If $t \in L(A)$ then there exists an accepting t-run $\hat{\pi}$ in A. We show that there is an accepting t-run $\hat{\pi}'$ in A'.

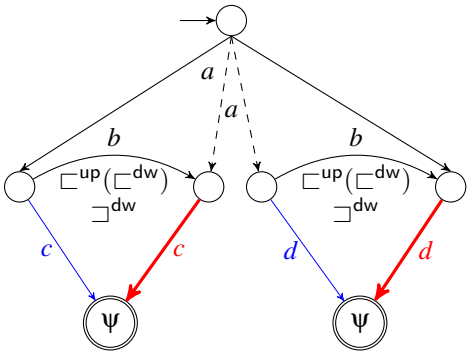
For each accepting t-run π in A, let $level_i(\pi)$ be the tuple of states that π visits at depth i in the tree, read from left to right. Formally, let $(x_1, \ldots, x_k)$ with $x_j \in \mathbb{N}^i$ be the set of all tree positions of depth i s.t. $x_j \in dom(\pi)$, in lexicographically increasing order. Then $level_i(\pi) = (\pi(x_1), \ldots, \pi(x_k)) \in Q^k$. By lifting partial orders on Q to partial orders on tuples, we can compare such tuples w.r.t. $\sqsubseteq^{\mathsf{up}}(W)$.

We say that an accepting t-run π is i-good iff it does not contain any transition from $A - A'$ from any position $v \in \mathbb{N}^*$ with $|v| < i$. I.e., no pruned transition is used in the first i levels of the tree.

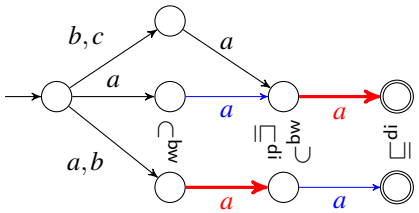
We now define a strict partial order $<_i$ on the set of accepting t-runs in A. Let $\pi <_i \pi'$ iff $\exists k \leq i.\ level_k(\pi) \sqsubset^{\mathsf{up}}(W)\ level_k(\pi')$ and $\forall l < k.\ level_l(\pi) \sqsubseteq^{\mathsf{up}}(W)\ level_l(\pi')$. Note that $<_i$ only depends on the first i levels of the run. Given A, t and i, there are only finitely many different such i-prefixes of accepting t-runs. By our assumption that $\hat{\pi}$ is an accepting t-run in A, the set of accepting t-runs in A is non-empty. Thus, for any i, there must exist some accepting t-run π in A that is maximal w.r.t. $<_i$.

We now show that this π is also i-good, by assuming the contrary and deriving a contradiction. Suppose that π is not i-good. Then it must contain a transition $\langle p, \sigma, r_1 \cdots r_n \rangle$ from $A - A'$ used at the root of some subtree t' of t at some level $j < i$. Since $A' = Prune(A, P(\sqsubset^{\mathsf{up}}(W), \subseteq^{\mathsf{dw}}))$, there must exist another transition $\langle p', \sigma, r'_1 \cdots r'_n \rangle$ in A' s.t. (1) $(r_1, \ldots, r_n) \subseteq^{\mathsf{dw}} (r'_1, \ldots, r'_n)$ and (2) $p \sqsubset^{\mathsf{up}}(W)\ p'$.

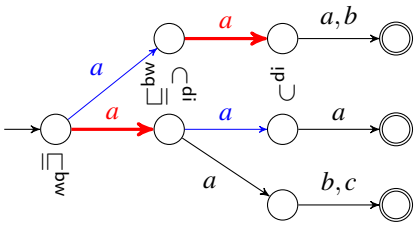
First consider the implications of (2). Upward simulation propagates upward stepwise (though only in non-strict form after the first step). So p' can imitate the upward path of p to the root of t, maintaining $\sqsubseteq^{\mathsf{up}}(W)$ between the corresponding states. The states on side branches joining in along the upward path from p can be matched by W-larger states in joining side branches along the upward path from p'. From Def. 5 we obtain that these W-larger states in p's joining side branches can accept their subtrees of t via computations that are everywhere $\sqsubseteq^{\mathsf{up}}(W)$ larger than corresponding states in computations from ps joining side branches. So there must be an accepting run π' on t s.t. (3) π' is at state p' at the root of t' and uses transition $\langle p', \sigma, r'_1 \cdots r'_n \rangle$ from p', and (4) for all $v \in \mathbb{N}^*$ where $t(v) \notin t'$ we have $\pi(v) \sqsubseteq^{\mathsf{up}}(W)\ \pi'(v)$. Moreover, by conditions (1) and (3), π' can be extended from



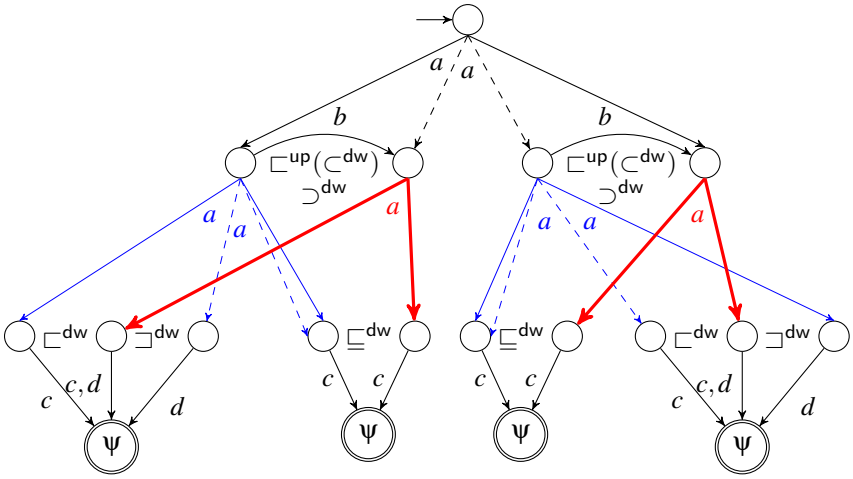
(a) $P(\sqsubset^{up}(\sqsubset^{dw}), id)$ is not GFP: if we remove the blue transitions, the automaton no longer accepts the tree $a(c,d)$. We are considering $\Sigma_0 = \{c,d\}$, $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{a\}$.



(b) $P(\subset^{bw}, \sqsubseteq^{di})$ is not GFP for words: if we remove the blue transitions, the automaton no longer accepts the word *aaa*.



(c) $P(\sqsubseteq^{bw}, \subset^{di})$ is not GFP for words: if we remove the blue transitions, the automaton no longer accepts the word *aaa*.



(d) $P(\sqsubset^{up}(\subset^{dw}), \sqsubset^{dw})$ is not GFP: if we remove the blue transitions, the tree $a(a(c,c),a(c,c))$ is no longer accepted. We are considering $\Sigma_0 = \{c,d\}$, $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{a\}$.

Figure 1 GFP counterexamples. A transition is drawn in dashed when a different transition by the same symbol departing from the same state already exists. We draw a transition in thick red when it is better than another transition (drawn in thin blue).

$r'_1,\ldots,r'_n$ to accept also the subtree t'. Thus π' is an accepting t-run in A. By conditions (2) and (4) we obtain that $\forall l \leq j.\ level_l(\pi) \sqsubseteq^{\mathsf{up}}(W)\ level_l(\pi')$. By (2) we get even $level_j(\pi) \sqsubset^{\mathsf{up}}(W)\ level_j(\pi')$ and thus $\pi <_j \pi'$. Since $j < i$ we also have $\pi <_i \pi'$ and thus π was not maximal w.r.t. $<_i$. Contradiction. So we have shown that for every $t \in L(A)$ there exists an i-good accepting run for every finite i.

If $t \in L(A)$ then there exists an accepting t-run $\hat{\pi}$ in A. Then there exists an accepting t-run $\hat{\pi}'$ that is i-good, where i is the height of t. Thus $\hat{\pi}'$ is a run in A' and $t \in L(A')$. recall that V is downup ◀

▶ **Corollary 8.** *It follows from Lemma 6 and from the fact that GFP is downward closed that* $P(\sqsubset^{\mathsf{up}}(V), \subseteq^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(V), \subset^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(V), \sqsubseteq^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(V), \sqsubset^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(V), id)$, $P(\sqsubset^{\mathsf{up}}(id), \subseteq^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(id), \subset^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(id), \sqsubseteq^{\mathsf{dw}})$ *and* $P(\sqsubset^{\mathsf{up}}(id), \sqsubset^{\mathsf{dw}})$ *are GFP.*

▶ **Theorem 9.** $P(\subseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$ *is GFP.*

Proof. Let $A' = Prune(A, P(\subseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}}))$. We show $L(A) \subseteq L(A')$. If $t \in L(A)$ then there exists an accepting t-run $\hat{\pi}$ in A. We show that there is an accepting t-run $\hat{\pi}'$ in A'.

