Computing Simulations over Tree Automata
(Efficient Techniques for Reducing Tree Automata)

Parosh A. Abdulla[1], Ahmed Bouajjani[2], Lukáš Holík[3], Lisa Kaati[1], and Tomáš Vojnar[3]

[1] University of Uppsala, Sweden, email: {parosh,lisa.kaati}@it.uu.se
[2] LIAFA, University Paris 7, France, email: abou@liafa.jussieu.fr
[3] FIT, Brno University of Technology, Czech Rep., email: {holik,vojnar}@fit.vutbr.cz

Abstract. We address the problem of computing simulation relations over tree automata. In particular, we consider downward and upward simulations on tree automata, which are, loosely speaking, analogous to forward and backward relations over word automata. We provide simple and efficient algorithms for computing these relations based on a reduction to the problem of computing simulations on labelled transition systems. Furthermore, we show that downward and upward relations can be combined to get relations compatible with the tree language equivalence, which can subsequently be used for an efficient size reduction of nondeterministic tree automata. This is of a very high interest, for instance, for symbolic verification methods such as regular model checking, which use tree automata to represent infinite sets of reachable configurations. We provide experimental results showing the efficiency of our algorithms on examples of tree automata taken from regular model checking computations.

1 Introduction

Tree automata are widely used for modelling and reasoning about various kinds of structured objects such as syntactical trees, structured documents, configurations of complex systems, algebraic term representations of data or computations, etc. (see [9]). For instance, in the framework of regular model checking, tree automata are used to represent and manipulate sets of configurations of infinite-state systems such as parameterized networks of processes with a tree-like topology, or programs with dynamic linked data-structures [7, 3, 5, 6].

In the above context, checking language equivalence and reducing automata wrt. the language equivalence is a fundamental issue, and performing these operations efficiently is crucial for all practical applications of tree automata. Computing a minimal canonical tree automaton is, of course, possible, but it requires determinisation, which may lead to an exponential blow-up in the size of the automaton. Therefore, even if the resulting automaton can be small, we may not be able to compute it in practice due to the very expensive determinisation step, which is, indeed, a major bottleneck when using canonical tree automata.

A reasonable and pragmatic approach is to consider a notion of equivalence that is stronger than language equivalence, but which can be checked efficiently, using a polynomial algorithm. Here, a natural trade-off between the strength of the considered equivalence and the cost of its computation arises. In the case of word automata, an equivalence which is widely considered as a good trade-off in this sense is simulation equivalence. It can be checked in polynomial time, and efficient algorithms have been designed for this purpose (see, e.g., [10, 14]). These algorithms make the computation of simulation equivalence quite affordable even in comparison with the one of bisimulation, which is cheaper [13], but which is also stronger, and therefore leads in general to less significant reductions in the sizes of the automata.

In this paper, we study notions of entailment and equivalence between tree automata, which are suitable in the sense discussed above, and we also provide efficient algorithms for their computation.

We start by considering a basic notion of tree simulation, called *downward simulation*, corresponding to a natural extension of the usual notion of simulation defined on *or*-structures to *and-or* structures. This relation can be shown to be compatible with the tree language equivalence.

The second notion of simulation that we consider, called *upward simulation*, corresponds intuitively to a generalisation of the notion of backward simulation to and-or structures. The definition of an upward simulation is parametrised by a downward simulation: Roughly speaking, two states q and q' are upward similar if whenever one of them, say q, considered within some vector $(q_1, \ldots, q_n)$ at position i, has an upward transition to some state s, then q' appears at position i of some vector $(q'_1, \ldots, q'_n)$ that has also an upward transition to a state s', which is upward similar to s, and moreover, for each position $j \neq i$, q_j is downward similar to q'_j.

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. However, we show an interesting fact that when we restrict ourselves to upward relations compatible with the set of final states of automata, the downward and upward simulation equivalences can be *combined* in such a way that they give rise to a new equivalence relation which is compatible with the tree language equivalence. This combination is not trivial. It is based on the idea that two states q_1 and q_2 may have different tree languages and different context languages, but for every t in the tree language of one of them, say q_1, and every C in the context language of the other, here q_2, the tree $C[t]$ (where t is plugged into C) is recognised by the automaton. The combined relation is coarser than (or, in the worst case, as coarse as) the downward simulation and according to our practical experiments, it usually leads to significantly better reductions of the automata.

In this way, we obtain two candidates for simulation-based equivalences for use in automata reduction. Then, we consider the issue of designing efficient algorithms for computing these relations. A deep examination of downward and upward simulation equivalences shows that they can be computed using essentially the same algorithmic pattern. Actually, we prove that, surprisingly, computing downward and upward tree simulations can be reduced in each case to computing simulations on standard labelled transition systems. These reductions provide a simple and elegant way of solving in a uniform way the problem of computing tree simulations by reduction to computing simulations in the word case. The best known algorithm for solving the latter problem, published recently in [14], considers simulation relations defined on Kripke structures. The use of this algorithm requires its adaptation to labelled transition systems. We provide such an adaptation and we provide also a proof for this algorithm which can be seen as an alternative, more direct, proof of the algorithm of [14]. The combination of our reductions with the labelled transition systems-based simulation algorithm leads to efficient algorithms for our equivalence relations on tree automata, whose precise complexities are also analysed in the paper.

We have implemented our algorithms and performed experiments on automata computed in the context of regular tree model checking (corresponding to representations of the set of reachable configurations of parametrised systems). The experiments show that, indeed, the relations proposed in this paper provide significant reductions of these automata and that they perform better than (existing) bisimulation-based reductions [11].

Related work As far as we know, this is the first work which addresses the issue of computing simulation relations for tree automata. The downward and upward simulation relations considered in this work have been introduced first in [4] where they have been used for proving soundness of some acceleration techniques used in the context of regular tree model checking. However, the problem of computing these relations has not been addressed in that paper. A form of combining downward and upward relations has also been defined in [4]. However, the combinations considered in that paper require some restrictions which are computationally difficult to check and that are not considered in this work. Bisimulations on tree automata have been considered in [2, 11]. The notion of a backward bisimulation used in [11] corresponds to what can be called a downward bisimulation in our terminology.

Outline The rest of the paper is organised as follows. In the next section, we give some preliminaries on tree automata, labelled transition systems, and simulation relations. Section 3 describes an algorithm for checking simulation on labelled transition systems. In Section 4 resp. Section 5, we translate downward resp. upward simulation on tree automata into corresponding simulations on labelled transition systems. Section 6 gives methods for reducing tree automata based on equivalences derived form downward and upward simulation. In Section 7, we report some experimental results. Finally, we give conclusions and directions for future research in Section 8.

Remark For space reasons, all proofs are deferred to [1].

2 Preliminaries

In this section, we introduce some preliminaries on trees, tree automata, and labelled transition systems (LTS). In particular, we recall two simulation relations defined on tree automata in [4], and the classical (word) simulation relation defined on LTS. Finally, we will describe an encoding which we use in our algorithms to describe pre-order relations, e.g., simulation relations.

For an equivalence relation $\equiv$ defined on a set Q, we call each equivalence class of $\equiv$ a *block*, and use $Q/\equiv$ to denote the set of blocks in $\equiv$.

Trees A *ranked alphabet* Σ is a set of symbols together with a function $Rank : \Sigma \to \mathbb{N}$. For $f \in \Sigma$, the value $Rank(f)$ is said to be the *rank* of f. 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 an 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 $p \in dom(t)$, if $Rank(t(p)) = n > 0$, then $\{i \mid pi \in dom(t)\} = \{1, \ldots, n\}$.

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

Tree Automata A (finite, non-deterministic, bottom-up) *tree automaton* (TA) is a 4-tuple $A = (Q, \Sigma, \Delta, F)$ 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 $((q_1, \ldots, q_n), f, q)$ where $q_1, \ldots, q_n, q \in Q$, $f \in \Sigma$, and $Rank(f) = n$. We use $(q_1, \ldots, q_n) \xrightarrow{f} q$ to denote that $((q_1, \ldots, q_n), f, q) \in \Delta$. In the special case where $n = 0$, we speak about the so-called *leaf rules*, which we sometimes abbreviate as $\xrightarrow{f} q$. We use $Lhs(A)$ to denote the set of *left-hand sides* of rules, i.e., the set of tuples of the form $(q_1, \ldots, q_n)$ where $(q_1, \ldots, q_n) \xrightarrow{f} q$ for some f and q. Finally, we denote by $Rank(A)$ the smallest $n \in \mathbb{N}$ such that $n \geq m$ for each $m \in \mathbb{N}$ where $(q_1, \ldots, q_m) \in Lhs(A)$ for some $q_i \in Q$, $1 \leq i \leq m$.

A *run* of 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 $q = \pi(p)$, we have that if $q_i = \pi(pi)$ for $1 \leq i \leq n$, then Δ has a rule $(q_1, \ldots, q_n) \xrightarrow{t(p)} q$. We write $t \overset{\pi}{\Longrightarrow} q$ to denote that π is a run of A over t such that $\pi(\varepsilon) = q$. We use $t \Longrightarrow q$ to denote that $t \overset{\pi}{\Longrightarrow} q$ for some run π. The *language* of a state $q \in Q$ is defined by $L(q) = \{t \mid t \Longrightarrow q\}$, while the *language* of A is defined by $L(A) = \bigcup_{q \in F} L(q)$.

Labelled Transition Systems A (finite) *labelled transition system (LTS)* is a tuple $T = (S, \mathcal{L}, \to)$ where S is a finite set of states, $\mathcal{L}$ is a finite set of labels, and $\to \subseteq S \times \mathcal{L} \times S$ is a transition relation.

Given an LTS $T = (S, \mathcal{L}, \to)$, a label $a \in \mathcal{L}$, and two states $q, r \in S$, we denote by $q \xrightarrow{a} r$ the fact that $(q, a, r) \in \to$. We define the set of *a-predecessors* of a state r as $pre_a(r) = \{q \in S \mid q \xrightarrow{a} r\}$. Given $X, Y \subseteq S$, we denote $pre_a(X)$ the set $\bigcup_{s \in X} pre_a(s)$, we write $q \xrightarrow{a} X$ iff $q \in pre_a(X)$, and $Y \xrightarrow{a} X$ iff $Y \cap pre_a(X) \neq \emptyset$.

Simulations For a tree automaton $A = (Q, \Sigma, \Delta, F)$, a *downward simulation* D is a binary relation on Q such that if $(q, r) \in D$ and $(q_1, \ldots, q_n) \xrightarrow{f} q$, then there are $r_1, \ldots, r_n$ such that $(r_1, \ldots, r_n) \xrightarrow{f} r$ and $(q_i, r_i) \in D$ for each i such that $1 \leq i \leq n$. It is easy to show [4] that any downward simulation can be closed under reflexivity and transitivity. Moreover, there is a unique maximal downward simulation over a given tree automaton, which we denote as $\preccurlyeq_{down}$ in the sequel.

Given a TA $A = (Q, \Sigma, \Delta, F)$ and a downward simulation D, an *upward simulation* U induced by D is a binary relation on Q such that if $(q, r) \in U$ and $(q_1, \ldots, q_n) \xrightarrow{f} q'$ with $q_i = q$, $1 \leq i \leq n$, then there are $r_1, \ldots, r_n, r'$ such that $(r_1, \ldots, r_n) \xrightarrow{f} r'$ where $r_i = r$, $(q', r') \in U$, and $(q_j, r_j) \in D$ for each j such that $1 \leq j \neq i \leq n$. In [4], it is shown that any upward simulation can be closed under reflexivity and transitivity. Moreover, there is a unique maximal upward simulation with respect to a fixed downward simulation over a given tree automaton, which we denote as $\preccurlyeq_{up}$ in the sequel.

Given an *initial* pre-order $I \subseteq Q \times Q$, it can be shown that there are unique maximal downward as well as upward simulations included in I on the given TA, which we denote $\preccurlyeq_x^I$ in the sequel, for $x \in \{down, up\}$. Further, we use $\cong_x$ to denote the equivalence relation $\preccurlyeq_x \cap \preccurlyeq_x^{-1}$ on Q for $x \in \{down, up\}$. Likewise, we define the equivalence relations $\cong_x^I$ for an initial pre-order I on Q and $x \in \{down, up\}$.

For an LTS $T = (S, \mathcal{L}, \rightarrow)$, a *(word) simulation* is a binary relation R on S such that if $(q,r) \in R$ and $q \xrightarrow{a} q'$, then there is an r' with $r \xrightarrow{a} r'$ and $(q',r') \in R$. In a very similar way as for simulations on trees, it can be shown that any given simulation on an LTS can be closed under reflexivity and transitivity and that there is a unique maximal simulation on the given LTS, which will we denote by $\preccurlyeq$. Moreover, given an *initial* pre-order $I \subseteq S \times S$, it can be shown that there is a unique maximal simulation included in I on the given LTS, which we denote $\preccurlyeq^I$ in the sequel. We use $\cong$ to denote the equivalence relation $\preccurlyeq \cap \preccurlyeq^{-1}$ on S and consequently $\cong^I$ to denote $\preccurlyeq^I \cap (\preccurlyeq^I)^{-1}$.

Encoding Let S be a set. A *partition-relation pair* over S is a pair (P, Rel) where (1) $P \subseteq 2^S$ is a partition of S (i.e., $S = \cup_{B \in P} B$, and for all $B, C \in P$, if $B \neq C$, then $B \cap C = \emptyset$), and (2) $Rel \subseteq P \times P$. We say that a partition-relation pair (P, Rel) over S *induces* (or defines) the relation $\delta = \bigcup_{(B,C) \in Rel} B \times C$.

Let $\preceq$ be a pre-order defined on a set S, and let $\equiv$ be the equivalence $\preceq \cap \preceq^{-1}$ defined by $\preceq$. The pre-order $\preceq$ can be represented—which we will use in our algorithms below—by a partition-relation pair (P, Rel) over S such that $(B,C) \in Rel$ iff $s_1 \preceq s_2$ for all $s_1 \in B$ and $s_2 \in C$. In this representation, if the partition P is as coarse as possible (i.e., such that $s_1, s_2 \in B$ iff $s_1 \equiv s_2$), then, intuitively, the elements of P are blocks of $\equiv$, while Rel reflects the partial order on P corresponding to $\preceq$.

3 Computing Simulations on Labelled Transition Systems

We now introduce an algorithm to compute the (unique) maximal simulation relation $\preccurlyeq^I$ on an *LTS* for a given initial pre-order I on states. Our algorithm is a re-formulation of the algorithm proposed in [14] for computing simulations over *Kripke structures*.

3.1 An Algorithm for Computing Simulations on LTS

For the rest of this section, we assume that we are given an LTS $T = (S, \mathcal{L}, \rightarrow)$ and an initial pre-order $I \subseteq S \times S$. We will use Algorithm 1 to compute the maximum simulation $\preccurlyeq^I \subseteq S \times S$ included in I. In the algorithm, we use the following notation. Given $\rho \subseteq S \times S$ and an element $q \in S$, we denote $\rho(q)$ the set $\{r \in S \mid (q,r) \in \rho\}$.

The algorithm performs a number of iterations computing a sequence of relations, each induced by a partition-relation pair (P, Rel). During each iteration, the states belonging to a block $B' \in P$ are those which are currently assumed as capable of simulating those from any B with $(B, B') \in Rel$. The algorithm starts with an initial partition-relation pair (P_{init}, Rel_{init}) that induces the initial pre-order I on S. The partition-relation pair is then gradually refined by splitting blocks of the partition P and by restricting the relation Rel on P. When the algorithm terminates, the final partition-relation pair (P_{sim}, Rel_{sim}) induces the required pre-order $\preccurlyeq^I$.

