Simulations and Antichains for Efficient Handling of Finite Automata

Lukáš Holík

March 8, 2011

Abstract

This thesis is focused on techniques for finite automata and their use in practice, with the main emphasis on nondeterministic tree automata. This concerns namely techniques for size reduction and language inclusion testing, which are two problems that are crucial for many applications of tree automata. For size reduction of tree automata, we adapt the simulation quotient technique that is well established for finite word automata. We give efficient algorithms for computing tree automata simulations and we also introduce a new type of relation that arises from a combination of tree automata downward and upward simulation and that is very well suited for quotienting. The combination principle is relevant also for word automata. We then generalise the so called antichain universality and language inclusion checking technique developed originally for finite word automata for tree automata. Subsequently, we improve the antichain technique for both word and tree automata by combining it with the simulation-based inclusion checking techniques, significantly improving efficiency of the antichain method. We then show how the developed reduction and inclusion checking methods improve the method of abstract regular tree model checking, the method that was the original motivation for starting the work on tree automata. Both the reduction and the language inclusion methods are based on relatively simple and general principles that can be further extended for other types of automata and related formalisms. An example is our adaptation of the reduction methods for alternating Büchi automata, which results in an efficient alternating automata size reduction technique.

Keywords

Finite automata, finite tree automata, alternating Büchi automata, nondeterminism, simulation, bisimulation, universality, language inclusion, antichain, quotienting, regular tree model checking.

Abstrakt

Cílem této práce je vývoj technik umožňujících praktické využití nedeterministických konečných automatů, zejména nedeterministických stromových automatů. Jde zvláště o techniky pro redukci velikosti a testování jazykové inkluze, jež hrají zásadní roli v mnoha oblastech aplikace konečných automatů. V oblasti redukce velikosti vycházíme z dobře známých metod pro slovní automaty které jsou založeny na relacích simulace. Navrhli jsme efektivní algoritmy pro výpočet stromových variant simulačních relací a identifikovali jsme nový typ relace založený na kombinaci takzvaných horních a dolních simulací nad stromovými automaty. Tyto kombinované relace jsou zvláště vhodné pro redukci velikosti automatů slučováním stavů. Navržený princip kombinace relací simulace je relevantní i pro slovní automaty. Náš přínos v oblasti testování jazykové inkluze je dvojí. Nejprve jsme zobecnili na stromové automaty takzvané protiřetězcové algoritmy, které byly původně navrženy pro slovními automaty. Dále se nám podařilo použitím simulačních relací výrazně zefektivnit protiřetězcové algoritmy pro testování jazykové inkluze jak pro slovní, tak pro stromové automaty. Relevanci našich technik pro praxi jsme demonstrovali jejich nasazením v rámci regulárního stromového model checkingu, což je verifikační metoda založená na stromových automatech. Použití našich algoritmů zde vedlo k výraznému zrychlení a zvětšení škálovatelnosti celé metody. Základní myšlenky našich algoritmů pro redukci velikosti automatů a testování jazykové inkluze jsou aplikovatelné i na jiné typy automatů. Příkladem jsou naše redukční techniky pro alternující Büchiho automaty prezentované v poslední části práce.

Klíčová slova

Konečný automat, konečný stromový automat, alternující Büchiho automat, nedeterminismus, univerzalita, jazyková inkluze, protiřetězec, simulace, bisimulace, redukce velikosti, regulární stromový model checking.

Citace

Lukáš Holík, Simulations and Antichains for Efficient Handling of Finite Automata, disertační práce, Brno, FIT VUT v Brně, 2010

Simulations and Antichains for Efficient Handling of Finite Automata

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením doc. Tomáše Vojnara. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Lukáš Holík
26. října 2010

Acknowledgements

I am most grateful to my advisor Tomáš Vojnar for his thoughtful approach and the enormous effort he spent when teaching me what it means to do research in computer science. I appreciate his trust that this investment would eventually pay off, which was a great source of motivation for me. I must also thank him for the opportunity to meet great people from our field, especially prof. Bouajjani, prof. Abdulla, doc. Habermehl, doc. Mayr. and also younger colleagues Dr. Kaati (the queen of tree automata), Dr. Chan., and Dr. Rogalewicz. I was continuously learning from them during our discussions, especially about the importance of talking and carefully listening to others. They deserve my thanks for always patiently listening to me (it was not always easy). I wish to express my gratitude to prof. Češka for his support and for his contribution towards creating an environment where a work such as mine is possible. I also thank my family for standing by me and for believing that the things I do make sense. I thank Marie for offering me a refuge and for teaching me the most essential things.

The work presented in this thesis was supported by the Czech Science Foundation (projects 102/07/0322, 102/09/H042, 103/10/0306), the Czech institutional project MSM 0021630528, the Barrande projects MEB 020840 and 2-06-27, the Czech COST project OC10009 associated with the ESF COST action IC0901, the internal BUT FIT grant FIT-S-10-1, and the ESF project Games for Design and Verification.

Contents

1 Introduction

Finite automata on finite words (FA) are one of the basic concepts of computer science. Besides classical applications of FA such as compiler construction or text searching, FA are widely used in modelling and verification, which are the application domains of our interest. Tree automata (TA) are a natural generalisation of FA that accepts ordered trees/terms. TA share most of the good properties of FA, from closure to decidability and complexity (even though complexities of many tree automata problems are higher, they are still comparable with the complexities of the corresponding FA ones). This makes tree automata a convenient tool 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, e.g., [CDG$^+$07]). One of the main motivations for this work is in particular the use of tree automata in verification, mainly in the method of regular tree model checking [Sha01, BT02, ALdR05, BHRV06a], an infinite-state system verification method where tree automata are used for representing sets of reachable states of a system.

In the above context, checking language equivalence/inclusion and reducing size of automata while preserving the language are fundamental issues, and performing these operations efficiently is crucial in practice. The language inclusion problem and the minimisation problem for (nondeterministic) automata are PSPACE-complete for FA and even EXPTIME-complete for TA. A classical approach to cope with these problems is determinisation. Both FA as well as TA can be determinised and minimised in a canonical way. Testing language inclusion of deterministic minimal automata is then easy. However, since even the canonical minimal deterministic automaton can still be exponentially larger than the original nondeterministic one, its computation easily becomes a major bottleneck of any automata-based method.