For each accepting t-run π in A, let $level_i(\pi)$ be the tuple of states that π visits at depth i in the tree, read from left to right. Formally, let $(x_1,\ldots,x_k)$ with $x_j \in \mathbb{N}^i$ be the set of all tree positions of depth i s.t. $x_j \in dom(\pi)$, in lexicographically increasing order. Then $level_i(\pi) = (\pi(x_1),\ldots,\pi(x_k)) \in Q^k$. By lifting partial orders on Q to partial orders on tuples we can compare such tuples w.r.t. $\sqsubseteq^{\mathsf{dw}}$.

We say that an accepting t-run π is i-good if it does not contain any transition from $A - A'$ from any position $v \in \mathbb{N}^*$ with $|v| < i$. I.e., no pruned transitions are used in the first i levels of the tree.

We now show, by induction on i, the following property (C): For every i and every accepting t-run π in A there exists an i-good accepting t-run π' in A s.t. $level_i(\pi) \sqsubseteq^{\mathsf{dw}} level_i(\pi')$.

The base case is $i = 0$. Every accepting t-run π in A is trivially 0-good itself and thus satisfies (C).

For the induction step, let S be the set of all $(i-1)$-good accepting t-runs π' in A s.t. $level_{i-1}(\pi) \sqsubseteq^{\mathsf{dw}} level_{i-1}(\pi')$. Since π is an accepting t-run, by induction hypothesis, S is non-empty. Let $S' \subseteq S$ be the subset of S containing exactly those runs $\pi' \in S$ that additionally satisfy $level_i(\pi) \sqsubseteq^{\mathsf{dw}} level_i(\pi')$. From $level_{i-1}(\pi) \sqsubseteq^{\mathsf{dw}} level_{i-1}(\pi')$ and the fact that $\sqsubseteq^{\mathsf{dw}}$ is preserved downward-stepwise, we obtain that S' is non-empty. Now we can select some $\pi' \in S'$ s.t. $level_i(\pi')$ is maximal, w.r.t. $\sqsubseteq^{\mathsf{dw}}$, relative to the other runs in S'. We claim that π' is i-good and $level_i(\pi) \sqsubseteq^{\mathsf{dw}} level_i(\pi')$. The second part of this claim holds because $\pi' \in S'$.

We show that π' is i-good by assuming the opposite and deriving a contradiction. Suppose that π' is not i-good. Then it must contain a transition $\langle p, \sigma, r_1 \cdots r_n \rangle$ from $A - A'$. Since π' is $(i-1)$-good, this transition must start at depth $(i-1)$ in the tree. Since $A' = Prune(A, P(\subseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}}))$, there must exist another transition $\langle p', \sigma, r'_1 \cdots r'_n \rangle$ in A' s.t. $p \subseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}})\ p'$ and $(r_1,\ldots,r_n) \sqsubset^{\mathsf{dw}} (r'_1,\ldots,r'_n)$. From the definition of $\subseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}})$ we obtain that there exists another accepting t-run π_1 in A (that uses the transition $\langle p', \sigma, r'_1 \cdots r'_n \rangle$) s.t. $level_i(\pi') \sqsubset^{\mathsf{dw}} level_i(\pi_1)$. The run π_1 is not necessarily i-good or $(i-1)$-good. However, by induction hypothesis, there exists some accepting t-run π_2 in A that is $(i-1)$-good and satisfies $level_{i-1}(\pi_1) \sqsubseteq^{\mathsf{dw}} level_{i-1}(\pi_2)$. Since $\sqsubseteq^{\mathsf{dw}}$ is preserved stepwise, there also exists an accepting t-run π_3 in A (that coincides with π_2 up-to depth $(i-1)$), which is $(i-1)$-good and satisfies $level_i(\pi_1) \sqsubseteq^{\mathsf{dw}} level_i(\pi_3)$. In particular, $\pi_3 \in S'$.

From $level_i(\pi') \sqsubset^{\mathsf{dw}} level_i(\pi_1)$ and $level_i(\pi_1) \sqsubseteq^{\mathsf{dw}} level_i(\pi_3)$ we obtain $level_i(\pi') \sqsubset^{\mathsf{dw}} level_i(\pi_3)$. This contradicts our condition above that π' must be $level_i$ maximal w.r.t. $\sqsubseteq^{\mathsf{dw}}$ in S'. This concludes the induction step and the proof of property (C).

If $t \in L(A)$ then there exists an accepting t-run $\hat{\pi}$ in A. By property (C), there exists an accepting t-run $\hat{\pi}'$ that is i-good, where i is the height of t. Therefore $\hat{\pi}'$ does not use any transition from $A - A'$ and is thus also a run in A'. So we obtain $t \in L(A')$. ◀

▶ **Corollary 10.** *It follows from Theorem 9 and the fact that GFP is downward closed that* $P(\subset^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$, $P(\sqsubseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$, $P(\sqsubset^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$, $P(\subseteq^{\mathsf{up}}(id), \sqsubset^{\mathsf{dw}})$, $P(\subset^{\mathsf{up}}(id), \sqsubset^{\mathsf{dw}})$, $P(\sqsubseteq^{\mathsf{up}}(id), \sqsubset^{\mathsf{dw}})$ *and* $P(id(id), \sqsubset^{\mathsf{dw}})$ *are GFP.*

6 State Quotienting Techniques

A classic method for reducing the size of automata is state quotienting. Given a suitable equivalence relation on the set of states, each equivalence class is collapsed into just one state. From a preorder $\sqsubseteq$ one obtains an equivalence relation $\equiv := \sqsubseteq \cap \sqsupseteq$. We now define quotienting w.r.t. $\equiv$. Let $A = (\Sigma, Q, \delta, I)$ be a TDTA and let $\sqsubseteq$ be a preorder on Q. Given $q \in Q$, we denote by $[q]$ its equivalence class w.r.t $\equiv$. For $P \subseteq Q$, $[P]$ denotes the set of equivalence classes $[P] = \{[p] \mid p \in P\}$. We define the quotient automaton w.r.t. $\equiv$ as $A/\equiv := (\Sigma, [Q], \delta_{A/\equiv}, [I])$, where $\delta_{A/\equiv} = \{\langle [q], \sigma, [q_1] \dots [q_n] \rangle \mid \langle q, \sigma, q_1 \dots q_n \rangle \in \delta_A\}$. It is trivial that $L(A) \subseteq L(A/\equiv)$ for any $\equiv$. If the reverse inclusion also holds, i.e., if $L(A) = L(A/\equiv)$, we say that $\equiv$ is *good for quotienting* (GFQ).

It was shown in [17] that $\sqsubseteq^{\mathsf{dw}} \cap \sqsupseteq^{\mathsf{dw}}$ and $\sqsubseteq^{\mathsf{up}}(id) \cap \sqsupseteq^{\mathsf{up}}(id)$ are GFQ. Here we generalize this result from simulation to trace equivalence. Let $\equiv^{\mathsf{dw}} := \subseteq^{\mathsf{dw}} \cap \supseteq^{\mathsf{dw}}$ and $\equiv^{\mathsf{up}}(R) := \subseteq^{\mathsf{up}}(R) \cap \supseteq^{\mathsf{up}}(R)$.

▶ **Theorem 11.** *$\equiv^{\mathsf{dw}}$ is GFQ.*

▶ **Theorem 12.** *$\equiv^{\mathsf{up}}(id)$ is GFQ.*

In Figure 9 (cf. Appendix A) we present a counterexample showing that $\equiv := \sqsubseteq^{\mathsf{up}}(\equiv^{\mathsf{dw}}) \cap \sqsupseteq^{\mathsf{up}}(\equiv^{\mathsf{dw}})$ is not GFQ. This is an adaptation from the Example 5 in [17], where the inducing relation is referred to as the *downward bisimulation equivalence* and the automata are seen bottom-up. As a corollary of this counterexample, $\equiv := \sqsubseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}) \cap \sqsupseteq^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}})$ is not GFQ either.

7 Experiments

Our tree automata minimization algorithm (tool available [6]) combines transition pruning techniques (Sec. 5) with quotienting techniques (Sec. 6). Trace inclusions are under-approximated by lookahead simulations (Sec. 4) where higher lookaheads are harder to compute but yield better approximations. The parameters $x, y \geq 1$ describe the lookahead for downward/upward lookahead simulations, respectively. Downward lookahead simulation is harder to compute than upward lookahead simulation, since the number of possible moves is doubly exponential in x (due to the downward branching of the tree) while for upward-sim. it is only single exponential in y. We use $(x, y) = (1,1), (2,4), (3,6)$.

Besides pruning and quotienting, we also use the operation *RU* that removes useless states, i.e., states that either cannot be reached from any initial state or from which no tree can be accepted. Let $Op(x, y)$ be the following sequence of operations on tree automata: *RU*, quotienting with $\preceq^{x\text{-}\mathsf{dw}}$, pruning with $P(id, \prec^{x\text{-}\mathsf{dw}})$, *RU*, quotienting with $\preceq^{y\text{-}\mathsf{up}}(id)$, pruning with $P(\prec^{y\text{-}\mathsf{up}}(id), id)$, pruning with $P(\sqsubset^{\mathsf{up}}(id), \preceq^{x\text{-}\mathsf{dw}})$, *RU*, quotienting with $\preceq^{y\text{-}\mathsf{up}}(id)$, pruning with $P(\preceq^{y\text{-}\mathsf{up}}(\sqsubseteq^{\mathsf{dw}}), \sqsubset^{\mathsf{dw}})$, *RU*. It is language preserving by the Theorems of Sections 5 and 6. The order of the operations is chosen according to some considerations of efficiency. (No order is ideal for all instances.)