The refinement performed during the iterations consists of splitting the blocks in P and then updating the relation Rel accordingly. For this purpose, the algorithm maintains a set $Remove_a(B)$ for each $a \in \mathcal{L}$ and $B \in P$. Such a set contains states that do not have an

a-transition going into states that are in B nor to states of any block B' with $(B,B') \in Rel$. Clearly, the states in $Remove_a(B)$ cannot simulate states that have an a-transition going into $\bigcup_{(B,B')\in Rel} B'$. Therefore, for any $Remove_a(B) \neq \emptyset$, we can split each block $C \in P$ to $C \cap Remove_a(B)$ and $C \setminus Remove_a(B)$. This is done using the function *Split* on line 6.

After performing the *Split* operation, we update the relation *Rel* and the *Remove* sets. This is carried out in two steps. First, we compute an approximation of the next values of *Rel* and *Remove*. More precisely, after a split, all *Rel* relations between the original "parent" blocks of states are inherited to their "children" resulting from the split (line 8)—the notation $\mathtt{parent}_{P_{\mathtt{prev}}}(C)$ refers to the parent block from which C arose within the split. On line 10, the remove sets are then inherited from parent blocks to their children. To perform the second step, we observe that the inheritance of the original relation *Rel* on parent blocks to the children blocks is not consistent with the split we have just performed. Therefore, on line 14, we subsequently prune *Rel* such that blocks C that have an a-transition going into B states cannot be considered as simulated by blocks D which do not have an a-transition going into $\bigcup_{(B,B')\in Rel} B'$—notice that due to the split that we have performed, the D blocks are now included in *Remove*. This pruning can then cause a necessity of further refinements as the states that have some b-transition into a D block (that was freshly found not to simulate C), but not to C nor any block that is still viewed as capable of simulating C, have to stop simulating states that can go into $\bigcup_{(C,C')\in Rel} C'$. Therefore, such states are added into $Remove_b(C)$ on line 17.

3.2 Correctness and Complexity of the Algorithm

In the rest of the section, we assume that Algorithm 1 is applied on an LTS $T = (S, \mathcal{L}, \rightarrow)$ with an initial partition-relation pair (P_{init}, Rel_{init}). The correctness of the algorithm is formalised in Theorem 1.

Theorem 1. *Suppose that I is the pre-order induced by (P_{init}, Rel_{init}). Then, Algorithm 1 terminates and the final partition-relation pair (P_{sim}, Rel_{sim}) computed by it induces the simulation relation $\preccurlyeq^I$, and, moreover, $P_{sim} = S/\cong^I$.*

A similar correctness result is proved in [14] for the algorithm on Kripke structures, using notions from the theory of abstract interpretation. In [1], we provide an alternative, more direct proof, which is, however, beyond the space limitations of this paper. Therefore, we will only mention the key idea behind the termination argument. In particular, the key point is that if we take any block B from P_{init} and any $a \in \mathcal{L}$, if B or any of its children B', which arises by splitting, is repeatedly selected to be processed by the while loop on line 3, then the $Remove_a(B)$ (or $Remove_a(B')$) sets can never contain a single state $s \in S$ at an iteration i of the while loop as well as on a later iteration j, $j > i$. Therefore, as the number of possible partitions as well as the number of states is finite, the algorithm must terminate.

The complexity of the algorithm is equal to that of the original algorithm from [14], up to the new factor $\mathcal{L}$ that is not present in [14] (or, equivalently, $|\mathcal{L}| = 1$ in [14]). The complexity is stated in Theorem 2.

Theorem 2. *Algorithm 1 has time complexity $O(|\mathcal{L}|.|P_{sim}|.|S| + |P_{sim}|.|\rightarrow|)$ and space complexity $O(|\mathcal{L}|.|P_{sim}|.|S|)$.*

A proof of Theorem 2, based on a similar reasoning as in [14], can be found in [1]. Here, let us just mention that the result expects the input LTS and the initial partition-relation

Algorithm 1: Computing simulations on states of an LTS

Input: An LTS $T = (S, \mathcal{L}, \rightarrow)$, an initial partition-relation pair (P_{init}, Rel_{init}) on S inducing a pre-order $I \subseteq S \times S$.

Data: A partition-relation pair (P, Rel) on S, and for each $B \in P$ and $a \in \mathcal{L}$, a set $Remove_a(B) \subseteq S$.

Output: The partition-relation pair (P_{sim}, Rel_{sim}) inducing the maximal simulation on T contained in I.

```
    /* initialisation */
1   $(P, Rel) \leftarrow (P_{init}, Rel_{init})$;
2   forall $a \in \mathcal{L}, B \in P$ do  $Remove_a(B) \leftarrow S \setminus pre_a(\bigcup Rel(B))$;
    /* computation */
3   while $\exists a \in \mathcal{L}\,.\, \exists B \in P.\ Remove_a(B) \neq \emptyset$ do
4       $Remove \leftarrow Remove_a(B); Remove_a(B) \leftarrow \emptyset$;
5       $P_{\mathsf{prev}} \leftarrow P; B_{\mathsf{prev}} \leftarrow B; Rel_{\mathsf{prev}} \leftarrow Rel$;
6       $P \leftarrow Split(P, Remove)$;
7       forall $C \in P$ do
8           $Rel(C) \leftarrow \{D \in P \mid D \subseteq \bigcup Rel_{\mathsf{prev}}(\mathsf{parent}_{P_{\mathsf{prev}}}(C))\}$;
9           forall $b \in \mathcal{L}$ do
10              $Remove_b(C) \leftarrow Remove_b(\mathsf{parent}_{P_{\mathsf{prev}}}(C))$
11      forall $C \in P.\ C \xrightarrow{a} B_{\mathsf{prev}}$ do
12          forall $D \in P.\ D \subseteq Remove$ do
13              if $(C, D) \in Rel$ then
14                  $Rel \leftarrow Rel \setminus \{(C, D)\}$;
15                  forall $b \in \mathcal{L}$ do
16                      forall $r \in pre_b(D) \setminus pre_b(\bigcup Rel(C))$ do
17                          $Remove_b(C) \leftarrow Remove_b(C) \cup \{r\}$
18  $(P_{sim}, Rel_{sim}) \leftarrow (P, Rel)$;
```

pair be encoded in suitable data structures. This fact is important for the complexity analyses presented later on as they build on using Algorithm 1.

In particular, the input LTS is represented as a list of records about its states—we call this representation as the *state-list* representation of the LTS. The record about each state $s \in S$ contains a list of nonempty $pre_a(s)$ sets[4], each of them encoded as a list of its members. The partition P_{init} (and later any of its refinements) is encoded as a doubly-linked list (DLL) of blocks. Each block is represented as a DLL of (pointers to) states of the block. The relation Rel_{init} (and later any of its refinements) is encoded as a Boolean matrix $P_{init} \times P_{init}$.

4 Computing Downward Simulation

In this section, we describe algorithms for computing downward simulation on tree automata. Our approach consists of two parts: (1) we translate the maximal downward simulation problem over tree automata into a corresponding maximal simulation

[4] We use a list rather than an array having an entry for each $a \in \mathcal{L}$ in order to avoid a need to iterate over alphabet symbols for which there is no transition.

problem over LTSs (i.e., basically word automata), and (2) we compute the maximal word simulation on the obtained LTS using Algorithm 1. Below, we describe how the translation is carried out.

We translate the downward simulation problem on a TA $A = (Q, \Sigma, \Delta, F)$ to the simulation problem on a derived LTS $A^\bullet$. Each state and each left hand side of a rule in A is represented by one state in $A^\bullet$, while each rule in A is simulated by a set of rules in $A^\bullet$. Formally, we define $A^\bullet = (Q^\bullet, \Sigma^\bullet, \Delta^\bullet)$ as follows:

– The set $Q^\bullet$ contains a state $q^\bullet$ for each state $q \in Q$, and it also contains a state $(q_1, \ldots, q_n)^\bullet$ for each $(q_1, \ldots, q_n) \in Lhs(A)$.
– The set $\Sigma^\bullet$ contains each symbol $a \in \Sigma$ and each index $i \in \{1, 2, \ldots, n\}$ where n is the maximal rank of any symbol in Σ.
– For each transition rule $(q_1, \ldots, q_n) \xrightarrow{f} q$ of A, the set $\Delta^\bullet$ contains both the transition $q^\bullet \xrightarrow{f} (q_1, \ldots, q_n)^\bullet$ and transitions $(q_1, \ldots, q_n)^\bullet \xrightarrow{i} q_i^\bullet$ for each $i : 1 \leq i \leq n$.
– The sets $Q^\bullet$, $\Sigma^\bullet$, and $\Delta^\bullet$ do not contain any other elements.

The following theorem shows correctness of the translation.

Theorem 3. *For all $q, r \in Q$, we have $q^\bullet \preccurlyeq r^\bullet$ iff $q \preccurlyeq_{down} r$.*

Due to Theorem 3, we can compute the simulation relation $\preccurlyeq_{down}$ on Q by constructing the LTS $A^\bullet$ and running Algorithm 1 on it with the initial partition-relation pair being simply $(P^\bullet, Rel^\bullet) = (\{Q^\bullet\}, \{(Q^\bullet, Q^\bullet)\})$[5].

4.1 Complexity of Computing the Downward Simulation

The complexity naturally consists of the price of compiling a given TA $A = (Q, \Sigma, \Delta, F)$ into its corresponding LTS $A^\bullet$, the price of building the initial partition-relation pair $(P^\bullet, Rel^\bullet)$, and the price of running Algorithm 1 on $A^\bullet$ and $(P^\bullet, Rel^\bullet)$.

We assume the automata not to have unreachable states and to have at most one (final) state that is not used in the left-hand side of any transition rule—general automata can be easily pre-processed to satisfy this requirement. Further, we assume the input automaton A to be encoded as a list of states $q \in Q$ and a list of the left-hand sides $l = (q_1, ..., q_n) \in Lhs(A)$. Each left-hand side l is encoded by an array of (pointers to) the states q_1, ..., q_n, plus a list containing a pointer to the so-called f-list for each $f \in \Sigma$ such that there is an f transition from l in Δ. Each f-list is then a list of (pointers to) all the states $q \in Q$ such that $l \xrightarrow{f} q$. We call this representation the *lhs-list* automata encoding. Then, the complexity of preparing the input for computing the downward simulation on A via Algorithm 1 is given by the following lemma.

Lemma 1. *For a TA $A = (Q, \Sigma, \Delta, F)$, the LTS $A^\bullet$ and the partition-relation pair $(P^\bullet, Rel^\bullet)$ can be derived in time and space $O(Rank(A) \cdot |Q| + |\Delta| + (Rank(A) + |\Sigma|) \cdot |Lhs(A)|)$.*

In order to instantiate the complexity of running Algorithm 1 for $A^\bullet$ and $(P^\bullet, Rel^\bullet)$, we first introduce some auxiliary notions. First, we extend $\preccurlyeq_{down}$ to the set $Lhs(A)$

[5] We initially consider all states of the LTS $A^\bullet$ equal, and hence they form a single class of $P^\bullet$, which is related to itself in $Rel^\bullet$.

such that $(q_1,\ldots,q_n) \preccurlyeq_{down} (r_1,\ldots,r_n)$ iff $q_i \preccurlyeq_{down} r_i$ for each $i : 1 \le i \le n$. We notice that $P_{sim} = Q^\bullet/\cong$. From an easy generalisation of Theorem 3 to apply not only for states from Q, but also the left-hand sides of transition rules from $Lhs(A)$, i.e., from the fact that $\forall l_1, l_2 \in Lhs(A). l_1 \preccurlyeq_{down} l_2 \;\Leftrightarrow\; l_1^\bullet \preccurlyeq l_2^\bullet$, we have that $|Q^\bullet/\cong| = |Q/\cong_{down}| + |Lhs(A)/\cong_{down}|$.

Lemma 2. *Given a tree automaton $A = (Q,\Sigma,\Delta,F)$, Algorithm 1 computes the simulation $\preccurlyeq$ on the LTS $A^\bullet$ for the initial partition-relation pair $(P^\bullet, Rel^\bullet)$ with the time complexity $O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/\cong_{down}| + |\Delta| \cdot |Lhs(A)/\cong_{down}|)$ and the space complexity $O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/\cong_{down}|)$.*

The complexity of computing the downward simulation for a tree automaton A via the LTS $A^\bullet$ can now be obtained by simply adding the complexities of computing $A^\bullet$ and $(P^\bullet, Rel^\bullet)$ and of running Algorithm 1 on them.

Theorem 4. *Given a tree automaton A, the downward simulation on A can be computed in time $O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/\cong_{down}| + |\Delta| \cdot |Lhs(A)/\cong_{down}|)$ and space $O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/\cong_{down}| + |\Delta|)$.* [6]

Moreover, under the standard assumption that the maximal rank and size of the alphabet are constants, we get the time complexity $O(|\Delta| \cdot |Lhs(A)/\cong_{down}|)$ and the space complexity $O(|Lhs(A)| \cdot |Lhs(A)/\cong_{down}| + |\Delta|)$.

5 Computing Upward Simulation

In a similar manner to the downward simulation, we translate the upward simulation problem on a tree automaton $A = (Q,\Sigma,\Delta,F)$ to the simulation problem on an LTS $A^\odot$. To define the translation from the upward simulation, we first make the following definition. An *environment* is a tuple of the form $((q_1,\ldots,q_{i-1},\Box,q_{i+1},\ldots,q_n), f, q)$ obtained by removing a state q_i, $1 \le i \le n$, from the i^{th} position of the left hand side of a rule $((q_1,\ldots,q_{i-1},q_i,q_{i+1},\ldots,q_n), f, q)$, and by replacing it by a special symbol $\Box \notin Q$ (called a *hole* below). Like for transition rules, we write $(q_1,\ldots,\Box,\ldots,q_n) \xrightarrow{f} q$ provided $((q_1,\ldots,q_{i-1},q_i,q_{i+1},\ldots,q_n), f, q) \in \Delta$ for some $q_i \in Q$. Sometimes, we also write the environment as $(q_1,\ldots,\Box_i,\ldots,q_n) \xrightarrow{f} q$ to emphasise that the hole is at position i. We denote the set of all environments of A by $Env(A)$.

The derivation of $A^\odot$ differs from $A^\bullet$ in two aspects: (1) we encode environments (rather than left-hand sides of rules) as states in $A^\odot$, and (2) we use a non-trivial initial partition on the states of $A^\odot$, taking into account the downward simulation on Q. Formally, we define $A^\odot = (Q^\odot, \Sigma^\odot, \Delta^\odot)$ as follows:

- The set $Q^\odot$ contains a state $q^\odot$ for each state $q \in Q$, and it also contains a state $((q_1,\ldots,\Box_i,\ldots,q_n) \xrightarrow{f} q)^\odot$ for each environment $(q_1,\ldots,\Box_i,\ldots,q_n) \xrightarrow{f} q$.
- The set $\Sigma^\odot$ contains each symbol $a \in \Sigma$ and also a special symbol $\lambda \notin \Sigma$.

[6] Note that in the special case of $Rank(A) = 1$ (corresponding to a word automaton viewed as a tree automaton), we have $|Lhs(A)| = |Q|$, which leads to the same complexity as Algorithm 1 has when applied directly on word automata.