A reasonable and pragmatic approach to the size reduction and language inclusion problem is to consider some relation on states of an automaton that respects language inclusion on states, but which can be checked efficiently, using a polynomial algorithm. Such a relation can then be used for approximating language inclusion between two automata by checking whether each initial state of one automaton is related to an initial state of other automaton. This method is sound but incomplete in the case when the relation is a proper subset the language inclusion on states. Such a relation can be also used for reducing the size of an automaton by collapsing equivalent states. Here, a natural trade-off between the strength of the considered relation and the cost of its computation arises. In the case of word automata, a relation which is widely considered as a good trade-off in this sense is simulation preorder. It can be checked in polynomial time, and efficient algorithms have been designed for this purpose (see, e.g., [GPP03, HHK95, RT07, CRT09]). These algorithms make the computation of simulation preorder quite affordable even in comparison with the one of bisimulation equivalence, which is cheaper [Hop71, PT87, Val09], but which is also stronger, and therefore leads to less significant reductions of automata and also its capability of approximating language inclusion is limited.

As for what concerns language inclusion and universality problem, apart from the classical determinisation-based methods and simulation-based approximation technique, there has recently been proposed the so called antichain universality and inclusion testing method for FA [WDHR06]. It is essentially an optimisation of the classical method based on subset construction (i.e., on determinisation), it is still of an exponential worst case complexity, but it behaves very well in practice.

In the case of tree automata, the only methods for size reduction that were previously studied (apart from deterministic minimisation) are based on bisimulation relations [AHK07, HMM07a] and concerning language inclusion testing, the only methods formerly available are the classical ones based on explicit determinisation. However, these methods are not efficient enough. The former ones are rather weak since bisimulation relations are usually relatively sparse and the latter ones suffer from the problem of state space explosion too often.

1.1 Goals of the Thesis

The lack of efficient methods for reducing size and testing language inclusion of nondeterministic tree automata described above has significantly limited their practical usability. Therefore, this thesis is aimed at adapting techniques that work well for word automata to tree automata, which in particular concerns the size reduction methods based on simulations and the language inclusion testing algorithms based on the antichain principle. Then, apart from generalising existing methods from word automata to tree automata, we also focus on improving the existing methods themselves. This concerns introduction of new types of relations suitable for reducing the size of word as well as tree automata and interconnecting the antichain principle with the simulation techniques into new language inclusion testing algorithms. Additionally, we show that the proposed methods are applicable to other kinds of automata too by designing a simulation-based reduction method for alternating Büchi automata that is similar to the one we proposed for tree automata.

1.2 An Overview of Achieved Results

Here we summarise the contributions that we have achieved within the particular areas marked out by the goals of this work.

Tree Automata Reduction Methods. Our tree automata reduction methods are build on the notions of downward and upward tree automata simulations (proposed first in [ALdR05]) that are the tree automata counterparts the forward and backward FA simulations.

We design efficient algorithms for computing tree automata simulations. A deep examination of the structure of the TA simulations reveals that both upward and downward TA simulations can be computed by the same algorithmic pattern. More specifically, the problems of computing a TA simulation can be reduced to a problem of computing a common FA simulation (a tree automaton is translated into an FA and then a common FA simulation algorithm is used). Moreover, tree automata bisimulations can also be computed efficiently this way using the same translations (instead of a simulation algorithm, an FA bisimulation algorithm is run on the FA obtained by translating the input TA). The resulting tree automata bisimulation algorithms are simple and competitive with the previously known algorithms from [HMM07a]. This results in a uniform and elegant framework for computing tree automata simulations and bisimulations that can utilise the best FA simulation and bisimulation algorithms.

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

Language Inclusion Checking for TA and FA. Our universality and language inclusion algorithms for tree and word automata build on the antichain based method for FA proposed first in [WDHR06]. It is a complete method that optimises the classical subset construction based algorithms. We first briefly review its main idea.

Consider a nondeterministic FA $\mathcal{A}$. In the simpler case of universality checking, the method is based on a search for a nonaccepting state of the determinised version $\mathcal{A}'$ of $\mathcal{A}$ reachable from an initial state of $\mathcal{A}'$. Such a state is a counterexample to universality of $\mathcal{A}$. When a counterexample is reached, the algorithm may terminate even before all states of $\mathcal{A}'$ are constructed. The states of $\mathcal{A}'$, called macro-states, have the form of subsets of the set of states of $\mathcal{A}$. The key idea is that some macro-states have a better chance of finding a counterexample than other ones since they have provably smaller languages (in our terminology, we say that they subsume the states with larger languages). Therefore, one can safely continue searching only from the generated macro-states that have minimal languages, and simply discard any generated macro-state that is subsumed by another one. In [WDHR06], the subsumption relation is just set inclusion, and already this simple solution gives a fundamental speedup.

We first adapt the FA antichain technique for tree automata. The adaptation is quite straightforward, and similarly as in the case of FA, it has a major impact on efficiency of the TA language inclusion and universality tests. We then improve the antichain technique for both FA and TA by interconnecting it with the simulation approximation technique. Simply speaking, we improve accuracy of the subsumption relation on macro-states by employing simulations on states of the original automaton. In the case of universality checking, a macro-state p subsumes a macro state q if all states in p are simulated by some state in q. Moreover, even the internal structure of macro-states can be simplified by keeping only simulation maximal states of $\mathcal{A}$ inside the macro-states. In the case of testing inclusion between two automata $\mathcal{A}$ and $\mathcal{B}$, macro-states have a more complicated structure, and it is possible to utilise simulation on states of $\mathcal{A}$, on states of $\mathcal{B}$, and also use simulation between states of $\mathcal{A}$ and $\mathcal{B}$. It can be said that this method combines advantages of both simulation approximation of language inclusion and the original antichain technique. It also behaves very well on our experimental data.

Simulations and Antichains in Abstract Regular Tree Model Checking. We have shown practical applicability of our tree automata reduction and inclusion testing methods in the framework of abstract regular tree model checking (ARTMC), an infinite state verification method where the two problems play a crucial role. In regular model checking (RMC), we start with an FA $\mathcal{A}_I$ representing a set of initial configurations I of a system and iteratively apply transition relation τ (symbolically, on the structure of the automaton) until a fixpoint is reached, thus computing an FA representing the set $\tau^*(\mathcal{A}_I)$ of all configurations reachable from the initial configurations. Then, it is checked whether this set satisfies the verified properties. In abstract regular model checking [BHV04], abstraction (together with a counterexample guided refinement) is used to accelerate the computation. Checking the fixpoint condition means to decide whether $\tau^i(A_I) \subseteq \tau^{i+1}(\mathcal{A}_I)$, which requires an efficient language inclusion algorithm. During the computation, the intermediate automata typically grow quickly, therefore it is needed to reduce their size. Tree automata are used instead of FA when configurations of the system being verified are better represented by trees than by words, e.g., certain parametrised communication protocols, pointer programs manipulating tree-like data structures etc.