Our algorithm $Heavy(1,1)$ just iterates $Op(1,1)$ until a fixpoint is reached. For efficiency reasons, the general algorithm $Heavy(x, y)$ does not iterate $Op(x, y)$, but uses a double loop: it iterates the sequence $Heavy(1,1)Op(x, y)$ until a fixpoint is reached.

We compare the minimization performance of several algorithms.

RU: *RU*. (Previously present in `libvata`.)
RUQ: *RU* and quotienting with $\sqsubseteq^{\mathsf{dw}}$. (Previously present in `libvata`.)
RUQP: *RU*, quotienting with $\sqsubseteq^{\mathsf{dw}}$ and pruning with $P(id, \sqsubset^{\mathsf{dw}})$. (Not in `libvata`, but simple.)
Heavy: $Heavy(1,1)$, $Heavy(2,4)$ and $Heavy(3,6)$. (New.)

We tested these algorithms on three sets of automata from the `libvata` distribution. The first set are 27 moderate-sized automata (87 states and 816 transitions on avg.) derived from regular model checking applications. Heavy(1,1), on avg., reduced the number of states and transitions *to* 27% and

14% of the original sizes, resp. (Note the difference between 'to' and 'by'.) In contrast, RUQ reduced the number of states and transitions only to 81% and 80% of the original sizes and RUQP reduced the number of states and transitions to 81% and 32% of the original sizes; cf. Fig. 2.

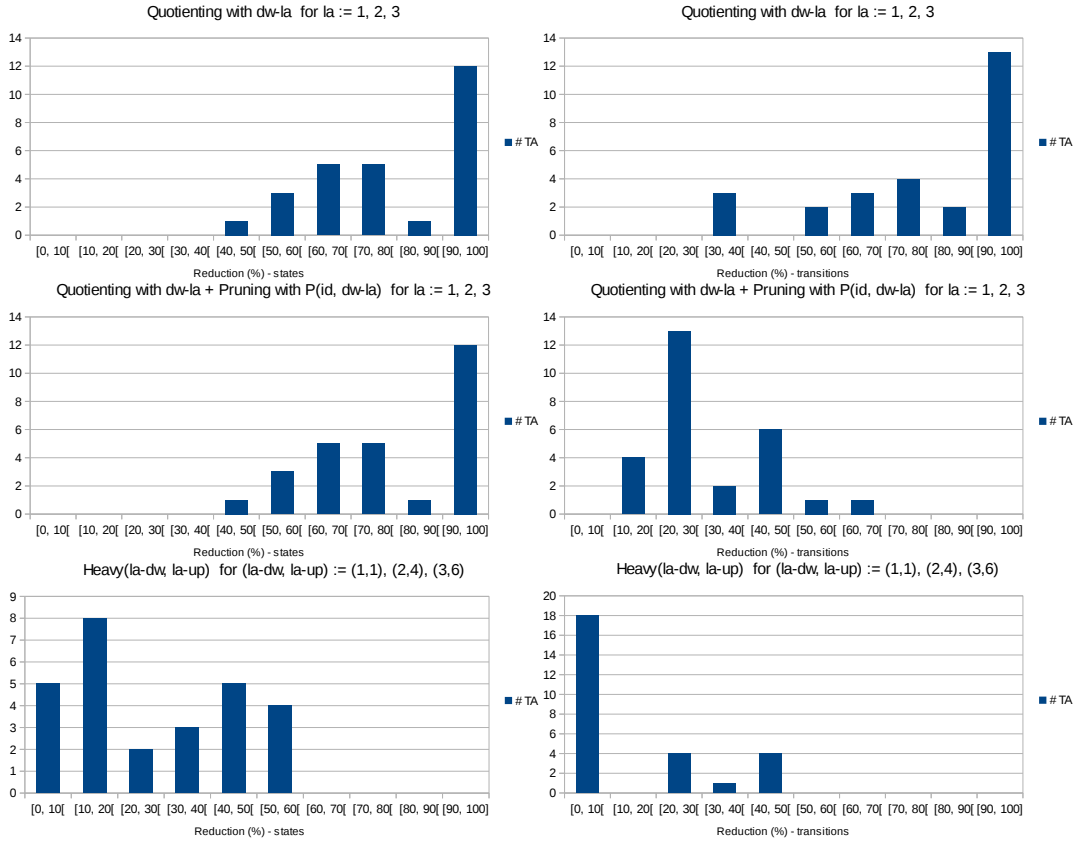


Figure 2 Minimization of 27 moderate-sized tree automata by methods RUQ (top), RUQP (middle) and Heavy (bottom). A bar of height h at an interval $[x, x+10[$ means that h of the 27 automata were reduced to a size between x% and $(x+10)$% of their original size. The reductions in the numbers of states/transitions are shown on the left/right, respectively. On this set of automata, the methods Heavy(2,4) and Heavy(3,6) gave exactly the same results as Heavy(1,1). The average computation time of Heavy(1,1) was 0.06s.

The second set are 66 larger automata (574 states and 8583 transitions, on avg.) derived from regular model checking applications. Heavy(1,1), on avg., reduced the number of states and transitions *to* 4.2% and 0.7% of the original sizes. (Avg. comp. time of Heavy(1,1): 3.0s. cf. Table 2 in App.C). RUQ reduced the number of states and transitions to 72.9% and 72.0% of the original sizes and RUQP reduced the number of states and transitions to 72.9% and 15.3% of the original sizes.

The third set are 14,498 automata (57 states and 266 transitions on avg.) from the shape analysis tool Forester [21]. Heavy(1,1), on avg., reduced the number of states/transitions *to* 76% and 68% of the original, resp. RUQ and RUQP reduced the states/transitions only to 94%/88% of the original.

Due to the particular structure of the automata in these 3 sample sets, $Heavy(2,4)$ and $Heavy(3,6)$ had hardly any advantage over $Heavy(1,1)$. However, in general they can perform significantly better.

We also tested the algorithms on randomly generated tree automata, according to a generalization of the Tabakov-Vardi model of random word automata [22]. Given parameters n, s, td (transition density) and ad (acceptance density), it generates tree automata with n states, s symbols (each of rank 2), $n * td$ randomly assigned transitions for each symbol, and $n * ad$ randomly assigned leaf rules. Figure 3 shows the results of minimizing automata of varying td with different methods.

8 Conclusion

The tables in Figure 4 and Figure 5 summarize all our results on pruning and quotienting, respectively. Note that negative results propagate to larger relations and positive results propagate to smaller relations (i.e., GFP/GFQ is downward closed).

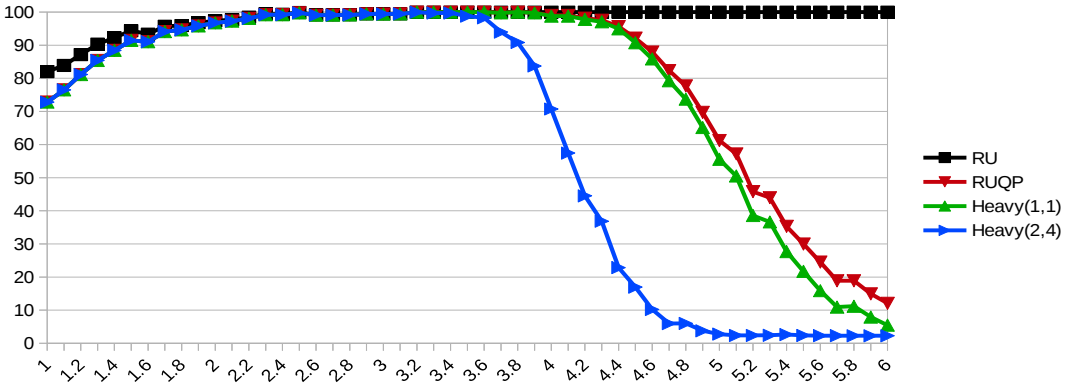


Figure 3 Minimization of Tabakov-Vardi random tree automata with $n = 100, s = 2$ and $ad = 0.8$. The x-axis gives the transition density td, and the y-axis gives the average number of states after minimization with the various methods (smaller is better). Each data point is the average of 400 random automata. Note that Heavy(2,4) minimizes much better than Heavy(1,1) for $td \geq 3.5$.