- For each transition rule $(q_1,\dots,q_n) \xrightarrow{f} q$ of A, the set $\Delta^{\odot}$ contains both the transitions $q_i^{\odot} \xrightarrow{\lambda} ((q_1,\dots,\square_i,\dots,q_n) \xrightarrow{f} q)^{\odot}$ for each $i \in \{1,...,n\}$ and the transition $((q_1,\dots,\square_i,\dots,q_n) \xrightarrow{f} q)^{\odot} \xrightarrow{f} q^{\odot}$.
- The sets $Q^{\odot}$, $\Sigma^{\odot}$, and $\Delta^{\odot}$ do not contain any other elements.

We define I to be the smallest binary relation on $Q^{\odot}$ containing all pairs of states of the automaton A, i.e., all pairs $(q_1^{\odot},q_2^{\odot})$ for each $q_1,q_2 \in Q$, as well as all pairs of environments $(((q_1,\dots,\square_i,\dots,q_n) \xrightarrow{f} q)^{\odot},((r_1,\dots,\square_i,\dots,r_n) \xrightarrow{f} r)^{\odot})$ such that $(q_j,r_j) \in D$ for each $j : 1 \leq j \neq i \leq n$.

The following theorem shows correctness of the translation.

Theorem 5. *For all $q,r \in Q$, we have $q \preccurlyeq_{up} r$ iff $q^{\odot} \preccurlyeq^{I} r^{\odot}$.*

The relation I is clearly a pre-order and so the relation $\iota = I \cap I^{-1}$ is an equivalence. Due to Theorem 5, we can compute the simulation relation $\preccurlyeq_{up}$ on Q by constructing the LTS $A^{\odot}$ and running Algorithm 1 on it with the initial partition-relation pair $(P^{\odot},Rel^{\odot})$ inducing I, i.e., $P^{\odot} = Q^{\odot}/\iota$ and $Rel^{\odot} = \{(B,C) \in P^{\odot} \times P^{\odot} \mid B \times C \subseteq I\}$.

5.1 Complexity of Computing the Upward Simulation

Once the downward simulation $\preccurlyeq_{down}$ on a given TA $A = (Q,\Sigma,\Delta,F)$ is computed, the complexity of computing the simulation $\preccurlyeq_{up}$ naturally consists of the price of compiling A into its corresponding LTS $A^{\odot}$, the price of building the initial partition-relation pair $(P^{\odot},Rel^{\odot})$, and the price of running Algorithm 1 on $A^{\odot}$ and $(P^{\odot},Rel^{\odot})$.

We assume the automaton A to be encoded in the same way as in the case of computing the downward simulation. Compared to preparing the input for computing the downward simulation, the main obstacle in the case of the upward simulation is the need to compute the partition $P_e^{\odot}$ of the set of environments $Env(A)$ wrt. I, which is a subset of the partition $P^{\odot}$ (formally, $P_e^{\odot} = P^{\odot} \cap 2^{Env(A)}$). If the computation of $P_e^{\odot}$ is done naively (i.e., based on comparing each environment with every other environment), it can introduce a factor of $|Env(A)|^2$ into the overall complexity of the procedure. This would dominate the complexity of computing the simulation on $A^{\odot}$ where, as we will see, $|Env(A)|$ is only multiplied by $|Env(A)/\cong_{up}|$.

Fortunately, this complexity blowup can be to a large degree avoided by exploiting the partition $Lhs(A)/\cong_{down}$ computed within deriving the downward simulation as shown in detail in [1]. Here, we give just the basic ideas.

For each $1 \leq i \leq Rank(A)$, we define an i-weakened version D_i of the downward simulation on left-hand sides of A such that $((q_1,\dots,q_n),(r_1,\dots,r_m)) \in D_i \iff n = m \geq i \wedge (\forall 1 \leq j \leq n.\ j \neq i \implies q_j \preccurlyeq_{down} r_j)$. Clearly, each D_i is a pre-order, and we can define the equivalence relations $\approx_i = D_i \cap D_i^{-1}$. Now, a crucial observation is that there exists a simple correspondence between $P_e^{\odot}$ and $Lhs(A)/\approx_i$. Namely, we have that $L \in Lhs(A)/\approx_i$ iff for each $f \in \Sigma$, there is a block $E_{L,f} \in P_e^{\odot}$ such that $E_{L,f} = \{(q_1,\dots,\square_i,\dots,q_n) \xrightarrow{f} q \mid \exists q_i,q \in Q.\ (q_1,...,q_i,...,q_n) \in L \wedge (q_1,...,q_i,...,q_n) \xrightarrow{f} q\}$.

The idea of computing $P_e^{\odot}$ is now to first compute blocks of $Lhs(A)/\approx_i$ and then to derive from them the $P_e^{\odot}$ blocks. The key advantage here is that the computation of the $\approx_i$-blocks can be done on blocks of $Lhs(A)/\cong_{down}$ instead of directly on elements of

$Lhs(A)$. This is because, for each i, blocks of $Lhs(A)/\cong_{down}$ are sub-blocks of blocks of $Lhs(A)/\approx_i$. Moreover, for any blocks K, L of $Lhs(A)/\cong_{down}$, the test of $K \times L \subseteq D_i$ can simply be done by testing whether $(k,l) \in D_i$ for any two representatives $k \in K, l \in L$. Therefore, all $\approx_i$-blocks can be computed in time proportional to $|Lhs(A)/\cong_{down}|^2$.

From each block $L \in Lhs(A)/\approx_i$, one block $E_{L,f}$ of $P_e^{\odot}$ is generated for each symbol $f \in \Sigma$. The $E_{L,f}$ blocks are obtained in such a way that for each left-hand side $l \in L$, we generate all the environments which arise by replacing the i^{th} state of l by $\Box$, adding f, and adding a right-hand side state $q \in Q$ which together with l form a transition $l \xrightarrow{f} q$ of A. This can be done efficiently using the lhs-list encoding of A. An additional factor $|\Delta| \cdot \log |Env(A)|$ is, however, introduced due to a need of not having duplicates among the computed environments, which could result from transitions that differ just in the states that are replaced by $\Box$ when constructing an environment. The factor $\log |Env(A)|$ comes from testing a set membership over the computed environments to check whether we have already computed them before or not.

Moreover, it can be shown that $Rel^{\odot}$ can be computed in time $|P^{\odot}|^2$. The complexity of constructing $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$ is then summarised in the below lemma.

Lemma 3. *Given a tree automaton $A = (Q, \Sigma, \Delta, F)$, the downward simulation $\preccurlyeq_{down}$ on A, and the partition $Lhs(A)/\cong_{down}$, the LTS $A^{\odot}$ and the partition-relation pair $(P^{\odot}, Rel^{\odot})$ can be derived in time $O(|\Sigma| \cdot |Q| + Rank(A) \cdot (|Lhs(A)| + |Lhs(A)/\cong_{down}|^2) + Rank(A)^2 \cdot |\Delta| \cdot \log |Env(A)| + |P^{\odot}|^2)$ and in space $O(|\Sigma| \cdot |Q| + |Env(A)| + Rank(A) \cdot |Lhs(A)| + |Lhs(A)/\cong_{down}|^2 + |P^{\odot}|^2)$.*

In order to instantiate the complexity of running Algorithm 1 for $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$, we again first introduce some auxiliary notions. Namely, we extend $\preccurlyeq_{up}$ to the set $Env(A)$ such that $(q_1, \dots, \Box_i, \dots, q_n) \xrightarrow{f} q \preccurlyeq_{up} (r_1, \dots, \Box_j, \dots, r_m) \xrightarrow{f} r \iff m = n \wedge i = j \wedge q \preccurlyeq_{up} r \wedge (\forall k \in \{1, ..., n\}.\ k \neq i \implies q_k \preccurlyeq_{down} r_k)$. We notice that $P_{sim} = Q^{\odot}/\cong^I$. From an easy generalisation of Theorem 5 to apply not only for states from Q, but also environments from $Env(A)$, i.e., from the fact that $\forall e_1, e_2 \in Env(A).\ e_1 \preccurlyeq_{up} e_2 \iff e_1^{\odot} \preccurlyeq^I e_2^{\odot}$, we have that $|Q^{\odot}/\cong^I| = |Q/\cong_{up}| + |Lhs(A)/\cong_{up}|$.

Lemma 4. *Given a tree automaton $A = (Q, \Sigma, \Delta, F)$, the upward simulation $\preccurlyeq_{up}$ on A can be computed by running Algorithm 1 on the LTS $A^{\odot}$ and the partition-relation pair $(P^{\odot}, Rel^{\odot})$ in time $O(Rank(A) \cdot |\Delta| \cdot |Env(A)/\cong_{up}| + |\Sigma| \cdot |Env(A)| \cdot |Env(A)/\cong_{up}|)$ and space $O(|\Sigma| \cdot |Env(A)| \cdot |Env(A)/\cong_{up}|)$.*

The complexity of computing upward simulation on a TA A can now be obtained by simply adding the price of computing downward simulation, the price of computing $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$, and the price of running Algorithm 1 on $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$.

Theorem 6. *Given a tree automaton $A = (Q, \Sigma, \Delta, F)$, let $T_{down}(A)$ and $S_{down}(A)$ denote the time and space complexity of computing the downward simulation $\preccurlyeq_{down}$ on A. Then, the upward simulation $\preccurlyeq_{up}$ on A can be computed in time*
$$O((|\Sigma| \cdot |Env(A)| + Rank(A) \cdot |\Delta|) \cdot |Env(A)/\cong_{up}| + Rank(A)^2 \cdot |\Delta| \cdot \log |Env(A)| + T_{down}(A))$$
and in space $O(|\Sigma| \cdot |Env(A)| \cdot |Env(A)/\cong_{up}| + S_{down}(A))$.[7]

[7] Note that in the special case of $Rank(A) = 1$ (corresponding to a word automaton viewed as a tree automaton), we have $|Env(A)| \leq |\Sigma| \cdot |Q|$, which leads to almost the same complexity (up to the logarithmic component) as Algorithm 1 has when applied directly on word automata.

Finally, from the standard assumption that the maximal rank and the alphabet size are constants and from observing that $|Env(A)| \leq Rank(A) \cdot |\Delta| \leq Rank(A) \cdot |\Sigma| \cdot |Q|^{Rank(A)+1}$, we get the time complexity $O(|\Delta| \cdot (|Env(A)/\cong_{up}| + \log|Q|) + T_{down}(A))$ and the space complexity $O(|Env(A)| \cdot |Env(A)/\cong_{up}| + S_{down}(A))$.

6 Reducing Tree Automata

In this section, we describe how to reduce tree automata while preserving the language of the automaton. The idea is to identify suitable equivalence relations on states of tree automata, and then collapse the sets of states which form equivalence classes. We will consider two reduction methods: one which uses downward simulation, and one which is defined in terms of both downward and upward simulation. The choice of the equivalence relation is a trade-off between the amount of reduction achieved and the cost of computing the relation. The second mentioned equivalence is heavier to compute as it requires that both downward and upward simulation are computed and then suitably composed. However, it is at least as coarse as—and often significantly coarser than—the downward simulation equivalence, and hence can give much better reductions as witnessed even in our experiments.

Consider a tree automaton $A = (Q, \Sigma, \Delta, F)$ and an equivalence relation $\equiv$ on Q. The *abstract tree automaton* derived from A and $\equiv$ is $A\langle\equiv\rangle = (Q\langle\equiv\rangle, \Sigma, \Delta\langle\equiv\rangle, F\langle\equiv\rangle)$ where:

– $Q\langle\equiv\rangle$ is the set of blocks in $\equiv$. In other words, we collapse all states which belong to the same block into one abstract state.
– $(B_1, \ldots, B_n) \xrightarrow{f} B$ iff $(q_1, \ldots, q_n) \xrightarrow{f} q$ for some $q_1 \in B_1, \ldots, q_n \in B_n, q \in B$. This is, there is a transition in the abstract automaton iff there is a transition between states in the corresponding blocks.
– $F\langle\equiv\rangle$ contains a block B iff $B \cap F \neq \emptyset$. Intuitively, a block is accepting if it contains at least one state which is accepting.

6.1 Downward Simulation Equivalence

Given a tree automaton $A = (Q, \Sigma, \Delta, F)$, we consider the abstract automaton $A\langle\cong_{down}\rangle$ constructed by collapsing states of A which are equivalent with respect to $\cong_{down}$. We show that the two automata accept the same language, i.e., $L(A) = L(A\langle\cong_{down}\rangle)$. Observe that the inclusion $L(A) \subseteq L(A\langle\cong_{down}\rangle)$ is straightforward. We can prove the inclusion in the other direction as follows. Using a simple induction on trees, one can show that downward simulation implies language inclusion. In other words, for states $q, r \in Q$, if $q \preceq_{down} r$, then $L(q) \subseteq L(r)$. This implies that for any $B \in Q\langle\cong_{down}\rangle$, it is the case that $L(B) \subseteq L(r)$ for any $r \in B$. Now suppose that $t \in L(A\langle\cong_{down}\rangle)$. It follows that $t \in L(B)$ for some $B \in F\langle\cong_{down}\rangle$. Since $B \in F\langle\cong_{down}\rangle$, there is some $r \in B$ with $r \in F$. It follows that $t \in L(r)$, and hence $t \in L(A)$. This gives the following Theorem.

Theorem 7. *$L(A) = L(A\langle\cong_{down}\rangle)$ for each tree automaton A.*

In fact, $A\langle\cong_{down}\rangle$ is the minimal automaton which is equivalent to A with respect to downward simulation and which accepts the same language as A.

6.2 Composed Equivalence

Consider a tree automaton $A = (Q,\Sigma,\Delta,F)$. Let I_F be a partitioning of Q such that $(q,r) \in I_F$ iff $q \in F \implies r \in F$. Consider a reflexive and transitive downward simulation D, and a reflexive and transitive upward simulation U induced by D. Assume that $U \subseteq I_F$. We will reduce A with respect to relations of the form $\equiv_R$ which preserve language equivalence, but which may be much coarser than downward simulations. Here, each $\equiv_R$ is an equivalence relation $R \cap R^{-1}$ defined by a pre-order R satisfying certain properties. More precisely, we use $D \oplus U$ to denote the set of relations on Q such that for each $R \in (D \oplus U)$, the relation R satisfies the following two properties: (i) R is transitive and (ii) $D \subseteq R \subseteq \left(D \circ U^{-1}\right)$. For a state $r \in Q$ and a set $B \subseteq Q$ of states, we write $(B,r) \in D$ to denote that there is a $q \in B$ with $(q,r) \in D$. We define $(B,r) \in U$ analogously. We will now consider the abstract automaton $A\langle\equiv_R\rangle$ where the states of A are collapsed according to $\equiv_R$. We will relate the languages of A and $A\langle\equiv_R\rangle$.

To do that, we first define the notion of a *context*. Intuitively, a context is a tree with "holes" instead of leaves. Formally, we consider a special symbol $\bigcirc \notin \Sigma$ with rank 0. A *context* over Σ is a tree c over $\Sigma \cup \{\bigcirc\}$ such that for all leaves $p \in c$, we have $c(p) = \bigcirc$. For a context c with leaves $p_1,\ldots,p_n$, and trees $t_1,\ldots,t_n$, we define $c[t_1,\ldots,t_n]$ to be the tree t, where

- $dom(t) = dom(c) \bigcup \{p_1 \cdot p' | \, p' \in dom(t_i)\} \bigcup \cdots \bigcup \{p_n \cdot p' | \, p' \in dom(t_n)\}$,
- for each $p = p_i \cdot p'$, we have $t(p) = t_i(p')$, and
- for each $p \in dom(c) \setminus \{p_1,\ldots,p_n\}$, we have $t(p) = c(p)$.