In that case, we speak about abstract regular tree model checking (ARTMC) [BT02, AJMd02, BHRV06a, BHRV06b]. This method was originally based on deterministic tree automata, involving implicit determinisation after each step. Our reduction and inclusion testing methods allowed us to redesign the method on top of nondeterministic tree automata, which led to a major increase of scalability and efficiency.

Simulations and Antichains for Other Types of Automata. The principles of our simulation-based reduction methods are relatively simple and general which allows extensions of the methods also for other types of automata. We have done this for alternating Büchi automata (ABA), for which we have designed simulation-based reduction method analogical to the one proposed for tree automata. ABA are acceptors of infinite words with the same expressive power as Büchi automata, but may be exponentially more succinct. Their applications can be found for instance in automata-based LTL model checking within a Büchi automata complementation procedure (e.g., [KV01]). Alternating Büchi automata are similar to tree automata in the sense that runs of both types of automata have a form of trees (ordered trees for TA and unordered trees for ABA). Therefore, the definitions of simulations look similar for the two types of automata. Forward simulation over alternating Büchi automata have been already studied (see [FW02, FW05]). It may bee seen as an analogy of the tree automata downward simulation. We have introduced the notion of ABA backward simulation, which is an analogy of TA upward simulation. We also show that it is possible to combine the ABA simulations in the same way as the TA simulation into a mediated equivalence suitable for collapsing states while preserving language. This equivalence gives better reductions than sole forward simulation, which we confirm also by experiments.

Generalisations of our universality and language inclusion algorithms are also possible. We are currently exploring ways of applying these techniques at deciding Büchi automata universality and language inclusion. Our first result has been published as [ACC$^+$10a] where we use the simulation subsumption technique to improve the so called Ramsey-based Büchi universality and inclusion test (see, e.g., [SVW85, FV09]). However, this work is already beyond the scope of this thesis.

1.3 Plan of the Thesis

Chapter 2 contains preliminaries on automata, simulations, and regular tree model checking. Chapter 3 presents an algorithm for computing simulations over labelled transition systems used within most of the algorithms presented further. In Chapter 4, we describe our simulation and bisimulation-based framework for reducing tree automata and the algorithms for computing the TA simulations and bisimulations. Chapter 5 deals with the language inclusion and universality problems for FA and TA. Alternating Büchi automata simulation-based reduction methods are discussed in Chapter 6 and Chapter 7 concludes the thesis.

2 Preliminaries

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

2.1 Relations

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

2.2 Labelled Transition Systems and Finite Automata

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

A *Nondeterministic Finite Automaton (FA)* $\mathcal{A}$ is a tuple $(\Sigma, Q, \delta, I, F,)$ where (Σ, Q, δ) is a labelled transition system, $I \subseteq Q$ is a non-empty set of *initial* states, and $F \subseteq Q$ is a set of *final* states.

A word $u = u_1 \ldots u_n$ is accepted by $\mathcal{A}$ from the state q_0 if there exists a sequence $q_0 u_1 q_1 u_2 \ldots u_n q_n$ such that $q_n \in F$ and $q_{j-1} \xrightarrow{u_j} q_j$ for all $0 < j \leq n$. The *language of a state* q in $\mathcal{A}$ is defined as $L(\mathcal{A})(q) := \{u \mid u \text{ is accepted by } \mathcal{A} \text{ from the state } q\}$ and the *language of the automaton* $\mathcal{A}$ is $L(\mathcal{A}) := \bigcup_{q \in I} L(\mathcal{A})(q)$. We say that $\mathcal{A}$ is *universal* if $L(\mathcal{A}) = \Sigma^*$.

2.3 Forward and Backward Simulations

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

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

I and denote $\preccurlyeq^I$ in the sequel. We use $\cong$ to denote the *simulation equivalence* $\preccurlyeq \cap \preccurlyeq^{-1}$ on Q and, consequently, $\cong^I$ to denote the *simulation equivalence* $\preccurlyeq^I \cap (\preccurlyeq^I)^{-1}$ *included in* I.

A *(forward) simulation on an FA* $\mathcal{A} = (\Sigma, Q, \delta, I, F)$ is a simulation on the LTS (Σ, Q, δ) such that if a state r simulates a state q, then $q \in F \implies r \in F$ (a simulation included in $(Q \times Q) \setminus (F \times (Q \setminus F))$). The following is a well-known lemma.

Lemma 2.1. *Given a forward simulation* $\preceq$ *on an FA* $\mathcal{A} = (\Sigma, Q, \delta, I, F)$, $p \preceq r \implies L(\mathcal{A})(p) \subseteq L(\mathcal{A})(r)$.

Backward simulation is a dual notion to forward simulation. A *backward simulation* over an LTS $\mathcal{T} = (\Sigma, Q, \delta)$ is a forward simulation over the LTS $\mathcal{T}^{-1} = (\Sigma, Q, \delta^{-1})$ where $\delta^{-1} = \{(p, a, q) \mid (q, a, p) \in \delta\}$, and a *backward simulation on an FA* $\mathcal{A} = (\Sigma, Q, \delta, I, F)$ is a forward simulation on the FA $\mathcal{A}^{-1} = (\Sigma, Q, \delta^{-1}, F, I)$. The notions of *backward simulation preorder* and *backward simulation equivalence* can be defined in the same way as for forward simulation and an equivalent of Lemma 2.1 holds for backward simulation and "backward languages" of states of $\mathcal{A}$.

2.4 Trees and Tree Automata

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

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

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

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

A *run* of $\mathcal{A}$ over a tree $t \in T(\Sigma)$ is a mapping $\pi : dom(t) \to Q$ such that, for each node $p \in dom(t)$ where $q = \pi(p)$, if $q_i = \pi(pi)$ for $1 \leq i \leq n$, then Δ

contains a rule $(q_1, \ldots, q_n) \xrightarrow{t(p)} q$. We write $t \overset{\pi}{\Longrightarrow} q$ to denote that π is a run of $\mathcal{A}$ over t such that $\pi(\epsilon) = q$. We use $t \Longrightarrow q$ to denote that $t \overset{\pi}{\Longrightarrow} q$ for some run π. The *language accepted at a state* q is defined by $L(q) = \{t \mid t \Longrightarrow q\}$, while the *language* of $\mathcal{A}$ is defined by $L(\mathcal{A}) = \bigcup_{q \in F} L(q)$.