		R_d				
	$R_u \backslash R_i$	id	$\sqsubset^{dw}$	$\sqsubseteq^{dw}$	$\subset^{dw}$	$\subseteq^{dw}$
id	id	–	✓	–	✓	–
$\sqsubset^{up}$	id	✓	✓	✓	✓	✓
	$\sqsubset^{dw}$	×	✓	×	×	×
	$\sqsubseteq^{dw}$	×	✓	×	×	×
	downup-rel.	✓	✓	✓	✓	✓
	$\subset^{dw}$	×	×	×	×	×
	$\subseteq^{dw}$	×	×	×	×	×
$\sqsubseteq^{up}$	id	–	✓	–	×	–
	$\sqsubset^{dw}$	–	✓	–	×	–
	$\sqsubseteq^{dw}$	–	✓	–	×	–
	$\subset^{dw}$	–	×	–	×	–
	$\subseteq^{dw}$	–	×	–	×	–
$\subset^{up}$	id	✓	✓	×	×	×
	$\sqsubset^{dw}$	×	✓	×	×	×
	$\sqsubseteq^{dw}$	×	✓	×	×	×
	$\subset^{dw}$	×	×	×	×	×
	$\subseteq^{dw}$	×	×	×	×	×
$\subseteq^{up}$	id	–	✓	–	×	–
	$\sqsubset^{dw}$	–	✓	–	×	–
	$\sqsubseteq^{dw}$	–	✓	–	×	–
	$\subset^{dw}$	–	×	–	×	–
	$\subseteq^{dw}$	–	×	–	×	–

Figure 4 GFP relations $P(R_u(R_i), R_d)$ for automata over trees. Relations which are GFP are marked with ✓, those which are not are marked with × and – is used to mark relations where the test does not apply due to them being reflexive (and therefore not asymmetric).

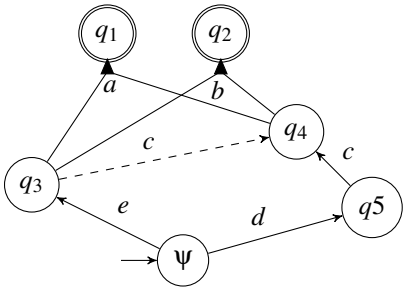
	R	
	$\subseteq^{dw}$	✓
	$\sqsubseteq^{dw}$	✓
$\sqsubseteq^{up}$	id	✓
	$\sqsubset^{dw}$	–
	$\sqsubseteq^{dw}$	×
	$\subset^{dw}$	–
	$\subseteq^{dw}$	×
$\subseteq^{up}$	id	✓
	$\sqsubset^{dw}$	–
	$\sqsubseteq^{dw}$	×
	$\subset^{dw}$	–
	$\subseteq^{dw}$	×

Figure 5 GFQ relations R for automata over trees. Relations which are GFQ are marked with ✓ and those which are not are marked with ×. The relations marked with – are not even reflexive in general (unless all transitions are linear; in this case we have a word automaton and these relations are the same as $\sqsubseteq^{up}(id)$ and $\subseteq^{up}(id)$, respectively).

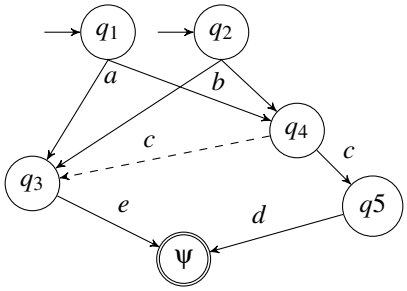
References

1. P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Computing simulations over tree automata. In *TACAS*, volume 4963 of *LNCS*, pages 93–108, 2008.
2. P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010.
3. P. A. Abdulla, L. Holík, B. Jonsson, O. Lengál, C. Q. Trinh, and T. Vojnar. Verification of heap manipulating programs with ordered data by extended forest automata. In D. V. Hung and M. Ogawa, editors, *ATVA*, volume 8172 of *Lecture Notes in Computer Science*, pages 224–239. Springer, 2013.
4. P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Simulation-based iteration of tree transducers. In *Proc. TACAS '05-11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, 2005.
5. P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree Regular Model Checking: A Simulation-Based Approach. *J. Log. Algebr. Program.*, 69(1-2):93–121, 2006.
6. R. Almeida, L. Holík, and R. Mayr. HeavyMinOTAut. `http://tinyurl.com/pm2b4qk`, 2015.
7. D. Basin, N. Karlund, and A. Møller. Mona. `http://www.brics.dk/mona`, 2015.
8. F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *Principles of Programming Languages (POPL), Rome, Italy*. ACM, 2013.
9. A. Bouajjani, P. Habermehl, L. Holík, T. Touili, and T. Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In O. H. Ibarra and B. Ravikumar, editors, *CIAA*, volume 5148 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2008.
10. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking of complex dynamic data structures. In *SAS*, volume 4134 of *LNCS*, pages 52–70, 2006.
11. L. Clemente and R. Mayr. Advanced automata minimization. In *40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 63–74. ACM, 2013.
12. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2008. release November, 18th 2008.
13. I. Durand. Autowrite. `http://dept-info.labri.fr/~idurand/autowrite`, 2015.
14. T. G. et al. Timbuk. `http://www.irisa.fr/celtique/genet/timbuk/`, 2015.
15. K. Etessami. A hierarchy of polynomial-time computable simulations for automata. In *CONCUR*, volume 2421 of *LNCS*, pages 131–144, 2002.
16. P. Habermehl, L. Holík, A. Rogalewicz, J. Simácek, and T. Vojnar. Forest automata for verification of heap manipulation. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 424–440, 2011.
17. L. Holík. *Simulations and Antichains for Efficient Handling of Finite Automata*. PhD thesis, Faculty of Information Technology of Brno University of Technology, 2011.
18. L. Holík, O. Lengál, A. Rogalewicz, J. Simácek, and T. Vojnar. Fully automated shape analysis based on forest automata. In *CAV*, volume 8044 of *LNCS*, pages 740–755, 2013.
19. L. Holík, O. Lengál, J. Simácek, and T. Vojnar. Efficient inclusion checking on explicit and semi-symbolic tree automata. In *ATVA*, volume 6996 of *LNCS*, pages 243–258, 2011.
20. O. Lengál, J. Simácek, and T. Vojnar. Libvata: highly optimised non-deterministic finite tree automata library. `http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/`, 2015.
21. O. Lengál, J. Simácek, T. Vojnar, P. Habermehl, L. Holík, and A. Rogalewicz. Forester: tool for verification of programs with pointers. `http://www.fit.vutbr.cz/research/groups/verifit/tools/forester/`, 2015.
22. D. Tabakov and M. Vardi. Model Checking Büchi Specifications. In *LATA*, volume Report 35/07. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 2007.

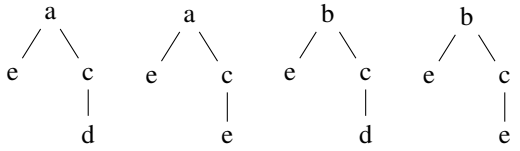
A Examples and Counterexamples for Tree Automata



(a) $A_{BU} = (\Sigma, Q, \delta_{BU}, F = \{q_1, q_2\})$.



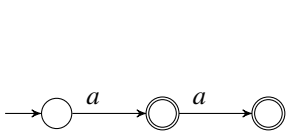
(b) $A_{TD} = (\Sigma, Q, \delta_{TD}, I = \{q_1, q_2\})$.



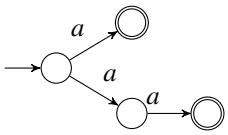
(c) The trees accepted by A_{BU} and A_{TD}.

Figure 6 Let Σ be a ranked alphabet such that $\Sigma_0 = \{d, e\}$, $\Sigma_1 = \{c\}$ and $\Sigma_2 = \{a, b\}$. Consider the BUTA A_{BU} and the TDTA A_{TD}, where $Q = \{q_1, \ldots, q_5\}$ and $\delta_{BU} = \{\langle \psi, e, q_4 \rangle, \langle \psi, d, q_5 \rangle, \langle q_4, c, q_3 \rangle, \langle q_5, c, q_3 \rangle, \langle q_3 q_4, a, q_1 \rangle, \langle q_3 q_4, b, q_2 \rangle\}$. A_{TD} is obtained from A_{BU} by reversing the transition rules in δ_{BU} and by swapping the roles of the accepting and the final states. The language accepted by the automata is $L = \{a(e, c(d)), a(e, c(e)), b(e, c(d)), b(e, c(e))\}$, as represented in c).

In Figure 6, we present two examples of TA, a BUTA and a TDTA, where the second is obtained from the first. We draw the automata vertically, either bottom-up or top-down (depending on if it is a BUTA or a TDTA), to make the reading of an input tree more natural.

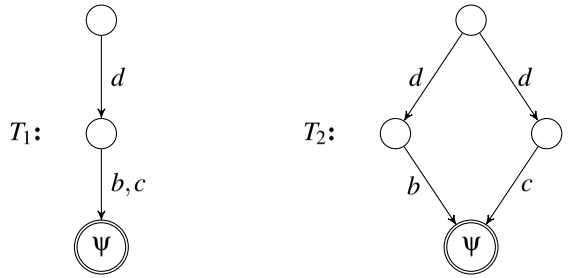


(a) Automaton A_1.