In other words, $c[t_1,\ldots,t_n]$ is the result of appending the trees $t_1,\ldots,t_k$ to the holes of c. We extend the notion of runs to contexts. Let c be a context with leaves $p_1,\ldots,p_n$. A *run* π of A on c from $(q_1,\ldots,q_n)$ is defined in a similar manner to a run on a tree except that for a leaf p_i, we have $\pi(p_i) = q_i$, $1 \le i \le n$. In other words, each leaf labelled with $\bigcirc$ is annotated by one q_i. We use $c\,[q_1,\ldots,q_n] \overset{\pi}{\Longrightarrow} q$ to denote that π is a run of A on c from $(q_1,\ldots,q_n)$ such that $\pi(\varepsilon) = q$. The notation $c\,[q_1,\ldots,q_n] \Longrightarrow q$ is explained in a similar manner to runs on trees.

Using the notion of a context, we can relate runs of A with those of the abstract automaton $A\langle\equiv_R\rangle$. More precisely, we can show that for blocks $B_1,\ldots,B_n,B \in Q\langle\equiv_R\rangle$ and a context c, if $c[B_1,\ldots,B_n] \Longrightarrow B$, then there exist states $r_1,\ldots,r_n,r \in Q$ such that $(B_1,r_1) \in D,\ldots,(B_n,r_n) \in D, (B,r) \in U$, and $c[r_1,\ldots,r_n] \Longrightarrow r$. In other words, each run in $A\langle\equiv_R\rangle$ can be simulated by a run in A which starts from larger states (with respect to downward simulation) and which ends up at a larger state (with respect to upward simulation). This leads to the following lemma.

Lemma 5. *If $t \Longrightarrow B$, then $t \Longrightarrow w$ for some w with $(B,w) \in U$. Moreover, if $B \in F\langle\equiv_R\rangle$, then also $w \in F$.*

In other words, each tree t which leads to a block B in $A\langle\equiv_R\rangle$ will also lead to a state in A which is larger than (some state in) the block B with respect to upward simulation. Moreover, if t can be accepted at B in $A\langle\equiv_R\rangle$ (meaning that B contains a final state of A, i.e., $B \cap F \neq \emptyset$), then it can be accepted at w in A (i.e., $w \in F$) too.

Notice that Lemma 5 holds for any downward and upward simulations satisfying the properties mentioned in the definition of $\oplus$. We now instantiate the lemma for the maximal downward and upward simulation to obtain the main result. We take D and U to be $\preccurlyeq_{down}$ and $\preccurlyeq_{up}^{I_F}$, respectively, and we let $\preccurlyeq_{comp}$ be any relation from the set of relations $(\preccurlyeq_{down} \oplus \preccurlyeq_{up}^{I_F})$. We let $\cong_{comp}$ be the corresponding equivalence.

Theorem 8. *$L(A\langle\cong_{comp}\rangle) = L(A)$ for each tree automaton A.*

Proof. The inclusion $L(A\langle\cong_{comp}\rangle) \supseteq L(A)$ is trivial. Let $t \in L(A\langle\cong_{comp}\rangle)$, i.e., $t \Longrightarrow B$ for some block B where $B \cap F \neq \emptyset$. Lemma 5 implies that $t \Longrightarrow w$ such that $w \in F$. $\square$

Note that it is clearly the case that $\cong_{down} \subseteq \cong_{comp}$. Moreover, note that a relation $\preccurlyeq_{comp} \in (\preccurlyeq_{down} \oplus \preccurlyeq_{up}^{I_F})$ can be obtained, e.g., by a simple (random) pruning of the relation $\preccurlyeq_{down} \circ (\preccurlyeq_{up}^{I_F})^{-1}$ based on iteratively removing links not being in $\preccurlyeq_{down}$ and at the same time breaking transitivity of the so-far computed composed relation. Such a way of computing $\preccurlyeq_{comp}$ does not guarantee that one obtains a relation of the greatest cardinality possible among relations from $\preccurlyeq_{down} \oplus \preccurlyeq_{up}^{I_F}$, but, on the other hand, it is cheap (in the worst case, cubic in the number of states). Moreover, our experiments show that even this simple way of computing the composed relation can give us a relation $\cong_{comp}$ that is much coarser (and yields significantly better reductions) than $\cong_{down}$.

Remark Our definition of a context coincides with the one of [8] where all leaves are holes. On the other hand, a context in [9] and [3] is a tree with a *single* hole. Considering single-hole contexts, one can define the *language of contexts* $L_c(q)$ of a state q to be the set of contexts on which there is an accepting run if the hole is replaced by q. Then, for all states q and r, it is the case that $q \preccurlyeq_{up} r$ implies $L_c(q) \subseteq L_c(r)$.

7 Experiments with Reducing Tree Automata

We have implemented our algorithms in a prototype tool written in Java. We have run the prototype on a number of tree automata that arise in the framework of *tree regular model checking*. Tree regular model checking is the name of a family of techniques for analysing infinite-state systems in which states are represented by trees, (infinite) sets of states by finite tree automata, and transitions by tree transducers. Most of the algorithms in the framework rely crucially on efficient automata reduction methods since the size of the generated automata often explodes, making computations infeasible without reduction. The (nondeterministic) tree automata that we have considered arose during verification of the *Percolate* protocol, the *Arbiter* protocol, and the *Leader* election protocol [4].

Our experimental evaluation was carried out on an AMD Athlon 64 X2 2.19GHz PC with 2.0 GB RAM. The time for minimising the tree automata varied from a few seconds up to few minutes. Table 1 shows the number of states and rules of the various considered tree automata before and after computing $\cong_{down}$, $\cong_{comp}$, and the backward bisimulation from [11]. Backward bisimulation is the bisimulation counterpart of downward simulation. The composed simulation equivalence $\cong_{comp}$ was computed in the simple way based on the random pruning of the relation $\preccurlyeq_{down} \circ (\preccurlyeq_{up}^{I_F})^{-1}$ as mentioned at the end of Section 6.2. As Table 1 shows, $\cong_{comp}$ achieves the best reduction (often reducing to less than one-third of the size of the original automaton). As expected, both $\cong_{down}$ and $\cong_{comp}$ give better reductions than backward bisimulation in all test cases.

Protocol	**original**		$\cong_{down}$		$\cong_{comp}$		**backward bisimulation**	
	states	rules	states	rules	states	rules	states	rules
percolate	10	72	7	45	7	45	10	72
	20	578	17	392	14	346	20	578
	28	862	13	272	13	272	15	341
arbiter	15	324	10	248	7	188	11	252
	41	313	28	273	19	220	33	285
	109	1248	67	1048	55	950	83	1116
leader	17	153	11	115	6	47	16	152
	25	384	16	235	6	59	23	382
	33	876	10	100	7	67	27	754

Table 1. Reduction of the number of states and rules using different reduction algorithms.

8 Conclusions and Future Work

We have presented methods for reducing tree automata under language equivalence. For this purpose, we have considered two kinds of simulation relations on the states of tree automata, namely downward and upward simulation. We give procedures for efficient translation of both kinds of relations into simulations defined on labelled transition systems. Furthermore, we define a new, language-preserving equivalence on tree automata, derived from compositions of downward and upward simulation, which (according to our experiments) usually gives a much better reduction on the size of automata than downward simulation.

There are several interesting directions for future work. First, we would like to implement the proposed algorithms in a more efficient way, perhaps over automata encoded in a symbolic way using BDDs like in MONA [12], in order to be able to experiment with bigger automata. Further, for instance, we can define *upward* and *downward bisimulation* for tree automata in an analogous way to the case of simulation. It is straightforward to show that the encoding we use in this paper can also be used to translate bisimulation problems on tree automata into corresponding ones for LTSs. Although reducing according to a bisimulation does not give the same reduction as for a simulation, it is relevant since it generates more efficient algorithms. Also, we plan to investigate coarser relations for better reductions of tree automata by refining the ideas behind the definition of the composed relation introduced in Section 6. We believe that it is possible to define a refinement scheme allowing one to define an increasing family of such relations between downward simulation equivalence and tree language equivalence. Finally, we plan to consider extending our reduction techniques to the class of unranked trees which are used in applications such as reasoning about structured documents or about configurations of dynamic concurrent processes.

Acknowledgement. The work was supported by the ANR-06-SETI-001 French project AVERISS, the Czech Grant Agency (projects 102/07/0322 and 102/05/H050), the Czech-French Barrande project 2-06-27, and the Czech Ministry of Education by the project MSM 0021630528 *Security-Oriented Research in Information Technology*.

References

1. P. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Computing Simulations over Tree Automata. Technical report, FIT-TR-2007-001, FIT, Brno University of Technology, Czech Republic, 2007.
2. P. Abdulla, J. Högberg, and L. Kaati. Bisimulation Minimization of Tree Automata. In *Proc. of CIAA'06*, *LNCS* 4094. Springer, 2006.
3. P. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular Tree Model Checking. In *Proc. of CAV'02*, *LNCS* 2404. Springer, 2002.
4. P. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree Regular Model Checking: A Simulation-based Approach. *The Journal of Logic and Algebraic Programming*, 69(1-2):93–121, 2006.
5. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular Tree Model Checking. In *ENTCS* 149(1):37–48. Elsevier, 2006.
6. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *SAS'06*, LNCS 4134. Springer, 2006.
7. A. Bouajjani and T. Touili. Extrapolating Tree Transformations. In *Proc. of CAV'02, LNCS* 2404. Springer, 2002.
8. A. Bouajjani and T. Touili. Reachability Analysis of Process Rewrite Systems. In *Proc. of FSTTCS'03*, *LNCS* 2914. Springer, 2003.
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
10. M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of FOCS'95*. IEEE, 1995.
11. J. Högberg, A. Maletti, and J. May. Backward and forward bisimulation minimisation of tree automata. In *Pre-proceedings of CIAA'07*. Czech Technical University in Prague, Czech Republic, 2007.
12. N. Klarlund and A. Møller. MONA Version 1.4 User Manual, 2001. BRICS, Department of Computer Science, University of Aarhus, Denmark.
13. R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16:973–989, 1987.
14. F. Ranzato and F. Tapparo. A New Efficient Simulation Equivalence Algorithm. In *Proc. of LICS'07*. IEEE CS, 2007.

A Proofs of the Theorems Presented in the Paper

A.1 Correctness of Computing Simulations on LTS (Algorithm 1)

Let us first introduce some notation. By an *iteration*, we will mean a single iteration of the while loop of the algorithm. For an iteration, the block B chosen on line 3 (also referd to as B_{prev}) will be denoted as the *pivot* of the iteration. An *ancestor* of a block C is any block D which appears during the computation and for which $C \subseteq D$, and on the contrary, C is a *descendent* of D. Moreover, if D is the immediate ancestor of C such that C was created while splitting D, then D is the *parent* of C and C is a *child* of D.

Given an LTS $T = (S, \mathcal{L}, \rightarrow)$ and $q, r \in S$, we will denote by $q \overset{a}{\not\longrightarrow} r$ the fact that $\neg(q \xrightarrow{a} r)$. Moreover, for any $B, C \subseteq S$, $q \overset{a}{\not\longrightarrow} C$ and $B \xrightarrow{a} C$ are defined analogously, i.e. provided that $q \notin pre_a(C)$ and $B \cap pre_a(C) = \emptyset$.

Lemma 6. *On line 3 of Algorithm 1, the pair (P, Rel) is always a partition-relation pair. The partition P can only be refined during the computation. Moreover, the relation induced by the partition-relation pair (P, Rel) can only shrink during the computation.*

Proof. The fact that (P, Rel) is a partition-relation pair can only be temporarily broken by the *Split* operation on line 6 but after inheriting all *Rel* links of parent classes to children classes on lines 7–10, it again holds. The other two claims of the lemma are also immediate as the algorithm can only split the classes of P (but never unites them), and it can only remove some elements from *Rel*. □

Lemma 7. *The following claims are invariants of the while loop of Algorithm 1:*

$$\forall B \in P.\ \forall a \in \mathcal{L}.\ Remove_a(B) \overset{a}{\not\longrightarrow} \bigcup Rel(B) \tag{1}$$

$$\forall B \in P.\ B \in Rel(B) \tag{2}$$

$$\forall B, C \in P.\ (B, C) \in Rel \implies \Big(\forall a \in \mathcal{L}.\ \forall D \in P.\ B \xrightarrow{a} D \implies C \subseteq pre_a(\bigcup Rel(D)) \cup Remove_a(D)\Big) \tag{3}$$

Proof. After the initialization, all the invariants hold. It is not so difficult to see it as I and therefore also *Rel* are transitive and reflexive.

- Invariant (1) can never be broken. After the initialization it holds. From there on, it holds because only such a state r can be moved into the $Remove_b(C)$ which is not in $pre_b(\bigcup Rel(C))$ (the test on line 16). Moreover, if r is once not in $pre_b(\bigcup Rel(C))$, then it will no more be there (Lemma 6).
- Invariant (2) can never be broken as breaking reflexivity of *Rel* requires choosing (C, D) on line 14 such that $C = D$. For any such a pair on line 14, it holds that $C \xrightarrow{a} B$ and $D \subseteq Remove_a(B)$ where B is the pivot block. But, thanks to Invariant (1), this is not possible for $C = D$.
- Invariant (3) can be temporarily broken on three places of the algorithm:
  lines 6–10: Let C be a block of P on line 7 and let $C' \in P_{\mathsf{prev}}$ be its parent. Then it is easy to see that after finishing the for loop on line 7, it holds that $\bigcup Rel(C) = \bigcup Rel_{\mathsf{prev}}(C')$ and for all $a \in \mathcal{L}$, $Remove_a(C) = Remove_a(C')$. Thus after finishing the for loop on line 7, invariant (3) can be broken only for those (B, C) pairs such that it was broken even for their parents on line 6.

line 4: Assume that line 4 breaks the invariant and let B by the pivot of the iteration of the while loop. Then there are $C,D \in P$ which break the invariant such that $(C,D) \in Rel$, $C \xrightarrow{a} B$, $D \subseteq pre_a(\bigcup Rel(B)) \cup Remove_a(B)$, and $D \nsubseteq pre_a(\bigcup Rel(B))$. The *Split* operation on line 6 divides D into $D_1 \subseteq pre_a(\bigcup Rel(B))$ and $D_2 \subseteq Remove$. Then *Rel* and the *Remove* sets are inherited on lines 7–10. Now only (C', D_2) pairs break the invariant where C' is a child of C which leads under a into a child of B. But exactly these pairs will be finally chosen on line 13 and the relation *Rel* will be cut on exactly these places.

line 16: The invariant can be broken on this line as there can be some states r such that $r \xrightarrow{b} D$ and thus before the update of *Rel*, $r \xrightarrow{b} \bigcup Rel(C)$, but after the removal of D from $Rel(C)$, it can happen that $r \stackrel{b}{\nrightarrow} \bigcup Rel(C)$. But exactly these r states are moved into $Remove_b(C)$, and so Invariant (3) holds after finishing the for loop on line 13 again.