2.5 Regular Tree Model Checking

Regular tree model checking (RTMC) [Sha01, BT02, ALdR05, BHRV06a] is a general and uniform framework for verifying infinite-state systems. In RTMC, configurations of a system being verified are encoded by trees, sets of the configurations by tree automata, and transitions of the verified system by a term rewriting system (usually given as a tree transducer or a set of tree transducers). Then, verification problems based on performing reachability analysis correspond to computing closures of regular languages under rewriting systems, i.e., given a term rewriting system τ and a regular tree language I, one needs to compute $\tau^*(I)$ where τ^* is the reflexive-transitive closure of τ. This computation is impossible in general. Therefore, the main issue in RTMC is to find accurate and powerful fixpoint acceleration techniques helping the convergence of computing language closures. One of the most successful acceleration techniques used in RTMC is abstraction whose use leads to the so-called *abstract regular tree model checking* (ARTMC) [BHRV06a], on which we concentrate in this work.

Abstract Regular Tree Model Checking. We briefly recall the basic principles of ARTMC in the way they were introduced in [BHRV06a]. Let Σ be a ranked alphabet and $\mathbb{M}_\Sigma$ the set of all tree automata over Σ. Let $\mathcal{I} \in \mathbb{M}_\Sigma$ be a tree automaton describing a set of initial configurations, τ a term rewriting system describing the behaviour of a system, and $\mathcal{B} \in \mathbb{M}_\Sigma$ a tree automaton describing a set of bad configurations. The safety verification problem can now be formulated as checking whether the following holds:

$$\tau^*(\mathcal{L}(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) = \emptyset \tag{2.1}$$

In ARTMC, the precise set of reachable configurations $\tau^*(\mathcal{L}(\mathcal{I}))$ is not computed to solve Problem (2.1). Instead, its overapproximation is computed by interleaving the application of τ and the union in $\mathcal{L}(\mathcal{I}) \cup \tau(\mathcal{L}(\mathcal{I})) \cup \tau(\tau(\mathcal{L}(\mathcal{I}))) \cup \ldots$ with an application of an abstraction function α. The abstraction is applied on the tree automata encoding the so-far computed sets of reachable configurations.

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

For a given abstraction function α, one can compute iteratively the sequence of automata $(\tau_\alpha^i(\mathcal{I}))_{i \geq 0}$. If the abstraction α is finitary, then there exists $k \geq 0$

such that $\tau_\alpha^{k+1}(\mathcal{I}) = \tau_\alpha^k(\mathcal{I})$. The definition of the abstraction function α implies that $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \supseteq \tau^*(\mathcal{L}(\mathcal{I}))$.

If $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$, then Problem (2.1) has a positive answer. If the intersection is non-empty, one must check whether a real or a spurious counterexample has been encountered. The spurious counterexample may be caused by the used abstraction (the counterexample is not reachable from the set of initial configurations). Assume that $\mathcal{L}(\tau_\alpha^k(\mathcal{I})) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$, which means that there is a symbolic path:

$$\mathcal{I},\ \tau_\alpha(\mathcal{I}),\ \tau_\alpha^2(\mathcal{I}), \ldots, \tau_\alpha^{n-1}(\mathcal{I}),\ \tau_\alpha^n(\mathcal{I}) \tag{2.2}$$

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

Let $X_n = \mathcal{L}(\tau_\alpha^n(\mathcal{I})) \cap \mathcal{L}(\mathcal{B})$. Now, for each l, $0 \leq l < n$, $X_l = \mathcal{L}(\tau_\alpha^l(\mathcal{I})) \cap \tau^{-1}(X_{l+1})$ is computed. Two possibilities may occur: (a) $X_0 \neq \emptyset$, which means that Problem (2.1) has a negative answer, and $X_0 \subseteq \mathcal{L}(\mathcal{I})$ is a set of dangerous initial configurations. (b) $\exists m, 0 \leq m < n, X_{m+1} \neq \emptyset \wedge X_m = \emptyset$ meaning that the abstraction function is too rough—one needs to refine it and start the verification process again.

In [BHRV06a], two general-purpose kinds of abstractions are proposed. Both are based on *automata state equivalences*. Tree automata states are split into several equivalence classes, and all states from one class are collapsed into one state. An abstraction becomes finitary if the number of equivalence classes is finite. The refinement is done by refining the equivalence classes. Both of the proposed abstractions allow for an automatic refinement to exclude the encountered spurious counterexample.

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

3 Computing Simulations over Labelled Transition Systems

This chapter is devoted to an algorithm for computing simulations onlabelled transition systems. As discussed in the previous chapter, simulation is a good candidate for reducing transition systems by collapsing equivalent states and also for approximating language/trace inclusion. It strongly preserves logics like $ACTL^*$, $ECTL^*$, and LTL [DGG93, GL94, HHK95], and with respect to its reduction power and computation cost, it offers a desirable compromise among the other common candidates, such as bisimulation equivalence [PT87, SJ05] and language equivalence. Our main motivation for presenting the algorithm here is that computing simulation over an LTS is a crucial step of almost all algorithms presented later in this thesis, namely algorithms for computing simulations over tree automata, alternating Büchi automata, and for checking language inclusion and universality of finite word and tree automata.