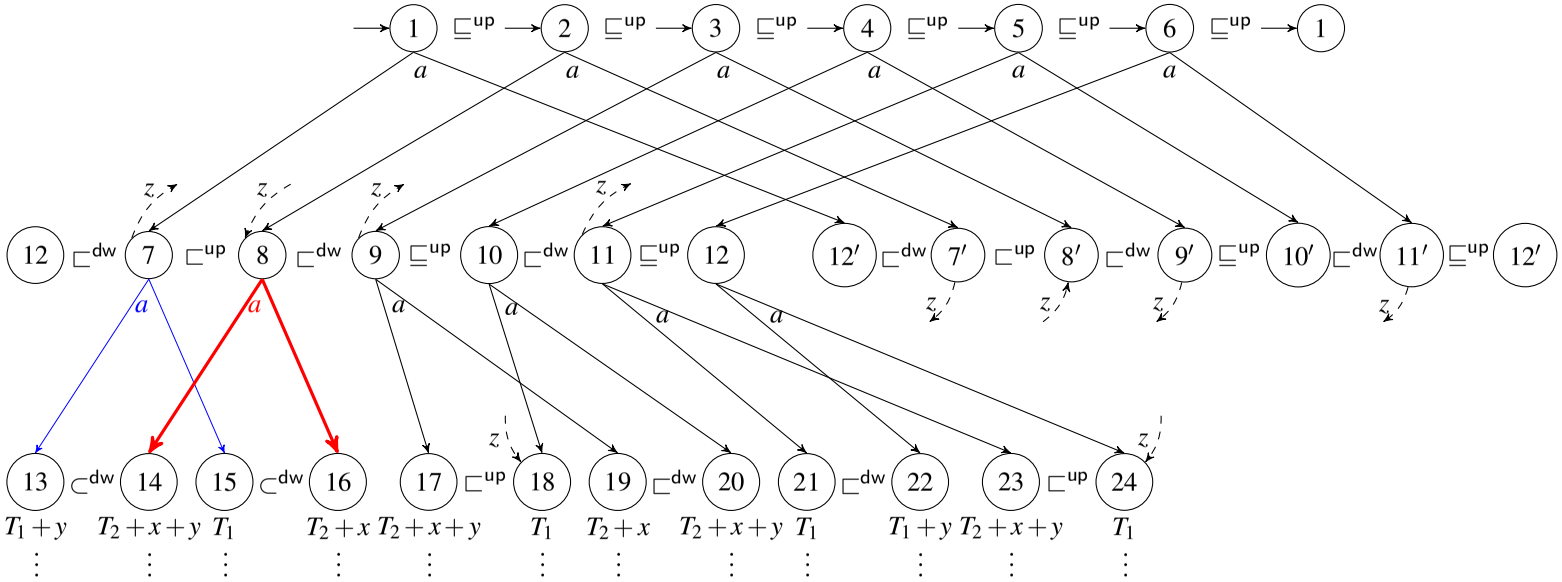


(b) Automaton A_2.

Figure 7 An example of two NFAs, A_1 and A_2, for which language inclusion holds but trace inclusion does not: the trace for *aaa* does not preserve acceptance in the second state in A_2.

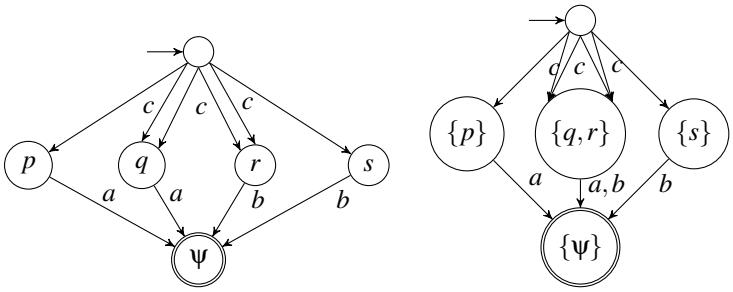


(a) Consider the macros T_1 and T_2. They are used in b) to introduce new transitions and new states in the third level of the automaton as here defined. Note that each of the intermediate states in T_2 is smaller w.r.t $\subset^{\mathsf{dw}}$ than the intermediate state in T_1. Thus, when the first state in T_2 is smaller w.r.t $\sqsubset^{\mathsf{up}}(\sqsubset^{\mathsf{dw}})$ than the first state in T_1, each of the d-transitions in T_2 is a blue one w.r.t. the d-transition in T_1, which is red. Let us also consider the symbols x and y of rank 1. We can extend a macro T by adding transitions from the first state to some new state by x, by y or by both and we denote these by T_x, T_y or T_{x+y}, respectively.



(b) We are considering $\Sigma_0 = \{b,c\}$, $\Sigma_1 = \{d,x,y,z\}$ and $\Sigma_2 = \{a\}$. The dashed arrows represent transitions by z from/to some new state. Each of the six initial states has an a-transition to one of the states from 7 to 12 on the left and one of the states from $7'$ to $12'$ on the right. Any state n' on the right side of the automaton does exactly the same downwardly as the state n on the left side, and therefore needs not be expanded in the figure. We abbreviate $\sqsubset^{\mathsf{up}}(\sqsubset^{\mathsf{dw}})$ as simply $\sqsubset^{\mathsf{up}}$ and $\sqsubseteq^{\mathsf{up}}(\sqsubset^{\mathsf{dw}})$ as $\sqsubseteq^{\mathsf{up}}$.

Figure 8 $P(\sqsubset^{\mathsf{up}}(\sqsubset^{\mathsf{dw}}), \subset^{\mathsf{dw}})$ is not GFP since the automaton presented in b) cannot read the full binary tree of a's with height 3 without using a blue transition: a run starting in state 1 encounters a blue transition from 7, as illustrated in the figure; and since $7'$ and $8'$ do the same downwardly as 7 and 8, respectively, and since $7' \sqsubset^{\mathsf{up}} 8'$, we have that there is a blue transition from $7'$ as well, and so 2 cannot be used either; since $17 \sqsubset^{\mathsf{up}} 18$ we have that, as explained in a), there is a blue transition departing from 17, thus a run starting at 3 too cannot be used; and since 9 is downwardly imitated by $9'$, a run using this state finds a blue transition as well, and so 4 is not safe; since $23 \sqsubset^{\mathsf{up}} 24$, a blue transition from 23 exists and so 5 cannot be used; finally, since 11 is imitated by $11'$, we have that a run using this state encounters a blue transition as well, and so 6 too is not safe.



(a) Automaton A. **(b)** Automaton $A/\equiv$.

Figure 9 $\equiv := \sqsubseteq^{\mathsf{up}}(\equiv^{\mathsf{dw}}) \cap \sqsupseteq^{\mathsf{up}}(\equiv^{\mathsf{dw}})$ is not GFQ. We are considering $\Sigma_0 = \{a,b\}$ and $\Sigma_2 = \{c\}$. Computing all the necessary relations to quotient A w.r.t. $\equiv$, we obtain $\sqsubseteq^{\mathsf{dw}} = \{(p,q),(r,s)\} = \sqsupseteq^{\mathsf{dw}} = \equiv^{\mathsf{dw}}$ and $\sqsubseteq^{\mathsf{up}}(\equiv^{\mathsf{dw}}) = \{(q,r),(r,q)\}$. Thus $\equiv = \{(q,r),(r,q)\}$. Computing $A/\equiv$, we verify that $c(b,a)$ is now accepted by the automaton, while it was not in the language of A.

B Proofs of Theorems

Theorem 1. For every strict partial order $R \subset \subseteq^{\mathsf{dw}}$, it holds that $P(id(id), R)$ is GFP.

Proof. Let $A' = Prune(A, P(id(id), R))$. We show $L(A) \subseteq L(A')$. If $t \in L(A)$ then there exists an accepting t-run $\hat{\pi}$ in A. We show that there exists an accepting t-run in A'.

A t-run π_1 in A is called i-good if no transition from $A - A'$ is used in the first i levels of the tree. Formally, there is no transition $\langle \pi_1(v), t(v), \pi_1(v1) \dots \pi_1(v\#(t(v)))\rangle$, for some $v \in \mathbb{N}^*$ with $|v| < i$, such that there exists a t-run π_2 and an A-transition $\langle \pi_2(v), t(v), \pi_2(v1) \dots \pi_2(v\#(t(v)))\rangle$ satisfying $\pi_2(v) = \pi_1(v)$ and $(\pi_1(v1) \dots \pi_1(v\#(t(v)))) \hat{R} (\pi_2(v1) \dots \pi_2(v\#(t(v))))$. Since A is finitely branching, for every transition *trans* there are only finitely many A-transitions *trans*$'$ such that *trans* $P(id(id), R)$ *trans*$'$. And since $P(id(id), R)$ is transitive and reflexive, for each transition *trans* in A we have that either **1)** *trans* is maximal w.r.t. $P(id(id), R)$, or **2)** there exists at least one *trans*$'$ which is maximal w.r.t. $P(id(id), R)$ and such that *trans* $P(id(id), R)$ *trans*$'$. Thus for every state p and every symbol σ there exists a transition by σ departing from p which is still in A'. Therefore, for every i-good accepting run π on t, one easily obtains an accepting run π' which is $(i+1)$-good. In the first i levels of t, π' is identical to π. In the $(i+1)$-th level of t, we have that for any transition $trans = \langle \pi(v), t(v), \pi(v1) \dots \pi(v\#(t(v)))\rangle$, for $|v| = i$, either *trans* is $P(id(id), R)$-maximal, and so we take $\pi'(vi) := \pi(vi)$, or *trans* is not maximal in $P(id(id), R)$, i.e., there exists a transition $trans' = \langle \pi(v), t(v), \pi''(v1) \dots \pi''(v\#(t(v)))\rangle$ such that $(\pi(v1) \dots \pi(v\#(t(v)))) \hat{R} (\pi''(v1) \dots \pi''(v\#(t(v))))$, and we take $\pi'(vi) := \pi''(vi)$ for any $1 \leq i \leq \#(t(v)))$. Since $R \subset \subseteq^{\mathsf{dw}}$ and $\subseteq^{\mathsf{dw}}$ preserves the acceptance of states, in both **1)** and **2)** we have that $\forall_{n \in \mathbb{N}^*}. \pi(v) = \psi \rightarrow \pi'(v) = \psi$.