□

Lemma 8. *If all the Remove sets are empty, then the relation* δ *induced by the partition relation pair* (P, Rel) *is a simulation on* T *included in* I.

Proof. The initial partition-relation pair is required to be such that δ is initially included in I. We have to show that δ is also a simulation on T. Let $q \in B \in P$, $r \in C \in P$, and $q\ \delta\ r$. Then, from the definition of δ, $(B,C) \in Rel$. Let $q \xrightarrow{a} s \in D$, thus $B \xrightarrow{a} D$. Therefore, from Invariant (3) and from the fact that all the *Remove* sets are empty, we get $C \subseteq pre_a(\bigcup Rel(D))$. This means that for all $t \in C$ there is $u \in E \in Rel(D)$ such that $t \xrightarrow{a} u$ and $s\ \delta\ u$ because $u \in E \in Rel(D)$ and $s \in D$. As $r \in C$, the lemma holds. □

Lemma 9. *Let* δ *be the relation induced by the partition-relation pair* (P, Rel)*, let* $\preccurlyeq^I$ *be the maximal simulation on* T *included in* I*, and let* $\preccurlyeq^I \subseteq \delta$*. Then, if we are on line 3 of Algorithm 1 and there are states* $q, r \in S$ *and blocks* $B, C, D \in P$ *such that* $q \preccurlyeq^I r$*,* $q \in C, r \in D$ *and* $(B,C) \in Rel$*, then also* $(B,D) \in Rel$*.*

Proof. Let us recall the relationship between a partition-relation pair (P, Rel) and its induced relation δ which is: For any $B, C \in P$ and $q \in B, r \in C$, it holds that $q\ \delta\ r$ iff $(B,C) \in Rel$. Therefore, if $\delta \subseteq \preccurlyeq^I$, then $q \preccurlyeq^I r$ implies $(B,C) \in Rel$.

We prove the lemma by induction on the number of iterations of the while loop. The base case: After the initialization, the claim holds as Rel_{init} is transitive (the relation I is a pre-order). We prove the induction step by contradiction. Suppose the lemma might get broken during the execution of the algorithm. Then, we can identify the first moment when it is broken.

Let M_i be the moment (computation step) when we are on line 3 at the begining of the i-th iteration and the lemma is broken for the first time. At that monent, we have $B, C, D \in P$, $q \in C$, $r \in D$, $q \preccurlyeq^I r$, $(B,C) \in Rel$, $\preccurlyeq^I \subseteq \delta$, and $(B,D) \notin Rel$. From $q \preccurlyeq^I r$ and $\preccurlyeq^I \subseteq \delta$, we have $(C,D) \in Rel$. Because the induced relation is shrinking only (Lemma 6), we have that at each moment of the computation that precedes M_i, the relation $\preccurlyeq^I$ was a subset of the induced relation, the ancestor C' of C was over the ancestor B' of B (i.e. $(B', C') \in Rel$), and also the ancestor of D was over the ancestor of C wrt. the current *Rel*. Because of this and because I (Rel_{init}) is transitive and the lemma is broken for the first time at M_i, we know that at each moment preceding M_i, the ancestor of D was over the ancestor of B.

Let us choose the moment before M_i when (B,D) is going to be removed from Rel' (the moment in the iteration $i-1$ preceding M_i, just before entering the for loop on line 11). The current partition P at that moment is the same as at M_i. The situation is such that $(B,C) \in Rel'$, $(C,D) \in Rel'$, $(B,D) \in Rel'$, and we are going to remove (B,D) from Rel' on line 14. However, we will not touch (B,C) and (C,D) during this iteration as these two pairs will be related at the moment M_i. This update ($Rel' \leftarrow Rel' \setminus \{(B,D)\}$) is caused by processing the $Remove_a(E)$ set, where $E \in P_{\mathsf{prev}}$ is the pivot of the iteration such that $B \xrightarrow{a} E, D \subseteq Remove_a(E)$ and $C \cap Remove_a(E) = \emptyset$ (we have split according to $Remove_a(E)$).

At the beginning of the $(i-1)$-th iteration, it still holds for the induced relation δ' that $\preccurlyeq^I \subseteq \delta'$ (this moment precedes M_i). Let $B', C', D' \in P_{\mathsf{prev}}$ be the ancestors of B, C, D (therefore $B \subseteq B', C \subseteq C', D \subseteq D'$). We have that $q \in C \subseteq C'$, $C \cap Remove_a(E) = \emptyset$, $B' \xrightarrow{a} E$, and $(B', D') \in Rel'_{\mathsf{prev}}$, and therefore, from Invariant (3), we have that $C' \subseteq pre_a(\bigcup Rel(E)) \cup Remove_a(E)$. This implies that $C \subseteq pre_a(\bigcup Rel(E))$. Thus, $q \xrightarrow{a} q' \in F \in Rel'_{\mathsf{prev}}(E)$.

Therefore, as $q \preccurlyeq^I r$, we have $r \xrightarrow{a} r'$ where $q' \preccurlyeq^I r'$ and because $\preccurlyeq^I \subseteq \delta'$, $r' \in G \in Rel'_{\mathsf{prev}}(F)$. Finally, because $r \in D \subseteq Remove_b(E)$, from Invariant (1), we get $(E,G) \notin Rel'_{\mathsf{prev}}$. However, the states q', r', the blocks $E, F, G \in P_{\mathsf{prev}}$, and the partition-relation pair $(P_{\mathsf{prev}}, Rel'_{\mathsf{prev}})$ (which is the current partition-relation pair on line 3 in the iteration $i-1$ preceding M_i) now form a situation breaking the lemma, which is the same as the situation at the moment M_i. This is not possible as M_i was supposed to be the first such moment. □

Lemma 10. *Let δ be the relation induced by the partition-relation pair (P, Rel) and let $\preccurlyeq^I$ be the maximal simulation on T included in I. Then, $\preccurlyeq^I \subseteq \delta$.*

Proof. By contradiction. We will show that breaking this lemma in a run of Algorithm 1 has to be preceded by breaking Lemma 9.

Let $q \in B \in P$, $r \in C \in P$ such that $q \preccurlyeq^I r$. Let us choose the moment when (B,C) is removed from Rel on line 14. This update of Rel is caused by processing the set $Remove_a(D)$ where $D \in P_{\mathsf{prev}}$ is the pivot of the concerned iteration of the while loop, $B \xrightarrow{a} D$, and $C \subseteq Remove_a(D)$. Let $B', C' \in P_{\mathsf{prev}}$ be the ancestors of B, C. From Invariant (2), we have that $(B', B') \in Rel_{\mathsf{prev}}$, and then $B' \xrightarrow{a} D$ together with Invariant (3) gives $B' \subseteq pre_a(\bigcup Rel_{\mathsf{prev}}(D)) \cup Remove$. Thus, as $q \notin Remove$, $q \xrightarrow{a} q' \in E \in Rel_{\mathsf{prev}}(D)$. From $q \preccurlyeq^I r$ and from the fact that $\preccurlyeq^I$ is a subset of the current induced relation (the lemma is going to be broken for the first time and it still holds), we have $r \xrightarrow{a} r' \in F \in Rel_{\mathsf{prev}}(E)$. However, as $r \in Remove_a(D)$ and because of Invariant (1), we have $(D,F) \notin Rel_{\mathsf{prev}}$. Therefore, the states q', r' and the blocks D, E, F break Lemma 9 (at the beginning of the given iteration). □

Lemma 11. *Let $\preccurlyeq^I$ be the maximal simulation on T included in I. Then, at any point in a run of Algorithm 1, any $q, r \in S$ such that $q \cong^I r$ are in the same block of P.*

Proof. By contradiction. We will show that breaking this lemma in a run of Algorithm 1 has to be preceded by breaking Lemma 9.

After the initialization the lemma holds. Let us choose the first moment when it is broken. At that moment, the states q, r are separated from each other by the *Split* operation during processing of some pivot block B where, without loss of generality, at the beginning of the concerned iteration of the while loop, $r \in Remove_a(B)$ and $q \notin Remove_a(B)$.

Let us now consider the moment just before entering the for loop on line 11 during which r will be added into $Remove_a(B')$ where B' is an ancestor of B. Let the partition-relation pair that the algorithm is working with at that moment be (P', Rel') inducing a relation δ, and let $q, r \in C \in P'$. There is an edge $r \xrightarrow{a} D \in Rel'(B')$ such that (B', D) will be removed from Rel'.

From $r \preccurlyeq^I q$ and $\preccurlyeq^I \subseteq \delta$ (Lemma 10) and because $r \xrightarrow{a} r' \in D$, there is an edge $q \xrightarrow{a} q' \in E \in Rel(D)$, $q' \preccurlyeq^I r'$. Moreover, from Lemma 9 (whose claim holds also for line 11 just before entering the for loop because lines 4–10 do not influence the induced relation), $(B, E) \in Rel(B')$.

Thus there are edges such as $q \xrightarrow{a} q' \in E \in Rel'(B')$ before entering the for loop on line 11. Moreover, at least one such edge $q \xrightarrow{a} q'' \in E' \in Rel(B')$ will remain also after finishing the for loop because if all the (B', X) relations such that $q \xrightarrow{a} X$ disappeared from Rel', then q would move to $Remove_a(B')$, which will not happen. Because $q \preccurlyeq^I r$ and $\preccurlyeq^I \subseteq \delta$, we have $r \xrightarrow{a} r'' \in F \in Rel'(E'), q'' \preccurlyeq^I r''$. But at the end of the for loop on line 11, $(B', F) \notin Rel'$ as r will be added into $Remove_a(B')$ (Invariant (1)). Therefore states q'', r'' and blocks B', E', F break Lemma 9 at the beginning of the following iteration of the while loop. $\square$

Lemma 12. *Let B, B' be two blocks appearing during a run of Algorithm 1 such that B' is an ancestor of B. Let $Remove_a(B)$ and $Remove_a(B')$ be two Remove sets at the (different) moments when B, resp. B', is chosen as the pivot. Then, $Remove_a(B) \cap Remove_a(B') = \emptyset$.*

Proof. If q is in $Remove_a(B)$ after the initialization, then $q \overset{a}{\nrightarrow} \bigcup Rel_{init}(B)$. If q is added into $Remove_a(B)$ later on, then $q \xrightarrow{a} \bigcup Rel(B)$ on line 13 in the while loop iteration when q is added into $Remove_a(B)$.[8] Moreover, subsequently, after the update of Rel on line 14, $q \overset{a}{\nrightarrow} \bigcup Rel(B)$. From Lemma 6, if once $q \overset{a}{\nrightarrow} \bigcup Rel(B)$, then it will never happen that $q \xrightarrow{a} \bigcup Rel(B')$ where B' is a descendent of B. Thus, $q \xrightarrow{a} (\bigcup Rel(B))$ is a neccesary condition which has to hold on line 13 for q to be added into $Remove_a(B)$ on line 17. However, if q is really added into $Remove_a(B)$, then the condition $q \xrightarrow{a} \bigcup Rel(B)$ is broken on line 14 and will never hold for any descendent of B again. Therefore, if q is once in $Remove_a(B)$, then the neccesary condition $q \xrightarrow{a} \bigcup Rel(B)$ will never hold and thus it can never happen that q is being added into any $Remove_a(B')$ where B' is a descendent of B. Then, it cannot happen that B is chosen as a pivot, $Remove_a(B)$ is emptied, and then some of its descendents B' is chosen as a pivot with a $Remove_a(B')$ set such that $Remove_a(B) \cap Remove_a(B') \neq \emptyset$ set. $\square$

[8] Note that at that time, B is referred to via C in the algorithm.

Proof of Theorem 1

Proof. Due to Lemma 12, for any block B which can arise during the computation, B can be chosen as a pivot only finitely many times as for any $a \in \Sigma$, all the $Remove_a(B)$ sets encountered on line 3 are disjoint. There are finitely many possible blocks and hence the algorithm terminates.

Lemma 8 implies that the relation δ induced by the final partition-relation pair (P_{sim}, Rel_{sim}) is a simulation included in I. Lemma 10 implies that this simulation is the maximal one. Finally, Lemma 11 implies that the resulting partition P_{sim} equals $S/\cong^I$. □

A.2 Complexity of Computing Simulations on LTS (Algorithm 1)

Data structures and important implementation details

We use resizable arrays (and matrices) which double (or quadruple) their size whenever needed. The insertion operation over these structures takes amortised constant (linear) time.

Each block B contains for each $a \in \Sigma$ a list of (pointers on) states from $Remove_a(B)$. Each time when any set $Remove_a(B)$ becomes nonempty, block B is moved to the beginning of the list of blocks. Choosing the pivot block on the line 3 then means just scanning the head of the list of blocks.

Each block contains, for each $a \in \Sigma$ and a state $q \in S$, a counter $RelCount_a(q,B) = |\{r \in S \mid r \in \bigcup Rel_a(B) \wedge q \xrightarrow{a} r\}|$. This counters enables us to perform the test on line 16 in $O(1)$ time.

The $Split(P, Remove)$ operation can be implemented as follows: Iterate through all $q \in Remove$. If $q \in B \in P$, add q into a block B_{child} (if B_{child} does not exist yet, create it and add it into P) and remove q from B. If B becomes empty, discard it.

At the initialization phase, we attach to each $q \in S$ an array indexed by symbols of $a \in \Sigma$ of pointers to $pre_a(q)$ lists. This way, we achieve constant time searching for $pre_a(q)$ lists (without the arrays, it would be $O(|\Sigma|)$).

Some auxiliary notions

For $B \subseteq S$ and $a \in \Sigma$, we denote by $in_a(B)$ the set $\{(r,a,q) \in \rightarrow \mid q \in B\}$, and by $in(B)$ the set $\bigcup_{a \in \Sigma} in_a(B)$. Note that $|pre_a(B)| \leq |in_a(B)|$. We also denote by $\xrightarrow{a}$ the set of all a-edges of $\rightarrow$.

We denote by $Anc(B)$ the set of all ancestors of B, and if B' is an ancestor of B, then B is a *descendent* of B'.

Proof of Theorem 2

Proof.

Initial observations

The complexity analysis builds a lot upon Lemma 12 and Lemma 6 proved within the proof of correctness of Algorithm 1. Using these lemmas, we can see that:

Observation 1. For any $a \in \Sigma$ and $B \in P_{sim}$, the sum of the cardinalities of the $Remove_a(B')$ sets for all $B' \in Anc(B)$ that are chosen as the pivot is below $|S|$.

Observation 2. If a pair (C,D) once appears on line 15, then any pair (C',D') such that $C \in Anc(C')$ and $D \in Anc(D')$ cannot appear on line 15 any more.

Most of the remaining complexity analysis then lies in a careful exploration of manipulations with the data structures used in the algorithm.

Space complexity

The arrays of pointers on the pre_a lists take $O(|\Sigma| \cdot |S|)$ space, the matrix encoding of *Rel* takes $O(|P_{sim}|^2)$ space, and the *Remove* sets as well as the counters take $O(|\Sigma| \cdot |P_{sim}| \cdot |S|)$ space. Thus the overall asymptotic space complexity is $O(|\Sigma| \cdot |P_{sim}| \cdot |S|)$.