Our LTS simulation algorithm is a relatively straightforward modification of the algorithm by Ranzato and Tapparo from [RT07] (referred to as RT in the following) for computing simulations over Kripke structures (a Kripke structure associate labels with states while an LTS attaches labels to transitions). Given a Kripke structure $\mathcal{K}$ with a set of states Q and a transition relation δ such that P_{sim} is the partition of Q according to simulation equivalence, RT runs in time $\mathcal{O}(|P_{sim}||\delta|)$ and space $\mathcal{O}(|P_{sim}||Q|)$. This algorithm refines the algorithm [HHK95] by Henzinger, Henzinger, and Kopke (referred to as HHK) with running time $\mathcal{O}(|Q||\delta|)$ and space $\mathcal{O}(|Q|^2)$. The main difference between HHK and RT is that instead of manipulating individual states, RT works on the level of iteratively refined equivalence classes of a relation that finally converges to simulation equivalence. We have chosen RT since it is the fastest known simulation algorithm. However, there are other algorithms that are slower but more space efficient. The algorithm with the lowest space complexity among all known simulation algorithms is the one by Gentiliny, Piazza, and Policriti [GPP03]. It runs in time $\mathcal{O}(|P_{sim}|^2|\delta|)$ and space $\mathcal{O}(|P_{sim}|^2 + |Q|\log|P_{sim}|)$. Then, there is a recent algorithm [CRT09] by Crafa, Ranzato, and Tapparo, which improves on space complexity of RT, reducing it to $\mathcal{O}(|P_{sim}||P_{rel}|)$, which is very close to the space complexity of the algorithm by Gentiliny et al., however, the price of this is a worse time complexity $\mathcal{O}(|P_{sim}||\delta| + |P_{sim}|^2|\delta_{rel}|)$. Here, P_{rel} is a certain partition of the set of states of $\mathcal{K}$ such that $|P_{sim}| \leq |P_{rel}| \leq |Q|$ and δ_{rel} is a partition of the set of transitions where $|\delta_{rel}| \leq |\delta|$.

In fact, any algorithm computing simulation over Kripke structures can be used for computing simulations on labelled transition systems. Every LTS $\mathcal{T}$ with n states and m transitions can be easily translated into a Kripke structure $\mathcal{K}_\mathcal{T}$ with $m + n$ states and $2m$ transitions (we turn every transition $q \xrightarrow{a} r$ of $\mathcal{T}$ into the two transitions $q \rightarrow (q, a, r) \rightarrow r$ where (q, a, r) is a new state with label a) such that the simulation on states of $\mathcal{K}_\mathcal{T}$ directly gives simulation on $\mathcal{T}$. However, observe that this increase in the number of states significantly affects complexity of the overall procedure. In the case of RT, the time and space complexity of computing simulation on $\mathcal{T}$ this way (running RT on $\mathcal{K}_\mathcal{T}$) would be almost the square of m, which is much worse than for Kripke structures.

We design our version of RT that runs directly on an LTS to eliminate this increase of complexity. This basically requires augmenting most of the data structures of RT by alphabet symbols and iterating certain subprocedures for all incoming/outgoing symbols of a state or a set of states. We obtain an algorithm that runs in time $\mathcal{O}(|P_{sim}||Q|+|\Sigma||P_{sim}||\delta|)$ and space $\mathcal{O}(|\Sigma||P_{sim}||Q|)$ where Σ is the alphabet. The modifications of RT are rather easy, nevertheless, notice that the dominating factor $|P_{sim}||\delta|$ of the time complexity formula is not multiplied by the size of the alphabet, which requires a sensitive approach when manipulating certain data structures. Apart from that, we provide a more straightforward (and abstract interpretation free) proof of correctness of the algorithm than the one in [RT07].

We also note that in [HŠ09a], we present an improved version of our LTS simulation algorithm where we to a large degree eliminate the multiplicative effect of the size of the alphabet in the complexity formulas. This algorithm can even turn nonuniformity of input and output symbols of states into an advantage. However, since the improvements described in [HŠ09a] are not essential for the rest of this work and are rather technical, we present only the original simpler version of the algorithm here.

3.1 Preliminaries

We first introduce some additional notation used within the chapter and the notion of partition-relation pair.

Given an LTS $\mathcal{T} = (\Sigma, Q, \delta)$, we define the set of *a-predecessors* of a state r as $pre_a(r) = \{q \in Q \mid q \xrightarrow{a} r\}$. Given $X, Y \subseteq Q$, we use $pre_a(X)$ to denote the set $\bigcup_{q \in X} pre_a(q)$, we write $q \xrightarrow{a} X$ iff $q \in pre_a(X)$, and $Y \xrightarrow{a} X$ iff $Y \cap pre_a(X) \neq \emptyset$.

Partition-Relation Pairs. A *partition-relation pair* over a set X is a pair $\langle P, Rel\rangle$ where (1) $P \subseteq 2^X$ is a partition of X (i.e., $X = \bigcup_{B \in P} B$, 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 $\langle P, Rel\rangle$ over X *induces* (or defines) the relation $R_{\langle P,Rel\rangle} = \bigcup_{(B,C) \in Rel} B \times C$.

A partition-relation pair $\langle P, Rel\rangle$ over X inducing a relation R is the *coarsest* iff there is no other partition-relation pair inducing R with the partition coarser than P. This means that $P = \{\{y \in X \mid R(x) = R(y) \wedge R^{-1}(x) = R^{-1}(y)\} \mid x \in X\}$—two elements of X are in the same block of P iff they are related by R with elements of X in the same way. Notice that in the case when R is a preorder, P is the set of equivalence classes of $R \cap R^{-1}$ and Rel is a partial order.

3.2 The LTS Simulation Algorithm

We now describe an algorithm to compute simulation over LTS. For the rest of this chapter, we assume that we are given an LTS $\mathcal{T} = (\Sigma, Q, \delta)$ and the coarsest partition-relation pair $\langle P_I, Rel_I \rangle$ inducing an initial preorder $I \subseteq Q \times Q$. Our algorithm takes $\mathcal{T}$ and $\langle P_I, Rel_I \rangle$ as the input and outputs the coarsest partition-relation pair $\langle P_{sim}, Rel_{sim} \rangle$ inducing the simulation preorder $\preccurlyeq^I$ on $\mathcal{T}$ included in I. Algorithm 1 describes the algorithm in pseudocode. Before we discuss it in detail and analyse its correctness and complexity, we give a brief outline.

The algorithm propagates the negative information about which pair of states are not related by simulation. It iteratively refines a partition-relation pair $\langle P, Rel \rangle$ (strengthening the induced relation) initialised as $\langle P_I, Rel_I \rangle$. The induced relation is always superset of the target simulation, the states belonging to a block $B \in P$ are those which are currently assumed as being possibly simulated by states from $\bigcup Rel(B)$. When the algorithm terminates, $\langle P, Rel \rangle$ equals $\langle P_{sim}, Rel_{sim} \rangle$.