Since $\hat{\pi}$ is an accepting t-run (which is trivially 0-good), there exists an accepting t-run $\hat{\pi}'$ which is h-good, for h the height of t. Moreover, $\hat{\pi}'$ is a run in A'. ◀

Theorem 3. For every strict partial order $R \subset \subseteq^{\mathsf{up}}(id)$, it holds that $P(R, id)$ is GFP.

Proof. Let $A' = Prune(A, P(R, id))$. We will show that for every accepting run π of A on a tree t, there exists an accepting run $\hat{\pi}$ of A' on t.

Let us first define some auxiliary notation. For an accepting run π of A on a tree t, $bad(\pi)$ is the smallest subtree of t which contains *all nodes* v of t where π uses a transition of $A - A'$, i.e., a transition which is not $P(R, id)$-maximal (where by π using a transition at node v we mean that the symbol of the transition is $t(v)$, $\pi(v)$ is the left-hand side of the transition, and the vector of π-values of children of v is its right-hand side). We will use the following auxiliary claim.

(C) For every accepting run π of A on a tree t with $|bad(\pi)| > 1$, there is an accepting run π' of A on t where $bad(\pi')$ is a proper subtree of $bad(\pi)$.

To prove (C), assume that v is a leaf of $bad(\pi)$ labeled by a transition $\langle p, \sigma, r_1 \dots r_n\rangle$. By the definition of $P(R, id)$ and by the minimality of $bad(\pi)$, there exists a $P(R, id)$-maximal transition $\tau = \langle p', \sigma, r_1 \dots r_n\rangle$ where $p \subset^{\mathsf{up}}(id)\ p'$. Since $p \subset^{\mathsf{up}}(id)\ p'$, it follows from the definition of $\subset^{\mathsf{up}}(id)$ that there exists a run π' of A on t that differs from π only in labels of prefixes of v (including v itself) with $\pi'(v) = p'$. In other words, $bad(\pi')$ differs from $bad(\pi)$ only in that it does not contain a certain subtree rooted by some ancestor of v. This subtree contains at least v itself, since π' uses the $P(R, id)$-maximal transition τ to label v. The tree $bad(\pi')$ is hence a proper subtree of $bad(\pi)$, which concludes the proof of (C).

With (C) in hand, we are ready to prove the lemma. By finitely many applications of (C), starting from π, we obtain an accepting run $\hat{\pi}$ on t with $bad(\hat{\pi}) = \epsilon$ (we only need finitely many applications since $bad(\pi)$ is a finite tree, and every application of (C) yields a run with a strictly smaller *bad* subtree). Since $bad(\hat{\pi}) = \epsilon$, $\hat{\pi}$ is using only $P(R, id)$-maximal transitions. Since R and hence also

$P(R,id)$ are strict p.o., $A' = Prune(A, P(R,id))$ contains all $P(R,id)$-maximal transitions of A, which means that $\hat{\pi}$ is an accepting run of A' on t. ◀

Theorem 11. $\equiv^{\mathsf{dw}}$ is GFQ.

Proof. Let $A' := A/\equiv^{\mathsf{dw}}$. It is trivial that $L(A) \subseteq L(A')$. For the reverse inclusion, we will show by induction on the height i of t, that for any tree t, if $t \in D_{A'}([q])$ for some $[q] \in [Q]$, then $t \in D_A(q)$. This guarantees $L(A') \subseteq L(A)$ since if $[q] \in [I]$ then there is some $q' \in I$ such that $q' \equiv^{\mathsf{dw}} q$ and thus, by the definition of $\equiv^{\mathsf{dw}}$, $D_A(q') = D_A(q)$. We will take $i = 0$ and $i = 1$ as our base cases.

If $i = 0$, then t is the empty tree ϵ and thus there exists a state $[q] \in [I]$ with $[q] = [\psi]$. Thus there exists a $q' \in I$ with $q' \equiv^{\mathsf{dw}} q$. Since in a TDTA no state can be downward trace equivalent to ψ other than ψ itself, we have $[\psi] = \{[\psi]\}$ and therefore $q' \equiv^{\mathsf{dw}} q \equiv^{\mathsf{dw}} \psi$. Since $[\psi] = \{\psi\}$ we obtain $q' = \psi$. Therefore $\psi \in I$, and consequently $\epsilon \in L(A)$.

For $i = 1$, t is a leaf-node σ, for some $\sigma \in \Sigma$. By hypothesis, $t \in L(A')$. So there exists $[q] \in [I]$ such that $t \Longrightarrow_{A'} [q]$. So $\langle [q], \sigma, [\psi] \rangle \in \delta_{A'}$. Since $[\psi] = \{\psi\}$, there exists $q' \in [q]$ such that $\langle q', \sigma, \psi \rangle \in \delta_A$. Since $[q] \in [I]$ there is some $q'' \in I$ with $q'' \equiv^{\mathsf{dw}} q \equiv^{\mathsf{dw}} q'$. We have $t \in D_A(q') = D_A(q'') \subseteq L(A)$.

Let us now consider $i > 1$. Let σ be the root of the tree t, and let $t_1, t_2, \ldots, t_n$, where $n = \#(\sigma)$, denote each of the immediate subtrees of t. As we assume $t \in L(A')$, there exists $[q] \in [I]$ such that $\langle [q], \sigma, [q_1][q_2]\ldots[q_n] \rangle \in \delta_{A'}$, for some $[q_1], [q_2], \ldots, [q_n] \in [Q]$, such that $t_i \in D_{A'}([q_i])$ for every i. By the definition of $\delta_{A'}$, there are $q'_1 \in [q_1]$, $q'_2 \in [q_2]$, …, $q'_n \in [q_n]$ and $q' \in [q]$, such that $\langle q', \sigma, q'_1 q'_2 \ldots q'_n \rangle \in \delta_A$. By induction hypothesis, we obtain $t_i \in D_A(q_i)$ for every i. Since $q_i \equiv^{\mathsf{dw}} q'_i$, it follows that $t_i \in D_A(q'_i)$ for every i and thus $t \in D_A(q')$. By $q \equiv^{\mathsf{dw}} q'$, we conclude that $t \in D_A(q)$. ◀

Theorem 12. $\equiv^{\mathsf{up}}(id)$ is GFQ.

Proof. Let $\equiv := \equiv^{\mathsf{up}}(id)$ and $A' := A/\equiv$. It is trivial that $L(A) \subseteq L(A')$. For the reverse inclusion, we will show, by induction on the height h of t, that for any tree t, if $t \in D_{A'}(\hat{q})$ for some $\hat{q} \in [Q]$, then $t \in D_A(q)$ for some $q \in \hat{q}$. This guarantees $L(A') \subseteq L(A)$ since if $\hat{q} \in [I]$ then, given that $\equiv$ preserves the initial states, $q \in I$. Again, we will take $h = 0$ and $h = 1$ as our base cases. We will use a second induction inside the induction on h.

If $h = 0$, then t is the empty tree ϵ, and thus there exists a state $\hat{q} \in [I]$ with $\hat{q} = [\psi]$. There must exist some $q' \in \hat{q}$ with $q' \in I$. Since $[\psi] = \{\psi\}$, $q' \equiv \psi$. Since $\equiv$ preserves the initial states, we have $\psi \in I$ and thus $\epsilon \in L(A)$.

If $h = 1$, the tree is a leaf-node σ, for some $\sigma \in \Sigma$. By hypothesis, $t \in L(A')$. So there exists a $[q] \in [I]$ such that $t \Longrightarrow_{A'} [q]$, and so $\langle [q], \sigma, [\psi] \rangle \in \delta_{A'}$. By the definition of $\delta_{A'}$, there exist $q' \in [q]$ and $r \in [\psi]$ such that $\langle q', \sigma, r \rangle \in \delta_A$. Since $\equiv$ preserves the acceptance condition and since $[\psi] = \{\psi\}$, $r = \psi$ and thus $t \Longrightarrow_A q'$. Given that $[q] \in [I]$, there exists a state $q'' \in I$ such that $q'' \equiv q$. Since $\equiv$ preserves the initial states and $q'' \equiv q \equiv q'$, $q' \in I$ and thus $t \in L(A)$.