Time complexity

The initialization of the arrays of pointers to the pre_a lists takes $O(|\Sigma| \cdot |S|)$ time. The *RelCount* counters are initialized by (1) setting all *RelCount* to 0, and then (2) for all $B \in P$, for all $q \in B$, for all $r \in pre_a(q)$, and for all C such that $(C,B) \in Rel$, incrementing $RelCount_a(r,C)$. This takes $O(|P_{init}| \cdot |{\rightarrow}|)$ time. The *Remove* sets are initialized by iterating through all $a \in \Sigma, q \in S, B \in P$, and if $RelCount_a(q,B) = 0$, then adding (appending) q to $Remove_a(B)$. This takes $O(|\Sigma| \cdot |P_{init}| \cdot |S|)$ time. Thus the overall initialization can be done in time $O(|P_{init}| \cdot |{\rightarrow}| + |\Sigma| \cdot |P_{init}| \cdot |S|)$.

One single $Split(P, Remove)$ operation takes $O(|Remove|)$ time. From Observation 1, we have that for a fixed block $B \in P_{sim}$ and $a \in \Sigma$, the sum of cardinalities of all $Remove_a(B')$ sets where B' is an ancestor of B according to which a *Split* is being done is below $|S|$. Therefore, for all symbols of Σ and all the blocks of P_{sim}, the overall time complexity of all *Split* operations is $O(|\Sigma| \cdot |P_{sim}| \cdot |S|)$.

The complexity analysis of lines 7–10 is based on the fact that it can happen at most $|P_{init}| - |P_{sim}|$ times that any block B is split. Moreover, the presented code can be optimised by not having the lines 7–10 as a separate loop (this was chosen just for clarity of the presentation), but the inheritance of *Rel*, *Remove*, and the counters can be done within the $Split$ function, and only for those blocks that were really split (not for all the blocks every time).

Whenever a new blocks is generated by *Split*, we have to do the following: (1) For each $a \in \Sigma$, copy the $Remove_a$ set of the parent block and attach the copy to the child block. As for all $a \in \Sigma, B \in P$, $Remove_a(B) \subseteq S$, and a new block will be generated at most $|P_{init}| - |P_{sim}|$ times, the overall time of this copying is in $O(|\Sigma| \cdot |P_{sim}| \cdot |S|)$. (2) Add a row and a column to the *Rel* matrix and copy the entries from those of the parent. This operation takes $O(|P_{sim}|)$ time for one added block as the size of the rows and columns of the *Rel*-matrix is bounded by $|P_{sim}|$. Thus. for all newly generated blocks, we achieve the overall time complexity of $O(|P_{sim}|^2)$. (3) Add and copy the *RelCount* counters. For one newly generated block, this operation takes an $O(|\Sigma| \cdot |S|)$ time and thus for all generated blocks, it gives time $O(|\Sigma| \cdot |P_{sim}| \cdot |S|)$.

Lines 13 and 14 are $O(1)$-time (*Rel* is a boolean matrix). Before we enter the for loop on line 11 with B being the pivot, we compute a list $RemoveList_a(B) = \{D \in P \mid D \subseteq Remove\}$. This is an $O(|Remove|)$ operation and by almost the same argument as in the case of the overall time complexity of *Split*, we get also exactly the same overall time complexity for computing all the $RemoveList_a(B)$ lists. On line 11, for each $q \in B$, we find the $pre_a(q)$ list (in $O(1)$ time using the array of pointers to the $pre_a(q)$ lists), and we iterate through all elements of $pre_a(q)$ and choose every $C, C \xrightarrow{a} \{q\}$. This takes $O(|in_a(B)|)$ time. For any $B \in P_{sim}$, let $RL_a(B)$ be the set of blocks $\bigcup_{B' \in Anc(B)} RemoveList_a(B')$. Then the overall time complexity of lines 11–14 is at most $O(\sum_{a \in \mathcal{L}} \sum_{B \in P_{sim}} |RL_a(B)| \cdot |in_a(B)|)$. From the initial observations, we can see that $|RL_a(B)| \leq |P_{sim}|$, and thus we have the overall time complexity of $O(\sum_{a \in \mathcal{L}} \sum_{B \in P_{sim}} |P_{sim}| \cdot |in_a(B)|) = O(\sum_{a \in \mathcal{L}} |P_{sim}| \cdot |\xrightarrow{a}|) = O(|P_{sim}| \cdot |\rightarrow|)$ for lines 11–14.

For a single C, D pair appearing on line 14, we iterate through all $q \in D$ and through all nonempty lists $pre_a(q)$, and for each $r \in pre_a(q)$, we decrement $RelCount_a(r, C)$. If $RelCount_a(r, C) = 0$ after the decrement, we append r into the $Remove_a(C)$ list. It follows from the initial observations that if any pair of blocks (C, D) once appears on line 14, then there will never appear any pair of their descendants on line 14. Thus, if we fix a block $C \in P_{sim}$ and a state q, then it can happen at most once that $q \in D$ and the pair (C', D) (where C' is an ancestor of C) is being separated within *Rel* (i.e. removed from *Rel*) on line 14. Thus, the contribution of the pair C, q to th etime complexity of lines 15–17 is $O(\sum_{a \in \Sigma} |pre_a(q)|)$. Therefore, the contribution of the C, r pairs for all $r \in S$ is $O(|\rightarrow|)$, and hence the overall time complexity of lines 15–17 is $O(|P_{sim}| \cdot |\rightarrow|)$.

From the above analysis, it follows that the overall time complexity of the algorithm is $O(|P_{sim}| \cdot |\rightarrow| + |\mathcal{L}| \cdot |P_{sim}| \cdot |S|)$. □

A.3 Correctness of Computing the Downward Simulation via LTS

Proof of Theorem 3

Proof.

(if) Suppose that $q^\bullet \preccurlyeq r^\bullet$. This means that there is a simulation $R^\bullet$ on $Q^\bullet$ such that $(q^\bullet, r^\bullet) \in R^\bullet$. We define D to be the smallest binary relation on Q such that $(q', r') \in D$ if $(q'^\bullet, r'^\bullet) \in R^\bullet$. Obviously, $(q, r) \in D$. We show that D is a downward simulation on Q which immediately implies the result.

Suppose that $(q', r') \in D$ and $(q_1, \ldots, q_n) \xrightarrow{f} q'$. Since $(q', r') \in D$ we know that $(q'^\bullet, r'^\bullet) \in R^\bullet$; and since $(q_1, \ldots, q_n) \xrightarrow{f} q'$ we know by definition of $A^\bullet$ that $q'^\bullet \xrightarrow{f} (q_1, \ldots, q_n)^\bullet$. Since $R^\bullet$ is a simulation, there are $r_1, \ldots, r_n \in Q$ with $r'^\bullet \xrightarrow{f} (r_1, \ldots, r_n)^\bullet$ and $((q_1, \ldots, q_n)^\bullet, (r_1, \ldots, r_n)^\bullet) \in R^\bullet$. Since $r'^\bullet \xrightarrow{f} (r_1, \ldots, r_n)^\bullet$ we have $(r_1, \ldots, r_n) \xrightarrow{f} r'$. Also, by definition of $A^\bullet$ we know that $((q_1, \ldots, q_n)^\bullet \xrightarrow{i} q_i^\bullet$ for each $i : 1 \leq i \leq n$. We observe that r_i is the only state such that $(r_1, \ldots, r_n)^\bullet \xrightarrow{i} r_i^\bullet$, and hence it must be the case that $(q_i^\bullet, r_i^\bullet) \in R^\bullet$. This means that $(q_i, r_i) \in D$ for each $i : 1 \leq i \leq n$.

(only if) Suppose that $q \preccurlyeq_{down} r$. This means that there is a simulation D on Q such that $(q, r) \in D$. We define $R^\bullet$ to be the smallest binary relation on $Q^\bullet$ such that $(q'^\bullet, r'^\bullet) \in R^\bullet$ if $(q', r') \in D^\bullet$, and $((q_1^\bullet, \ldots, q_n^\bullet), (r_1^\bullet, \ldots, r_n^\bullet)) \in R^\bullet$ if $(q_i, r_i) \in D$ for each $i : 1 \leq i \leq n$. Obviously, $(q, r) \in R^\bullet$. We show that $R^\bullet$ is a simulation on $Q^\bullet$ which immediately implies the result. In the proof, we consider two sorts of states in $A^\bullet$; namely those corresponding to states and those corresponding to left hand sides in A.

Suppose that $(q'^\bullet, r'^\bullet) \in R^\bullet$ and $q'^\bullet \xrightarrow{f} (q_1, \ldots, q_n)^\bullet$. Since $(q'^\bullet, r'^\bullet) \in R^\bullet$, we know that $(q', r') \in D$, and since $q'^\bullet \xrightarrow{f} (q_1, \ldots, q_n)^\bullet$, we know by definition of $A^\bullet$ that $(q_1, \ldots, q_n) \xrightarrow{f} q'$. Since D is a downward simulation, there are $r_1, \ldots, r_n \in Q$ with

$(r_1,\ldots,r_n) \xrightarrow{f} r'$ and $(q_i,r_i) \in D$ for each $i : 1 \leq i \leq n$. Since $(r_1,\ldots,r_n) \xrightarrow{f} r'$, we have $r'^{\bullet} \xrightarrow{f} (r_1,\ldots,r_n)^{\bullet}$. By definition of $R^{\bullet}$, it follows that $((q_1^{\bullet},\ldots,q_n^{\bullet}),(r_1^{\bullet},\ldots,r_n^{\bullet})) \in R^{\bullet}$.

Now, suppose that $((q_1^{\bullet},\ldots,q_n^{\bullet}),(r_1^{\bullet},\ldots,r_n^{\bullet})) \in R^{\bullet}$ and that $(q_1,\ldots,q_n)^{\bullet} \xrightarrow{i} q_i^{\bullet}$. By definition of $A^{\bullet}$ we know that $(r_1,\ldots,r_n)^{\bullet} \xrightarrow{i} r_i^{\bullet}$. Since $((q_1^{\bullet},\ldots,q_n^{\bullet}),(r_1^{\bullet},\ldots,r_n^{\bullet})) \in R^{\bullet}$, it follows by definition of $R^{\bullet}$ that $(q_i,r_i) \in D$ and hence also that $(q_i^{\bullet},r_i^{\bullet}) \in R^{\bullet}$. □

A.4 Complexity of Computing the Downward Simulation via LTS

Proof of Lemma 1

Proof. The state-list encoding of the LTS $A^{\bullet}$ can be obtained from the lhs-list encoding of A by the following steps:

1. for all $q \in Q$, add $q^{\bullet}$ into the state-list encoding of $A^{\bullet}$ (and also add an additional pointer from q to $q^{\bullet}$, which we will need later on), and
2. for each $l = (q_1,\ldots,q_n) \in Lhs(A)$,
   (a) add $l^{\bullet}$ into the state-list encoding of $A^{\bullet}$,
   (b) for each $f \in \Sigma$ and each right-hand side r in the f-list of l, add $r^{\bullet}$ into $pre_f(l^{\bullet})$, i.e. add the $r^{\bullet} \xrightarrow{f} l^{\bullet}$ edges, and
   (c) for each $1 \leq i \leq n$, add $l^{\bullet}$ into $pre_i(q_i^{\bullet})$, i.e. add the $l^{\bullet} \xrightarrow{i} q_i^{\bullet}$ edges[9].

In order to have a constant time access to the particular pre_a-lists for $a \in \Sigma^{\bullet}$ in the state-list encoding of $A^{\bullet}$ being built by the above construction, we may temporarily replace the state-lists by arrays. This means that we first construct, for each $q^{\bullet} \in Q^{\bullet}$ where $q \in Q$, a temporary array indexed by $i \in \Sigma^{\bullet}, 1 \leq i \leq Rank(A)$, of pointers to the $pre_i(q^{\bullet})$ lists (initialized with *null* values), and, for each $l^{\bullet} \in Q^{\bullet}$ where $l \in Lhs(A)$, a similar temporary array of pointers to the $pre_f(l)$-lists for $f \in \Sigma$. The time and space needed for creating these temporary arrays is $O(Rank(A) \cdot |Q| + |\Sigma| \cdot |Lhs(A)|)$.

After creating the temporary arrays, we traverse the lhs-list representation of A in time $O(|Q| + |\Delta| + Rank(A) \cdot |Lhs(A)|)$ while building the state-list representation (with arrays used instead of state-lists) of $A^{\bullet}$ with each step done in constant time (due to the use of the temporary arrays and the auxiliary pointers from q to $q^{\bullet}$). In the complexity, $|Q|$ corresponds to traversing the list of states, $|\Delta|$ to traversing the transitions of A while creating the f-labelled transitons of $A^{\bullet}$ for $f \in \Sigma$, and $Rank(A) \cdot |Lhs(A)|$ to traversing the left-hand sides while creating the i-labelled transitions of $A^{\bullet}$ for $1 \leq i \leq Rank(A)$. The remaining step is then to convert the auxiliary arrays into state-lists which can be done with the same complexity as initialising the arrays (we do not traverse the contents of the state-lists, we just leave out the state lists that are empty).

Thus, using suitable linked data structures, the creation of the state-list encoding of $A^{\bullet}$ is done in time $O(Rank(A) \cdot |Q| + |\Delta| + (Rank(A) + |\Sigma|) \cdot |Lhs(A)|)$.

The space complexity corresponds to the size of the temporary arrays and the size of the resulting LTS $A^{\bullet}$, which is $O(|Q| + |\Delta| + Rank(A) \cdot |Lhs(A)|))$. Indeed, we need space $O(|Q|)$ to represent states, $O(|\Delta|)$ to represent the f-labelled transitons of $A^{\bullet}$ for

[9] Here, we use the pointers from q to $q^{\bullet}$ introduced at the beginning.

$f \in \Sigma$, and $O(Rank(A) \cdot |Lhs(A)|)$ to represent the i-labelled transitions of $A^\bullet$ for $1 \leq i \leq Rank(A)$. In total, we obtain the same formula as in the case of the time complexity, i.e. $O(Rank(A) \cdot |Q| + |\Delta| + (|\Sigma| + |Rank(A)|) \cdot |Lhs(A)|)$.

Finally, the creation of $(P^\bullet, Rel^\bullet)$ is trivial, and its complexity is apparently covered by the complexity of creating $A^\bullet$. □

Proof of Lemma 2

Proof. We get the complexity of running Algorithm 1 on $A^\bullet$ and $(P^\bullet, Rel^\bullet)$ by instantiating the parameters of $A^\bullet$ in the formula of Theorem 2. More precisely, from the construction of $A^\bullet$, it follows that (1) $|\Sigma^\bullet| = |\Sigma| + Rank(A)$, (2) $|Q^\bullet| = |Q| + |Lhs(A)|$, and (3) $|\Delta^\bullet| \leq |\Delta| + Rank(A) \cdot |Lhs(A)|$. Then the running time of Algorithm 1 with input $A^\bullet$ and $(P^\bullet, Rel^\bullet)$ is:

$$O((|\Sigma| + Rank(A)) \cdot (|Q| + |Lhs(A)|) \cdot (|Q/{\cong_{down}}| + |Lhs(A)/{\cong_{down}}|) \\ + ((|\Delta| + Rank(A) \cdot Lhs(A)) \cdot (|Q/{\cong_{down}}| + |Lhs(A)/{\cong_{down}}|))).$$

Observe that $|Lhs(A)| \leq |\Delta|$ and that $|Q| \leq |Lhs(A)| + 1$[10]. Therefore, the time complexity amounts to

$$O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/{\cong_{down}}| + |\Delta| \cdot |Lhs(A)/{\cong_{down}}|)$$

and as the space complexity formula from Theorem 2 equals the first summand of the time complexity formula, we are getting the space complexity

$$O((|\Sigma| + Rank(A)) \cdot |Lhs(A)| \cdot |Lhs(A)/{\cong_{down}}|).$$

□

A.5 Correctness of Computing the Upward Simulation via LTS

Proof of Theorem 5

Proof.