The pair $\langle P, Rel \rangle$ is refined by splitting the blocks of the partition in P and pruning the relation Rel. For this purpose, the algorithm maintains a set $Remove_a(B)$ for each $a \in \Sigma$ and $B \in P$. $Remove_a(B)$ contains states that was recently identified as not having an a-transition leading into $\bigcup Rel(B)$. Clearly, a state in $Remove_a(B)$ cannot simulate states that have an a-transition going into B. Therefore, for a set $Remove_a(B) \neq \emptyset$ chosen at the beginning of an iteration, the algorithm splits each block $C \in P$ to $C \cap Remove_a(B)$ and $C \setminus Remove_a(B)$ (states not capable and states possibly capable of simulating states from $pre_a(B)$). 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, the algorithm refines the values of Rel and *Remove* to be consistent with the new value of the partition P refined by the *Split*. All Rel relations between the original "parent" blocks of states are inherited to their "children" blocks into which the parents were split (line 8)—the notation $\mathsf{parent}_{P_{\mathsf{prev}}}(C)$ refers to the parent block of which C was a part before the *Split*. On line 10, the *Remove* sets are inherited from parent blocks to their children. In the second step, the algorithm performs the actual refinement of the relation induced by $\langle P, Rel \rangle$. On line 14, Rel is being pruned to reflect that states that have an a-transition going into B cannot be simulated by states which do not have an a-transition going into $\bigcup Rel(B)$. This is done by removing the relation between blocks included in $Remove_a(B)$ and blocks with states leading to B via a. Refinement of Rel is then propagated further to *Remove* sets. Removing a pair of blocks (C, D) from Rel may cause that a state that has a b-transition into D (therefore, it had a b-transition into $\bigcup Rel(C)$ before removing (C, D) from Rel) now does not have any b-transition into $\bigcup Rel(C)$. Such a state is freshly identified as not being capable of simulating states from $pre_a(C)$. We add it into $Remove_b(C)$ on line 17, which ensures propagation of the negative information.

Algorithm 1: Computing simulation on an LTS

```
Input: An LTS $\mathcal{T} = (Q, \Sigma, \delta)$, the coarsest partition-relation pair
       $\langle P_I, Rel_I \rangle$ on $Q$ inducing a preorder $I \subseteq Q \times Q$.
Data: A partition-relation pair $\langle P, Rel \rangle$ on $Q$, and for each $B \in P$ and
      $a \in \Sigma$, a set $Remove_a(B) \subseteq Q$.
Output: The coarsest partition-relation pair $\langle P_{sim}, Rel_{sim} \rangle$ inducing $\preccurlyeq^I$.
/* initialisation */
$\langle P, Rel \rangle \leftarrow \langle P_I, Rel_I \rangle$;
forall $a \in \Sigma, B \in P$ do  $Remove_a(B) \leftarrow Q \setminus pre_a(\bigcup Rel(B))$;
/* computation */
while $\exists a \in \Sigma.\ \exists B \in P.\ Remove_a(B) \neq \emptyset$ do
    $Remove \leftarrow Remove_a(B); Remove_a(B) \leftarrow \emptyset$;
    $P_{\mathsf{prev}} \leftarrow P; B_{\mathsf{prev}} \leftarrow B; Rel_{\mathsf{prev}} \leftarrow Rel$;
    $P \leftarrow Split(P, Remove)$;
    forall $C \in P$ do
        $Rel(C) \leftarrow \{D \in P \mid D \subseteq \bigcup Rel_{\mathsf{prev}}(\mathsf{parent}_{P_{\mathsf{prev}}}(C))\}$;
        forall $b \in \Sigma$ do
            $Remove_b(C) \leftarrow Remove_b(\mathsf{parent}_{P_{\mathsf{prev}}}(C))$
    forall $C \in P.\ C \xrightarrow{a} B_{\mathsf{prev}}$ do
        forall $D \in P.\ D \subseteq Remove$ do
            if $(C, D) \in Rel$ then
                $Rel \leftarrow Rel \setminus \{(C, D)\}$;
                forall $b \in \Sigma$ do
                    forall $r \in pre_b(D)$ such that $r \notin pre_b(\bigcup Rel(C))$ do
                        $Remove_b(C) \leftarrow Remove_b(C) \cup \{r\}$
return $\langle P, Rel \rangle$;
```

3.2.1 Correctness of the Algorithm

The correctness of the algorithm is formalised in Theorem 1. A similar correctness result is proved in [RT07] for the algorithm on Kripke structures, using notions from the theory of abstract interpretation. We provide here an alternative, more direct proof.

We will prove termination and partial correctness, this is, that (1) the final partition-relation pair that we denote $\langle P_{fin}, Rel_{fin} \rangle$ induces $\preccurlyeq^I$; and (2) that $\langle P_{fin}, Rel_{fin} \rangle$ is also the coarsest. The two points together give $\langle P_{fin}, Rel_{fin} \rangle = \langle P_{sim}, Rel_{sim} \rangle$.

Theorem 1. *Algorithm 1 terminates and returns the partition-relation pair* $\langle P_{sim}, Rel_{sim} \rangle$.

Let us first introduce some notation that will be needed within the proof of the theorem. 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 referred 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 *descendant* 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. We will denote by $q \overset{a}{\nrightarrow} r$ the fact that $\neg(q \xrightarrow{a} r)$. Moreover, for any $B, C \subseteq Q$, $q \overset{a}{\nrightarrow} C$ and $B \overset{a}{\nrightarrow} C$ are defined analogously, i.e. provided that $q \notin \mathit{pre}_a(C)$ and $B \cap \mathit{pre}_a(C) = \emptyset$. We will use $R_{\langle P, \mathit{Rel} \rangle}$ to denote the relation induced by the partition-relation pair $\langle P, \mathit{Rel} \rangle$ in a particular state of a run of the algorithm.