Let us now consider $h > 1$. As we assume $t \in L(A')$, there exists $\hat{q} \in [I]$ such that $t \Longrightarrow_{A'} \hat{q}$, and so $\langle \hat{q}, \sigma, \hat{q}_1 \ldots \hat{q}_n \rangle \in \delta_{A'}$, for $n = \#\sigma$ and some $\hat{q}_1, \ldots, \hat{q}_n \in [Q]$ such that $t_i \in D_{A'}(\hat{q}_i)$ for every $i : 0 \leq i \leq n$, where t_i are the subtrees of t. We define the following auxiliary notion: a transition $\langle r, \sigma, r_1 \ldots r_n \rangle$ of A satisfying $r \in I$ and $\forall_{1 \leq k \leq n}. r_k \in \hat{q}_k$ is said to be j-good iff $\forall_{1 \leq k \leq j}. t_k \in D_A(r_k)$. We will use induction on j to show that there is a j-good transition for any j, which implies that there is some initial state r^* such that $t \in D_A(r^*)$.

The base case is $j = 0$. By the definition of $\delta_{A'}$ and the fact that $\langle \hat{q}, \sigma, \hat{q}_1 \ldots \hat{q}_n \rangle \in \delta_{A'}$, there exist $q_1 \in \hat{q}_1$, …, $q_n \in \hat{q}_n$ and $q \in \hat{q}$ such that $\langle q, \sigma, q_1 \ldots q_n \rangle \in \delta_A$. This transition is trivially 0-good.

Now we show the induction step. By the hypothesis of the inner induction on j, there exists a transition $trans = \langle r, \sigma, r_1 \ldots r_n \rangle$, for some t-run π such that $\pi(\varepsilon) = r$ and $\forall_{1 \leq v \leq n}. \pi(v) = r_v$, which is j-good for some $j \geq 0$. By the hypothesis of the outer induction on h, there are $q'_i \in \hat{q}_i$ such that $t_i \in D_A(q'_i)$, for $i : 1 \leq i \leq n$. In particular, there is a $q'_{j+1} \in \hat{q}_{j+1}$ such that $t_{j+1} \in D_A(q'_{j+1})$.

By the definition of j-good, we also have that $r_{j+1} \in \hat{q}_{j+1}$. Consequently, $r_{j+1} \equiv q'_{j+1}$. Consider now $\hat{t}$, the open tree of height 2 such that $\hat{t}(\varepsilon) = t(\varepsilon) = \sigma$ and $\forall_{v \in \{1,\ldots,j,j+2,\ldots,n\}}.\, \hat{t}(v) = t(v)$. And consider the $\hat{t}$-run $\hat{\pi}$, a run such that $\hat{\pi}(\varepsilon) = \pi(\varepsilon) = r$ and $\forall_{1 \leq v \leq n}.\, \hat{\pi}(v) = \pi(v) = r_v$. Thus $\hat{\pi}$ has exactly one dangling leaf, $j+1$. From $r_{j+1} \equiv q'_{j+1}$ it follows that there exists a $\hat{t}$-run $\hat{\pi}'$ with exactly one dangling leaf, $v'(j+1)$, and such that $\hat{\pi}'(v'(j+1)) = q'_{j+1}$ and **1)** $\forall_{i \neq j+1}.\, \hat{\pi}'(v'i) = \hat{\pi}(i) = r_i$. **2)** $\forall_{v \in \mathbb{N}^*}.\, (\hat{\pi}(v) = \psi \rightarrow \hat{\pi}'(v) = \psi) \wedge (\hat{\pi}(v) \in I \rightarrow \hat{\pi}'(v) \in I)$. This run must use some transition $trans' = \langle r', \sigma, r_1 \ldots r_j q'_{j+1} r_{j+2} \ldots r_n \rangle$ in A, with r' some initial state, since $r \in I$[1]. Since *trans* is j-good and $t_{j+1} \in D_A(q'_{j+1})$, we have that $trans'$ is $(j+1)$-good. Therefore, there exists a j-good transition for any $j \geq 0$. This concludes the induction on j. As a consequence we obtain that there exists a j-good transition for $j = n$. Then $t \in D_A(r^*)$, for some $r^* \in I$, and this concludes the induction on h. Thus $L(A) = L(A')$. ◀

[1] Note that it is thanks to the fact that the inducing relation in $\equiv^{\mathsf{up}}(id)$ is the identity that $trans'$ preserves the first j arriving states of *trans* exactly. This would not happen if we were using a different inducing relation - take for instance the case of $\equiv^{\mathsf{up}}(\sqsubseteq^{\mathsf{dw}})$, which is not GFQ.

C More Data from the Experiments

TA name	$\#Q_i$	$\#Delta_i$	$\#Q_f$	$\#Delta_f$	Q reduction	Delta reduction	Time(s)
A0087	88	1015	12	23	13.6	2.3	0.060
A0065	66	562	11	23	16.7	4.1	0.027
A0057	58	245	24	58	41.4	23.7	0.023
A0130	131	1504	11	23	8.3	1.5	0.043
A0058	59	257	25	65	42.4	25.3	0.022
A0177	178	1781	26	58	14.6	3.3	0.143
A0063	64	571	11	23	17.2	4.0	0.028
A0055	56	182	27	73	48.2	40.1	0.018
A0060	61	244	32	111	52.5	45.5	0.036
A0059	60	263	24	59	40.0	22.4	0.024
A0089	90	1006	12	21	13.3	2.1	0.064
A0111	112	1790	11	42	9.8	2.39	0.138
A0083	84	713	26	65	31.0	9.1	0.067
A0054	55	241	28	93	50.9	38.6	0.027
A0056	57	230	24	55	42.1	23.9	0.019
A0126	127	1196	11	23	8.7	1.9	0.083
A0080	81	672	26	58	32.1	8.6	0.057
A0120	121	1367	12	21	9.9	1.5	0.069
A0062	63	276	32	112	50.8	40.6	0.032
A0082	83	713	26	65	31.3	9.1	0.063
A0117	118	2088	25	106	21.2	5.1	0.201
A0086	87	1402	26	112	29.9	8.0	0.118
A0064	65	574	11	23	16.9	4.0	0.024
A0172	173	1333	11	23	6.4	1.7	0.105
A0053	54	159	27	66	50.0	41.5	0.016
A0070	71	622	11	23	15.5	3.7	0.018
A0088	89	1027	12	23	13.5	2.2	0.063
Average	87.07	816.04			26.97	13.94	0.059

Table 1 Results on minimizing the 27 moderate-sized tree automata (from `libvata`'s regular model checking examples) with our $Heavy(1,1)$ algorithm. The columns give the name of each automaton, $\#Q_i$: its original number of states, $\#Delta_i$: its original number of transitions, $\#Q_f$: the number of states after minimization, $\#Delta_f$: the number of transitions after minimization, the reduction ratio for states in percent $100 * \#Q_f/\#Q_i$ (smaller is better), the reduction ratio for transitions in percent $100 * \#Delta_f/\#Delta_i$ (smaller is better), and the computation time in seconds. Note that the reduction ratios for transitions are smaller than the ones for states, i.e., the automata get not only smaller but also sparser. Experiments run on Intel 3.20GHz i5-3470 CPU.

Table 2 shows the results on minimizing the 66 larger automata (those not called moderate-sized) from `libvata`'s regular model checking examples with our $Heavy(1,1)$ algorithm. The columns give the name of each automaton, $\#Q_i$: original number of states, $\#Delta_i$: original number of transitions, $\#Q_f$: states after minimization, $\#Delta_f$: transitions after minimization, the reduction ratio for states in percent $100 * \#Q_f/\#Q_i$ (smaller is better), the reduction ratio for transitions in percent $100 * \#Delta_f/\#Delta_i$ (smaller is better), and the computation time in seconds. Note that the reduction ratios for transitions are smaller than the ones for states, i.e., the automata get sparser. Experiments run on Intel 3.20GHz i5-3470 CPU.