(if) Suppose that $q^\odot \preccurlyeq^I r^\odot$. This means that there is a simulation $R^\odot \subseteq I$ on $Q^\odot$ such that $(q^\odot, r^\odot) \in R^\odot$. We define U to be the smallest binary relation on Q such that $(q', r') \in U$ if $(q'^\odot, r'^\odot) \in R^\odot$. Obviously, $(q, r) \in U$. We show that U is an upward simulation on Q induced by $\preccurlyeq_{down}$, which immediately implies the result.

Suppose that $(q', r') \in U$ and $(q_1, \ldots, q_n) \xrightarrow{f} q''$, where $q_i = q'$. Since $(q', r') \in U$, we know that $(q'^\odot, r'^\odot) \in R^\odot$, and since $(q_1, \ldots, q_n) \xrightarrow{f} q''$, we know by definition of $A^\odot$ that $q_i^\odot \xrightarrow{\lambda} ((q_1, \ldots, \Box_i, \ldots, q_n) \xrightarrow{f} q'')^\odot$. Since $R^\odot$ is a simulation, there are $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n, r'' \in Q$ with $r'^\odot \xrightarrow{\lambda} ((r_1, \ldots, \Box_i, \ldots, r_n) \xrightarrow{f} r'')^\odot$ and $(((q_1, \ldots, \Box_i, \ldots, q_n) \xrightarrow{f} q'')^\odot, ((r_1, \ldots, \Box_i, \ldots, r_n) \xrightarrow{f} r'')^\odot) \in R^\odot$. Since $R^\odot \subseteq I$, we know that $(((q_1, \ldots, \Box_i, \ldots, q_n) \xrightarrow{f} q'')^\odot, ((r_1, \ldots, \Box_i, \ldots, r_n) \xrightarrow{f} r'')^\odot) \in I$ and hence $(q_j, r_j) \in \preccurlyeq_{down}$ for each j such that $1 \leq j \neq i \leq n$. Since $r'^\odot \xrightarrow{\lambda} ((r_1, \ldots, \Box_i, \ldots, r_n) \xrightarrow{f}$

[10] Recall that we assume the automata not to have unreachable states and to have at most one state that is not used in any left-hand side.

$r'')^{\odot}$ we have $(r_1,\dots,r_n) \xrightarrow{f} r''$ where $r_i = r'$. Also, by definition of $A^{\odot}$ we know that the only transitions from $((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}$ resp. $((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot}$ are $((q_1,\dots,\Box,\dots,q_n) \xrightarrow{f} q'')^{\odot} \xrightarrow{f} q''^{\odot}$ resp. $((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot} \xrightarrow{f} r''^{\odot}$. Consequently, it must be the case that $(q''^{\odot}, r''^{\odot}) \in R^{\odot}$. This means that $(q'',r'') \in U$.

(only if) Assume that there is an upward simulation U on Q induced by $\preccurlyeq_{down}$ such that $(q,r) \in U$. We define $R^{\odot}$ to be the smallest binary relation on $Q^{\odot}$ such that $(q'^{\odot}, r'^{\odot}) \in R^{\odot}$ if $(q',r') \in U$, and $(((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}, ((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot}) \in R^{\odot}$ if $(q'',r'') \in U$ and $q_j \preccurlyeq_{down} r_j$ for each i such that $1 \le j \ne i \le n$. Obviously $R^{\odot} \subseteq I$ and $(q,r) \in R^{\odot}$. We show that $R^{\odot}$ is a simulation on $Q^{\odot}$ which immediately implies the result. In the proof, we consider two sorts of states in $A^{\odot}$; namely those corresponding to states and those corresponding to environments.

Suppose now that $(q'^{\odot}, r'^{\odot}) \in R^{\odot}$ and $q'^{\odot} \xrightarrow{\lambda} ((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}$. Since $(q'^{\odot}, r'^{\odot}) \in R^{\odot}$, we know that $(q',r') \in U$; and since $q'^{\odot} \xrightarrow{\lambda} ((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}$ we know by definition of $A^{\odot}$ that $(q_1,\dots,q_n) \xrightarrow{f} q''$ where $q' = q_i$. Since U is an upward simulation induced by $\preccurlyeq_{down}$, there are $r_1,\dots,r_n,r'' \in Q$ with $(r_1,\dots,r_n) \xrightarrow{f} r''$, $r_i = r'$, $(p'',r'') \in U$ and $q_j \preccurlyeq_{down} r_j$ for each $j : 1 \le j \ne i \le n$. Since $(r_1,\dots,r_n) \xrightarrow{f} r''$ we have $r'^{\odot} \xrightarrow{\lambda} ((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot}$. By definition of $R^{\odot}$ it follows that $(((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}, ((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot})$.

Now, suppose that $(((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}, ((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot}) \in R^{\odot}$ and that $(((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot} \xrightarrow{f} q''^{\odot}$. By definition of $A^{\odot}$, we know that $(((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot} \xrightarrow{f} r''^{\odot}$. Moeover, since $(((q_1,\dots,\Box_i,\dots,q_n) \xrightarrow{f} q'')^{\odot}, ((r_1,\dots,\Box_i,\dots,r_n) \xrightarrow{f} r'')^{\odot}) \in R^{\odot}$, it follows by definition of $R^{\odot}$ that $(q'',r'') \in U$ and hence also that $(q''^{\odot}, r''^{\odot}) \in R^{\odot}$. $\Box$

A.6 Complexity of Computing the Upward Simulation via LTS

Proof of Lemma 3

Proof. We assume to start with the lhs-list representation of $A = (Q,\Sigma,\Delta,F)$. We need to derive the LTS $A^{\odot}$ in the state-list format and the partition-relation pair $(P^{\odot}, Rel^{\odot})$. Algorithm 2 is a simplified encoding of the procedure. We know that $P^{\odot} = \{\{q^{\odot} \mid q \in Q\}\} \cup P_e^{\odot}$. Algorithm 2 computes $P_e^{\odot}$ using the partition $Lhs(A)/\cong_{down}$ constructed within the computation of the downward simulation on A. The state-list representation of LTS $A^{\odot}$ is created within this computation without increasing the overall asymptotic time complexity. The last step is then computing of $Rel^{\odot}$.

We denote two sets of environments *i-compatible* iff all their elements have the same symbol and the hole on the i^{th} position.

Lines 1–3 At the first step (lines 1–3) we compute for each $1 \le i \le Rank(A)$ a binary relations Rel_i on blocks of $Lhs(A)/\cong_{down}$ such that the partition-relation pair

Algorithm 2: Upward Initialization

```
    Input: A tree automaton $A = (Q,\Sigma,\Delta,F)$ and a partition $Lhs(A)/\cong_{down}$
    Data: for each $1 \le i \le Rank(A)$, a relation $Rel_i \subseteq Lhs(A)/\cong_{down} \times Lhs(A)/\cong_{down}$
    Output: The partition-relation pair $(P^{\odot}, Rel^{\odot})$ and the LTS $A^{\odot} = (Q^{\odot}, \Sigma^{\odot}, \Delta^{\odot})$
 1  forall $K, L \in Lhs(A)/\cong_{down}$ do
 2      forall $1 \le i \le Rank(A)$ do
 3          if $K \times L \subseteq D_i$ then $Rel_i \leftarrow Rel_i \cup \{(K,L)\}$
 4  $Q^{\odot} \leftarrow \{q^{\odot} \mid q \in Q\}; \Sigma^{\odot} \leftarrow \Sigma \cup \{\lambda\}; \Delta^{\odot} \leftarrow \emptyset$;
 5  forall $1 \le i \le Rank(A)$ do
 6      foreach equivalence class $\{L_1,\dots,L_m\} \in (Lhs(A)/\cong_{down})/(Rel_i \cap Rel_i^{-1})$ do
 7          merge $L_j$s into a new block of $Lhs(A)/\approx_i$, the block $B = \bigcup_{1 \le j \le m} L_j$;
 8          generate all maximal $i$-compatible sets $E$ such that $gen(E) = B$, update $A^{\odot}$ within
            this procedure. Then add $E$ into $P_e^{\odot}$;
 9  forall $1 \le i \le Rank(A)$ and all $i$-compatible blocks $E, E' \in P_e^{\odot}$ do
10      if $(gen(E), gen(E')) \in Rel_i$ then $Rel^{\odot} \leftarrow Rel^{\odot} \cup \{(E,E')\}$
11  $(P^{\odot}, Rel^{\odot}) \leftarrow (P_e^{\odot} \cup \{\{q^{\odot} \mid q \in Q\}\}, Rel^{\odot} \cup (\{q^{\odot} \mid q \in Q\}, \{q^{\odot} \mid q \in Q\}))$;
```

$(Lhs(A)/\cong_{down}, Rel_i)$ induces D_i. Here we exploit several properties of the structures we work with in order to decrease computational complexity:

1. For blocks K, L of $Lhs(A)/\cong_{down}$, the test on $K \times L \subseteq D_i$ can be done simply by testing any two representatives $k \in K, l \in L$ on $(k,l) \in D_i$. (it holds that $K \times L \subseteq D_i$ or $K \times L \cap D_i = \emptyset$)
2. For any left-hand sides k, l, there are three possibilities with respect to membership of (k,l) in D_i:
   (a) $(k,l) \in D_i$ for all i, i.e. k is simulated by l on all the positions ($(k,l) \in \cong_{down}$)
   (b) $(k,l) \in D_i$ for just one i, i.e. k is simulated by l on all positions except the i^{th} one
   (c) $(k,l) \notin D_i$ for all i, i.e. k is not simulated by l on more than one position.
   From item 1. we see that analogical relationships holds for any $K, L \in Lhs(A)/\cong_{down}$ with respect the $K \times L \subseteq D_i$ inclusions.

From these properties follows that given two blocks $K, L \in Lhs(A)/\cong_{down}$, the tests $K \times L \subseteq D_i$ can be done for all i in time $O(Rank(A))$ and, moreover, all the relations Rel_i can be stored in one common matrix with cells containing three types of values: `all`, `one`-i, `none`. This corresponds to the possibilities (a), (b), (c) from the above enumeration.

Therefore the line 3 can be done in constant time and thus the for loop on lines 1–3 can be finished in time $O(Rank(A) \cdot |Lhs(A)/\cong_{down}|^2)$. Furthermore, encoding of all the Rel_i relations takes only $O(|Lhs(A)/\cong_{down}|^2)$ space.

Lines 5–8 On lines 5–8 we construct partition $P_e^{\odot}$ together with LTS $A^{\odot}$. On line 6 we need to list all equivalence classes of $(Lhs(A)/\cong_{down})/(Rel_i \cap Rel_i^{-1})$. With the above matrix encoding of the Rel_i relations, this operation can be implemented in such a way that it takes $O(Rank(A) \cdot |Lhs(A)/\cong_{down}|^2)$ time overall.

Merging of the class $\{L_1,\dots,L_m\}$ on line 7 can be done in linear time to the cardinality of $\bigcup_{1\leq j\leq Rank(A)} L_j$ and therefore the overall time of the merging is $Rank(A)\cdot O(|Lhs(A)|)$ (the class $\{L_1,\dots,L_m\}$ can be encoded as a list of the L-blocks and each L-block can be encoded as a list of states).

On line 8 we generate all the environments of E and update $A^{\odot}$. We encode an environment e as a quadruple consisting of a pointer to any of $l\in gen(e)$, a symbol, a position of hole and a pointer to its right hand side state. We remind that we use the lhs-list encoding of A, i.e. each l is connected to a list indexed by symbols from Σ, where the f-indexed element contains the list of states $q\in Q$ such that $l\xrightarrow{f} q$. Thus for each $l\in B$, we can effectively iterate through all rules of the form $l\xrightarrow{f} q$ and for each of them we: (1.) create a new environment; and (2.) update $A^{\odot}$ in the following way:

(1.) We create a representation of environment e consisting of a pointer on l, symbol f, hole-index i and a pointer on q. A problem is that there can be more than one $l\in B$ such that $l\in gen(E)$. Thus we can obtain the same environment more than once while creating a block E from a block B. In order to avoid these duplicities, after having created e, we test if e has or has not been created before. This can by done by testing each newly created environment on membership in the set S of the so-far created environments (and adding it there if the membership test returns false).

We attempt to create a new environment (and add it to the set S of already known environments) $Rank(A)\cdot|\Delta|$ times. In the end (when $S=Env(A)$), we get $|Env(A)|$ different environments. We can assume that testing equality of two environments takes $O(Rank(A))$ time and that we use a set representation with a logarithmic membership test and addition. Thus, in total, the time $O(Rank(A)^2\cdot|\Delta|\cdot\log|Env(A)|)$ is spent by testing membership of environments in S and by extending S by the environments not yet there.

(2.) Having a representation of an environment $e=(q_1,\dots,\Box_i,\dots,q_n)\xrightarrow{f} q$ created, if $e\notin S$ (a representation of e was created for the first time), we add the state $e^{\odot}$ into $Q^{\odot}$ and also a pointer on $e^{\odot}$ into $pre_f(q)$. Then, regardless on the result of the $e\in S$ thest, we add the pointer on $q_i^{\odot}$ into $pre_{\lambda}(e^{\odot})$ (This requires finding the $pre_{\lambda}(e^{\odot})$ set in the state-set representation of $A^{\odot}$. We can use a similar searching structure as in the case of solving duplicities and then the complexity of this searching will be covered the complexity of solving duplicities.) As creating an $e^{\odot}$ state and adding an element into a *pre* set are constant time, the overall complexity of these updates of $A^{\odot}$ is covered by the complexity of the above creating of the elements of the E blocks.

Lines 9–10 On lines 9-10 we compute the main part of relation $Rel^{\odot}$. We exploit here the fact that for any i-compatible blocks $E,E'\in P_e^{\odot}$, $(E,E')\in Rel^{\odot}$ iff $gen(E)\times gen(E')\subseteq D_i$ and moreover that any $(B,C)\in\approx_i$ iff for any two $L,K\in Lhs(A)/{\cong_{down}}$ such that $K\subseteq B, L\subseteq C$, it holds that $K\subseteq L\in D_i$. As $K\times L\subseteq D_i$ means that $(K,L)\in Rel_i$, we can implement the test on line 10 this way:

When creating block E on line 7, we connect it with a representative block $repre(E)=L_j$ (any of $L_1\dots,L_m$). Then the test on line 10 can be done in constant time via testing if $(repre(E),repre(E'))\in Rel_i$, because we know that $(repre(E),repre(E'))\in Rel_i \iff (E,E')\in P_e^{\odot}$. Therefore lines 9–10 can be done in time $O(|P_e^{\odot}|^2)$.