Lemma 3.1. *On line 3 of Algorithm 1, the pair $\langle P, \mathit{Rel} \rangle$ is always a partition-relation pair. The partition P can only be refined during the computation. Moreover, the relation $R_{\langle P, \mathit{Rel} \rangle}$ is monotonically getting smaller during the computation.*

Proof. The initial value of $\langle P, \mathit{Rel} \rangle$ is clearly a partition-relation pair. After *Split* on line 6, $\langle P, \mathit{Rel} \rangle$ is temporarily not a partition-relation pair as *Rel* is a relation on P_{prev}, not on P. However, after inheriting all *Rel* links of parent blocks by their children on lines 7–10, $\langle P, \mathit{Rel} \rangle$ is a partition-relation pair again. 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 can only remove elements from *Rel*. ☐

Lemma 3.2. *The following claims are invariants of the while loop (of line 3) of Algorithm 1:*

$$\forall B \in P.\ \forall a \in \Sigma.\ \mathit{Remove}_a(B) \overset{a}{\nrightarrow} \bigcup \mathit{Rel}(B) \tag{3.1}$$

$$\forall B \in P.\ B \in \mathit{Rel}(B) \tag{3.2}$$

$$\begin{aligned}&\forall B, C \in P.\ (B, C) \in \mathit{Rel} \implies \\ &\Big(\forall a \in \Sigma.\ \forall D \in P.\ B \xrightarrow{a} D \implies C \subseteq \mathit{pre}_a(\bigcup \mathit{Rel}(D)) \cup \mathit{Remove}_a(D)\Big)\end{aligned} \tag{3.3}$$

Proof. After the initialisation, all the invariants hold. It is immediate for Invariants 3.1 and 3.2. It is also fairly obvious for Invariant 3.3, as after the algorithm passes line 2, for all $q \in Q, a \in \Sigma, D \in P$, it holds that either q has an a transition leading to $\bigcup \mathit{Rel}(D)$ or q is in $\mathit{Remove}_a(D)$.

- Invariant (3.1) can never be broken. After the initialisation it holds. From there on, it holds because only such a state r can be moved into the $\mathit{Remove}_b(C)$ which is not in $\mathit{pre}_b(\bigcup \mathit{Rel}(C))$ (the test on line 16). Moreover, if r is once not in $\mathit{pre}_b(\bigcup \mathit{Rel}(C))$, then it will never be there from that moment on (by Lemma 3.1).

- Invariant (3.2) can never be broken as violating reflexivity of *Rel* requires choosing a pair (C, D) on line 14 such that $C = D$. The (C, D) pair can be chosen on line 14 only if $C \xrightarrow{a} B$ and $D \subseteq \mathit{Remove}_a(B)$ where B is the pivot block. Thanks to Invariant (3.1), this is not possible for $C = D$.

- Invariant (3.3) 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 $a \in \Sigma$, $Remove_a(C) = Remove_a(C')$. Thus, after finishing the for loop on line 7, Invariant (3.3) can be broken only for those (B, C) pairs such that it was broken even for their parents on line 6. Therefore, if the invariant holds on line 3, then it also holds after returning from the for loop on line 7.

  line 4: Assume the invariant holds at the beginning of some iteration and is then violated by emptying the $Remove_a(B)$ set on line 4. Then, there are $C, D \in P$ which break the invariant and for which it holds 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 $D_2 \subseteq Remove$. After that, *Rel* and the *Remove* sets are inherited on lines 7–10. Now only those (C', D_2) pairs break the invariant where C' is a child of C such that it leads via a into a child of B. But exactly these pairs will be chosen on line 13 within this iteration for pruning *Rel*. Hence, after finishing the iteration, the invariant will not be violated from the reason of emptying $Remove_a(B)$.

  line 16: Pruning *Rel* on line 14 lead to breaking the invariant as there may 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 $r \overset{b}{\nrightarrow} \bigcup Rel(C)$. However, exactly these r states are moved into $Remove_b(C)$, and so Invariant (3.3) is restored after finishing the for loop on line 13.

□

Lemma 3.3. *If all the Remove sets are empty when evaluating the condition on line 3, then $R_{\langle P,Rel\rangle}$ is a simulation on $\mathcal{T}$ included in I.*

Proof. By Lemma 3.1, it is clear that $R_{\langle P,Rel\rangle}$ is always a subset of I. We have to show that $R_{\langle P,Rel\rangle}$ is also a simulation on $\mathcal{T}$. Let $q\ R_{\langle P,Rel\rangle}\ r$ for some $q \in B, r \in C$ where $B, C \in P$. From the definition of $R_{\langle P,Rel\rangle}$, $(B, C) \in Rel$. Let $q \xrightarrow{a} s$ for some $s \in D, D \in P$. Then $B \xrightarrow{a} D$. Therefore, by Invariant (3.3) and since all the *Remove* sets are empty, we get $C \subseteq pre_a(\bigcup Rel(D))$. This means that there is $u \in \bigcup Rel(D)$ such that $r \xrightarrow{a} u$. By the definition of $R_{\langle P,Rel\rangle}$, we have $s\ R_{\langle P,Rel\rangle}\ u$. Therefore, $R_{\langle P,Rel\rangle}$ is a simulation on $\mathcal{T}$ and the lemma holds. □

During the computation, the relation *Rel* is not necessarily always transitive. We can prove only the following property of *Rel* that roughly resembles transitivity, and which is crucial for our correctness proof.

Lemma 3.4. *Under the assumption that $\preccurlyeq^I \subseteq R_{\langle P,Rel\rangle}$, the following invariant always holds on line 3 of Algorithm 1: For any $q, r \in Q$ with $q \preccurlyeq^I r$ and*

$B, C, D \in P$ such that $q \in C$, $r \in D$ and $(B, C) \in Rel$, it holds that also $(B, D) \in Rel$.

Proof. Let us recall the relationship between a partition-relation pair $\langle P, Rel\rangle$ and its induced relation $R_{\langle P,Rel\rangle}$ which is: For any $B, C \in P$ and $q \in B, r \in C$, it holds that $q\ R_{\langle P,Rel\rangle}\ r$ iff $(B, C) \in Rel$. Therefore, if $R_{\langle P,Rel\rangle} \subseteq \preccurlyeq^I$, 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 initialisation, the claim holds as Rel_I is transitive (the relation I is a preorder). We prove the induction step by contradiction.

Let the lemma be broken for the first time at the beginning of the i-th iteration of the while loop. We use $Start_i$ to denote the state of the algorithm at this moment. At $Start_i$, we have that $\preccurlyeq^I \subseteq R_{\langle P,Rel\rangle}$ and there are some $B, C, D \in P$, $q \in C$, and $r \in D$ such that $q \preccurlyeq^I r$, $(B, C) \in Rel$, and $(B, D) \notin Rel$. From $q \preccurlyeq^I r$ and $\preccurlyeq^I \subseteq R_{\langle P,Rel\rangle}$, we have $(C, D) \in Rel$. Because the induced relation is shrinking only (Lemma 3.1), we have that at each moment of the computation preceding $Start_i$, the relation $\preccurlyeq^I$ was a subset of the relation induced by the current partition-relation pair, the ancestor C' of C was above the ancestor B' of B wrt. the current Rel, and also the ancestor of D was above the ancestor of C. Because of this and as the lemma is broken for the first time at $Start_i$, we know that at the beginning of any iteration prior to the i-th one, the ancestor of D was above the ancestor of B wrt. the current state of Rel.