TA name	$\#Q_i$	$\#Delta_i$	$\#Q_f$	$\#Delta_f$	Q reduction	Delta reduction	Time(s)
A494	495	8533	12	21	2.42	0.25	3.08
A1404	1405	18839	24	52	1.71	0.28	5.43
A2003	2004	30414	24	52	1.20	0.17	14.2
A312	313	3367	11	23	3.51	0.68	0.26
A756	757	8884	26	58	3.43	0.65	2.23
A646	647	6054	19	34	2.94	0.56	0.98
A837	838	13038	11	23	1.31	0.18	5.20
A676	677	11043	34	76	5.02	0.69	6.52
A881	882	15575	12	21	1.36	0.13	3.69
A0483	484	5592	25	55	5.17	0.98	0.95
A323	324	6199	26	112	8.02	1.81	1.58
A691	692	11047	34	76	4.91	0.69	6.62
A980	981	21109	12	21	1.22	0.10	4.69
A315	316	3387	24	52	7.59	1.54	0.66
A728	729	11903	12	21	1.65	0.18	3.18
A348	349	3681	11	23	3.15	0.62	0.34
A0369	370	4134	24	52	6.49	1.26	0.43
A667	668	8131	26	58	3.89	0.71	2.13
A0310	311	3343	24	52	7.72	1.56	0.76
A670	671	11021	34	76	5.07	0.69	6.34
A498	499	8612	12	21	2.40	0.24	3.20
A1003	1004	21302	12	21	1.20	0.10	4.06
A496	497	8618	12	21	2.41	0.24	2.84
A620	621	9218	12	21	1.93	0.23	1.55
A328	329	3517	26	58	7.90	1.65	0.71
A334	335	3936	11	23	3.28	0.58	0.69
A569	570	8351	26	58	4.56	0.69	1.76
A0312	313	3367	11	23	3.51	0.68	0.28
A0348	349	3681	11	23	3.15	0.62	0.36
A339	340	5596	12	21	3.53	0.38	0.51
A322	323	3651	35	100	10.84	2.74	0.88
A491	492	8708	12	21	2.44	0.24	3.65
A354	355	3522	24	52	6.76	1.48	0.96
A700	701	11245	36	81	5.14	0.72	7.35
A369	370	4134	24	52	6.49	1.26	0.42
A355	356	3895	25	55	7.02	1.41	0.53
A532	533	8867	12	23	2.25	0.26	3.12
A483	484	5592	25	55	5.17	0.98	0.93
A678	679	11172	26	56	3.83	0.50	5.77
A390	391	5390	11	23	2.81	0.43	1.14
A493	494	7523	12	21	2.43	0.28	0.66
A387	388	4117	24	52	6.19	1.26	0.68
A589	590	9606	12	21	2.03	0.22	3.15
A1306	1307	19699	25	55	1.91	0.28	3.98
A723	724	9376	26	58	3.59	0.62	2.53
A320	321	3623	26	65	8.10	1.79	0.75
A0246	247	2944	11	42	4.45	1.43	0.48
A694	695	11191	34	76	4.89	0.68	6.59
A501	502	8632	12	21	2.39	0.24	3.19
A329	330	5961	24	100	7.27	1.68	1.57
A335	336	3738	26	58	7.74	1.55	0.80
A679	680	11032	34	76	5.00	0.69	6.46
A400	401	5461	11	23	2.74	0.42	1.41
A703	704	11255	34	76	4.83	0.68	6.87
A692	693	11066	34	76	4.91	0.69	6.70
A673	674	11157	25	55	3.71	0.49	5.86
A301	302	4468	12	21	3.97	0.47	0.32
A488	489	8493	12	21	2.45	0.25	2.81
A447	448	7924	12	23	2.68	0.29	2.58
A701	702	11244	36	83	5.13	0.74	7.11
A321	322	3407	24	52	7.45	1.53	0.76
A487	488	4891	16	28	3.28	0.57	0.44
A689	690	11207	31	71	4.49	0.63	6.42
A693	694	11188	34	76	4.90	0.68	6.59
A695	696	11070	34	76	4.89	0.69	6.62
A489	490	8516	12	21	2.45	0.25	2.93
Average	573.7	8582.7			4.22	0.73	3.00

Table 2 Results on minimizing the 66 larger automata (those not called moderate-sized) from `libvata`.

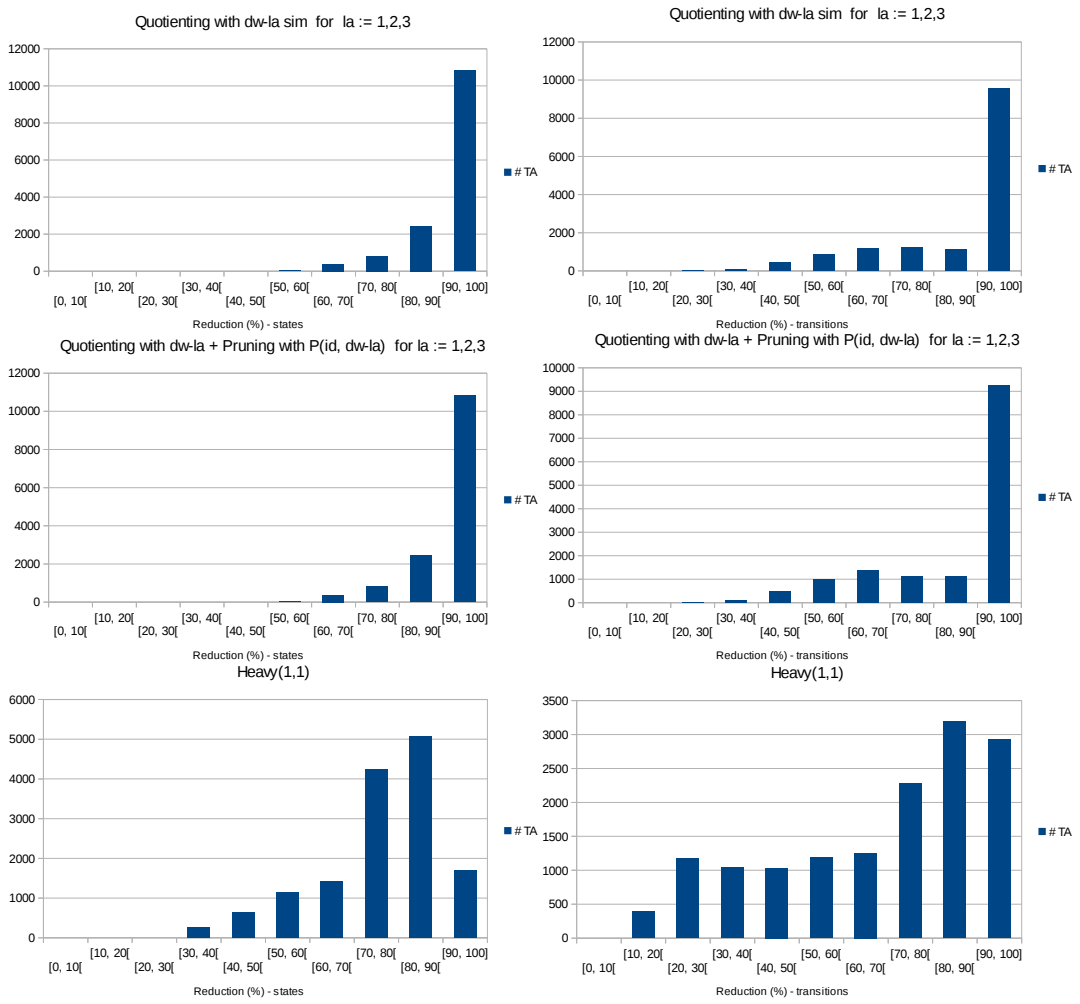


Figure 10 Minimization of 14498 tree automata from the Forester tool [21], by methods RUQ (top), RUQP (middle) and Heavy (bottom). A bar of height h at an interval $[x, x+10[$ means that h of the 14498 automata were reduced to a size between $x\%$ and $(x+10)\%$ of their original size. The reductions in the numbers of states/transitions are shown on the left/right, respectively. Using higher lookaheads made hardly any difference in this sample set.

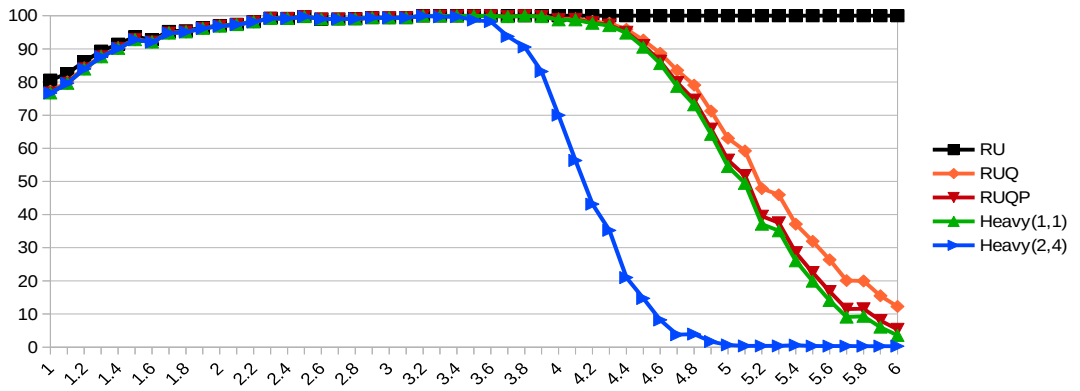


Figure 11 Minimization of Tabakov-Vardi random tree automata with $n = 100, s = 2$ and $ad = 0.8$. The x-axis gives the transition density td, and the y-axis gives the average number of transitions after minimization (in percent of the original number) with the various methods (smaller is better). Each data point is the average of 400 random automata. Note that Heavy(2,4) minimizes much better than Heavy(1,1) for $td \geq 3.5$.