Finishing construction of $(P^{\odot},Rel^{\odot})$ on line 11 is already easy. $\Box$

Proof of Lemma 4

Proof. We get the complexity of running Algorithm 1 on $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$ by instantiating the parameters of $A^{\odot}$ in the formula of Theorem 2. More precisely, from the construction of $A^{\odot}$, it follows that (1) $|\Sigma^{\odot}| = |\Sigma| + 1$, (2) $|Q^{\odot}| = |Q| + |Env(A)|$, and (3) $|\Delta^{\odot}| = Rank(A) \cdot |\Delta| + |Env(A)| \leq 2 \cdot Rank(A) \cdot |\Delta|$. Then, the running time of Algorithm 1 with the input $A^{\odot}$ and $(P^{\odot}, Rel^{\odot})$ is:

$$O(|\Sigma| \cdot (|Q| + |Env(A)|) \cdot (|Q/{\cong_{up}}| + |Env(A)/{\cong_{up}}|) \\ + Rank(A) \cdot |\Delta| \cdot (|Q/{\cong_{up}}| + |Env(A)/{\cong_{up}}|)).$$

Observe that, as we suppose the automata not to have unreachable states, $|Q| \leq |Env(A)|$. Therefore, the time complexity amounts to

$$O(|\Sigma| \cdot |Env(A)| \cdot |Env(A)/{\cong_{up}}| + Rank(A) \cdot |\Delta| \cdot |Env(A)/{\cong_{up}}|)$$

and, as the space complexity in Theorem 2 equals the first summand of the time complexity formula, we get the space complexity $O(|\Sigma| \cdot |Env(A)| \cdot |Env(A)/{\cong_{up}}|)$. $\square$

A.7 Reducing TA Using the Downward Simulation (Theorem 7)

In order to prove Theorem 7, we first show the following lemma.

Lemma 13. *For all q and r, if $q \preccurlyeq_{down} r$ then $L(q) \subseteq L(r)$.*

Proof. Suppose that $q \preccurlyeq_{down} r$ and $t \in L(q)$. We show that $t \in L(r)$ using induction on the structure of t. The base case (where t is empty) is trivial. We consider the case where t contains at least one node. We know that $t \overset{\pi}{\Longrightarrow} q$ for some π. with $\pi(\varepsilon) = q$. Let $t(\varepsilon) = f$. Furthermore, we know that there are $q_1, \ldots, q_n$ such that $(q_1, \ldots, q_n) \xrightarrow{f} q$, and $\pi(i) = q_i$ for each $i : 1 \leq i \leq n$. In other words, the run labels the root with q, and labels the children of the root with $q_1, \ldots, q_n$ respectively. This means that $t_i \in L(q_i)$ where t_i is the i^{th} subtree of t. Since $q \preccurlyeq_{down} r$ we know that there are $r_1, \ldots, r_n$ such that $(r_1, \ldots, r_n) \xrightarrow{f} r$ and $q_i \preccurlyeq_{down} r_i$ for each $i : 1 \leq i \leq n$. By the induction hypothesis, it follows that $t_i \in L(r_i)$, and hence $t \in L(r)$. $\square$

Proof of Theorem 7

Proof. The inclusion $L(A) \subseteq L(A\langle\cong_{down}\rangle)$ is obvious. We show that $A\langle\cong_{down}\rangle \subseteq L(A)$. First, we show that for any block B and $r \in B$ it is the case that $L(B) \subseteq L(r)$. Suppose that $t \in L(B)$. We show that $t \in L(r)$ using induction on the structure of t. The base case (where t is empty) is trivial. We consider the case where t has at least one node. We know that $t \overset{\pi}{\Longrightarrow} B$ for some π with $\pi(\varepsilon) = B$. Let $t(\varepsilon) = f$. Furthermore, we know that there are blocks $B_1, \ldots, B_n$ such that $(B_1, \ldots, B_n) \xrightarrow{f} B$, and $\pi(i) = B_i$ for each $i : 1 \leq i \leq n$. In other words, the run labels the root with B, and labels the children of the root with $B_1, \ldots, B_n$ respectively. This means that $t_i \in L(B_i)$ where t_i is the i^{th} subtree of t. Since $(B_1, \ldots, B_n) \xrightarrow{f} B$ we know that there are $q_1 \in B_1, \ldots, q_n \in B_n, q \in B$ such that $(q_1, \ldots, q_n) \xrightarrow{f} q$. By the induction hypothesis, we know that $t_i \in L(q_i)$. Since $q, r \in B$

it follows that $q \cong_{down} r$ and hence $q \preccurlyeq_{down} r$. It follows that there are $r_1, \ldots, r_n$ such that $(r_1, \ldots, r_n) \xrightarrow{f} r$ and $q_i \preccurlyeq_{down} r_i$ for each $i : 1 \leq i \leq n$. By Lemma 13 it follows that $t_i \in L(r_i)$ for each $i : 1 \leq i \leq n$, and hence $t \in L(r)$.

Now suppose that $t \in L(A\langle \cong_{down} \rangle)$. It follows that $t \in L(B)$ for some $B \in F\langle \cong_{down} \rangle$. Since Since $B \in F\langle \cong_{down} \rangle$, there is some $r \in B$ with $r \in F$. By the above property it follows that $t \in L(r)$, and hence This implies that $t \in L(A)$. □

A.8 Reducing TA Using the Upward Simulation (Lemma 5)

To prove Lemma 5, we need two auxiliary lemmas. We fix a reflexive and transitive downward simulation D and a reflexive and transitive upward simulation U induced by D included in I_F. Further, let $R \in (D \oplus U)$ and $\equiv_R$ be the equivalence relation defined by R.

Lemma 14. *If $c[q_1, q_2, \ldots, q_n] \Longrightarrow q$ and $(q_i, r_i) \in U$ for some $1 \leq i \leq n$, then there are states $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n, r$ such that $(q_j, r_j) \in D$ for each j such that $1 \leq j \neq i \leq n$, $(q, r) \in U$, and $c[r_1, \ldots, r_n] \Longrightarrow r$.*

Proof. To simplify the notation, we assume (without loss of generality) that $i = 1$. We use induction on the structure of c. The base case is trivial since the context c consists of a single hole. For the induction step, we assume that c is not only a single hole. Suppose that $c[q_1, q_2, \ldots, q_n] \overset{\pi}{\Longrightarrow} q$ for some run π and that $(q_1, r_1) \in U$. Let $p_1, \ldots, p_j$ be the left-most leaves of c with a common parent. Let p be the parent of $p_1, \ldots, p_j$. Notice that $q_1 = \pi(p_1), \ldots, q_j = \pi(p_j)$. Let $q' = \pi(p)$ and let c' be the context c with the leaves $p_1, \ldots, p_j$ deleted. In other words, $dom(c') = dom(c) \setminus \{p_1, \ldots, p_j\}$, $c'(p') = c(p')$ if $p' \in dom(c') \setminus \{p, p_1, \ldots, p_j\}$, and $c'(p) = \bigcirc$. Observe that $c'[q', q_{j+1}, \ldots, q_n] \Longrightarrow q$ and that $(q_1, q_2, \ldots, q_j) \xrightarrow{f} q'$ for some f. By definition of the upward simulation and the premise $(q_1, r_1) \in U$, it follows that there are $r_2, \ldots, r_n, r'$ such that $(q_2, r_2) \in D, \ldots, (q_j, r_j) \in D, (q', r') \in U$, and $(r_1, r_2, \ldots, r_j) \xrightarrow{f} r'$. Since c' is smaller than c, we can apply the induction hypothesis and conclude that there are $r_{j+1}, \ldots, r_n, r$ such that $(q_{j+1}, r_{j+1}) \in D, \ldots, (q_n, r_n) \in D, (q, r) \in U$, and $c'[r', r_{j+1}, \ldots, r_n] \Longrightarrow r$. The claim follows immediately. □

Lemma 15. *For blocks $B_1, \ldots, B_n, B \in Q\langle \equiv_R \rangle$ and a context c, if $c[B_1, \ldots, B_n] \Longrightarrow B$, then there exist states $r_1, \ldots, r_n, r \in Q$ such that $(B_1, r_1) \in D, \ldots, (B_n, r_n) \in D, (B, r) \in U$, and $c[r_1, \ldots, r_n] \Longrightarrow r$. Moreover, if $B \in F\langle \equiv_R \rangle$, then also $r \in F$.*

Proof. The claim is shown by induction on the structure of c. In the base case, the context c consists of a single hole. We choose any $q \in B \cap F$ provided that $B \cap F \neq \emptyset$, and any $q \in B$ otherwise. The claim holds obviously by reflexivity of D and U.

For the induction step, we assume that c is not only a single hole. Suppose that $c[B_1, \ldots, B_n] \overset{\pi}{\Longrightarrow} B$ for some run π. Let $p_1, \ldots, p_j$ be the left-most leaves of c with a common parent. Let p be the parent of $p_1, \ldots, p_j$. Notice that $B_1 = \pi(p_1), \ldots, B_j = \pi(p_j)$. Let $B' = \pi(p)$ and let c' be the context c with the leaves $p_1, \ldots, p_j$ deleted. In other words, $dom(c') = dom(c) \setminus \{p_1, \ldots, p_j\}$, $c'(p') = c(p')$ provided $p' \in dom(c') \setminus \{p, p_1, \ldots, p_j\}$, and $c'(p) = \bigcirc$. Observe that $c'[B', B_{j+1}, \ldots, B_n] \Longrightarrow B$. Since c' is smaller than c, we can apply the induction hypothesis and conclude that there are $v, q'_{j+1}, \ldots, q'_n, q'$ such that $(B', v) \in D, (B_{j+1}, q'_{j+1}) \in D, \ldots, (B_n, q'_n) \in D, (B, q') \in U$, $c'[v, q'_{j+1}, \ldots, q'_n] \Longrightarrow q'$, and if $B \cap F \neq \emptyset$, then $q' \in F$. It follows that there are $u \in B', q_{j+1} \in B_{j+1}, \ldots, q_n \in B_n, q \in B$ with $(u, v) \in D, (q_{j+1}, q'_{j+1}) \in D, \ldots, (q_n, q'_n) \in D$, and $(q, q') \in U$. By definition of $A\langle \equiv_R \rangle$, there are states $q_1 \in B_1, \ldots, q_j \in B_j$, and $z \in B'$ such that $(q_1, \ldots, q_j) \xrightarrow{f} z$ for some f. Since $D \subseteq R$ and $(u, v) \in D$, we get $(u, v) \in R$. Since $u, z \in B'$, it follows that $u \equiv_R z$ and hence $(z, u) \in R$. From transitivity of R, we get $(z, v) \in R$. From the definition of R, there is a state w such that $(z, w) \in D$ and $(v, w) \in U$. By the definition of downward simulation and premises $(z, w) \in D$ and $(q_1, \ldots, q_j) \xrightarrow{f} z$, there are states $r_1, \ldots, r_j$ with $(q_1, r_1) \in D, \ldots, (q_j, r_j) \in D$, and $(r_1, \ldots, r_j) \xrightarrow{f} w$. By Lemma 14 and premises $(v, w) \in U$ and $c'[v, q'_{j+1}, \ldots, q'_n] \Longrightarrow q'$, there are states $r_{j+1}, \ldots, r_n$, and r with $(q'_{j+1}, r_{j+1}) \in D, \ldots, (q'_n, r_n) \in D, (q', r) \in U$, and $c'[w, r_{j+1}, \ldots, r_n] \Longrightarrow r$. Finally, by transitivity of D and U, we get $(q_{j+1}, r_{j+1}) \in D, \ldots, (q_n, r_n) \in D$, $(q, r) \in U$. Moreover, by definition of U and the fact that $q' \in F$ if $B \cap F \neq \emptyset$, we get that $r \in F$ if $B \in F\langle \equiv_R \rangle$. The claim thus holds. □

Now we can give the proof of Lemma 5.

Proof of Lemma 5

Proof. Suppose that $t \overset{\pi}{\Longrightarrow} B$ for some π. Let $p_1, \ldots, p_n$ be the leafs of t, and let $\pi(p_i) = B_i$ for each $i : 1 \leq i \leq n$. Let c be the context we get from t by deleting the leaves $p_1, \ldots, p_n$. Observe that $c[B_1, \ldots, B_n] \overset{\pi}{\Longrightarrow} B$. It follows from Lemma 15 that there exist states $r_1, \ldots, r_n, r \in Q$ and $q_1 \in B_1, \ldots, q_n \in B_n, q \in B$ such that $(q_1, r_1) \in D, \ldots, (q_n, r_n) \in D, (q, r) \in U$, $c[r_1, \ldots, r_n] \Longrightarrow r$, and if $B \cap F \neq \emptyset$, then $r \in F$. By definition of $A\langle \equiv_R \rangle$, it follows that there are $q'_1 \in B_1, \ldots, q'_n \in B_n$ and $f_1, \ldots, f_n$ such that $\xrightarrow{f_i} q'_i$ for each i such that $1 \leq i \leq n$. We show by induction on i that for each i such that $1 \leq i \leq n$ there are states $u^i_1, \ldots, u^i_i, v^i_{i+1}, \ldots, v^i_n, w^i$ such that $(q'_1, u^i_1) \in D, \ldots, (q'_i, u^i_i) \in D, (q_{i+1}, v^i_{i+1}) \in D, \ldots, (q_n, v^i_n)) \in D, (r, w^i)) \in U$, and $c[u^i_1, \ldots, u^i_i, v^i_{i+1}, \ldots, v^i_n] \Longrightarrow w^i$. The base case where $i = 0$ is trivial. We consider the induction step. Since $D \subseteq R$ and $(q_{i+1}, v_{i+1}) \in D$, we get $(q_{i+1}, v_{i+1}) \in R$. Since $q_{i+1}, q'_{i+1} \in B_{i+1}$, we have that $q'_{i+1} \equiv_R q_{i+1}$ and hence $(q'_{i+1}, q_{i+1}) \in R$. By transitivity of R, it follows that $(q'_{i+1}, v_{i+1}) \in R$. By the definition of R, there is z_{i+1} such that $(q'_{i+1}, z_{i+1}) \in D$ and $(v_{i+1}, z_{i+1}) \in U$. By Lemma 14, there are $z_1, \ldots, z_i, z_{i+2}, \ldots, z_n, z$ such that $(u^i_1, z_1) \in D, \ldots, (u^i_i, z_i) \in D, (v^i_{i+2}, z_{i+2}) \in D, \ldots, (v^i_n, z_n) \in D, (w^i, z) \in U$, and $c[z_1, \ldots, z_n] \Longrightarrow z$. By transitivity of D and the premises (q'_j, u^i_j) and $(u^i_j, z_j) \in D$, we have $(q'_j, z_j) \in D$ for each $j : 1 \leq j \leq i$. By transitivity of D and the premises (q_j, v^i_j) and $(v^i_j, z_j) \in D$, we have $(q_j, z_j) \in D$ for each $j : i + 2 \leq j \leq n$. Define $u^{i+1}_j = z_j$ for $j : 1 \leq j \leq i + 1$; $v^{i+1}_j = z_j$ for $j : i + 2 \leq j \leq n$; and $w^{i+1} = z$.

The induction proof above implies that $c[u^n_1, \ldots, u^n_n] \Longrightarrow w^n$. From the definition of downward simulation and the premises $\xrightarrow{f_i} q'_i$ and $(q'_i, u^n_i) \in D$, it follows that $\xrightarrow{f_i} u^n_i$ for each $i : 1 \leq i \leq n$. It follows that $t = c[f_1, \ldots, f_n] \Longrightarrow w^n$. By definition of U and the fact that $r \in F$ if $B \cap F \neq \emptyset$, it follows that $\forall 1 \leq i \leq n.\ w^i \in F$ provided that $B \in F\langle \equiv_R \rangle$. Thus, in the claim of the lemma, it suffices to take $w = w^n$. □