Let us analyse the moment before $Start_i$ when (B, D) is going to be removed from relation this is, we are within the $i-1$-th, just before entering the for loop on line 11). Let $\langle P, Rel'\rangle$ be the current partition-relation pair (the current partition P at that moment is the same as at $Start_i$, since no splitting will be done until $Start_i$). 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 keep (B, C) and (C, D) in Rel' during this iteration as these two pairs will be in Rel at $Start_i$. Removing (B, D) from Rel' is caused by processing the $Remove_a(E)$ set where $E \in P_{\mathsf{prev}}$ is the pivot of the $i-1$-th iteration. Thus, we have that $B \xrightarrow{a} E$, $D \subseteq Remove_a(E)$ and $C \cap Remove_a(E) = \emptyset$.

Let us examine the state of the algorithm at the beginning of the $i-1$-th iteration, the moment referred to as $Start_{i-1}$. The current partition relation pair at $Start_{i-1}$ is $\langle P_{\mathsf{prev}}, Rel'_{\mathsf{prev}}\rangle$. It holds that $\preccurlyeq^I \subseteq R_{\langle P_{\mathsf{prev}},Rel'_{\mathsf{prev}}\rangle}$. 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 (3.3), 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, there is $F \in Rel'_{\mathsf{prev}}(E)$ and $q' \in F$ with $q \xrightarrow{a} q'$. Since $q \preccurlyeq^I r$, there is $r' \in Q$ with $r \xrightarrow{a} r'$ and $q' \preccurlyeq^I r'$. Because $\preccurlyeq^I \subseteq R_{\langle P_{\mathsf{prev}},Rel_{\mathsf{prev}}\rangle}$, the block $G \in P_{\mathsf{prev}}$ containing r' must be in $Rel'_{\mathsf{prev}}(F)$. Finally, because $r \in D \subseteq Remove_b(E)$, from Invariant (3.1), we get $(E, G) \notin Rel'_{\mathsf{prev}}$.

To conclude the proof, observe that the states q', r', the blocks $E, F, G \in P_{\mathsf{prev}}$, and the partition-relation pair $\langle P_{\mathsf{prev}}, Rel'_{\mathsf{prev}}\rangle$ form a situation that violates the lemma at $Start_{i-1}$ (to recap, we have that $\preccurlyeq^I \subseteq R_{\langle P_{\mathsf{prev}},Rel'_{\mathsf{prev}}\rangle}$, $q' \in F, r' \in$

$G, q' \preccurlyeq^I r'$, and $(E, F) \in \mathit{Rel}'_{\mathsf{prev}}$, but $(E, G) \notin \mathit{Rel}'_{\mathsf{prev}}$). This is a contradiction since Start_i was supposed to be the first such a moment. □

Lemma 3.5. *At any point of a run of Algorithm 1,* $\preccurlyeq^I \subseteq R_{\langle P,\mathit{Rel}\rangle}$.

Proof. The lemma apparently holds after initialisation. We will prove that it always holds by contradiction—we will show that violating this lemma in a run of Algorithm 1 has to be preceded by breaking Lemma 3.4.

Let us choose the moment just before the lemma is violated for the first time. This is, some (B, C) is going to be removed from Rel on line 14 such that there are $q \in B$ and $r \in C$ with $q \preccurlyeq^I r$. This update of Rel is caused by processing the set $\mathit{Remove}_a(D)$ where $D \in P_{\mathsf{prev}}$ is the pivot of the current iteration of the while loop, $B \xrightarrow{a} D$, $B \cap \mathit{Remove} = \emptyset$ (Remove is the recorder value of $\mathit{Remove}_a(D)$ which was emptied on line 4 in this iteration), and $C \subseteq \mathit{Remove}$. Let $B', C' \in P_{\mathsf{prev}}$ be the ancestors of B, C. From Invariant (3.2), we have that $(B', B') \in \mathit{Rel}_{\mathsf{prev}}$.

Let us examine the state at the beginning of this iteration. We have that $B' \xrightarrow{a} D$ because of $B \xrightarrow{a} D$, which by Invariant (3.3) gives $B' \subseteq \mathit{pre}_a(\bigcup \mathit{Rel}_{\mathsf{prev}}(D)) \cup \mathit{Remove}_a(D)$. Since $q \in B$, $q \notin \mathit{Remove}_a(D)$, and therefore there are $E \in \mathit{Rel}_{\mathsf{prev}}(D)$ and $q' \in E$ with $q \xrightarrow{a} q'$. 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, it holds so far), we have that there are $F \in \mathit{Rel}_{\mathsf{prev}}(E)$ and $r' \in F$ with $r \xrightarrow{a} r'$. However, as $r \in \mathit{Remove}_a(D)$ and because of Invariant (3.1), we have $(D, F) \notin \mathit{Rel}_{\mathsf{prev}}$. Hence the states q', r' and the blocks D, E, F violates Lemma 3.4 at the beginning of this iteration. □

Lemma 3.6. *At any point of a run of Algorithm 1, any two states* $q, r \in Q$ *with* $q \cong^I r$ *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 3.4.

After the initialisation the lemma holds. Let us choose the first moment when it is broken. At that moment, states q, r with $q \cong^I r$ are separated from each other by the *Split* operation during processing of some pivot block B. Without loss of generality, we assume that at the beginning of this iteration $r \in \mathit{Remove}_a(B)$ and $q \notin \mathit{Remove}_a(B)$.

Consider now the moment within some of the preceding iterations, just before entering the for loop on line 11 during which r will be added into $\mathit{Remove}_a(B')$ where B' is an ancestor of B. Let the current partition-relation pair be $\langle P, \mathit{Rel}\rangle$, and let $q, r \in C, C \in P$. There is some block $D \in \mathit{Rel}(B')$ with $r \xrightarrow{a} D$ such that (B', D) will be removed from Rel and r will be added to $\mathit{Remove}_a(B')$ because of that within this iteration.

Since sine $r \xrightarrow{a} D$, there is $r' \in D$ with $r \xrightarrow{a} r'$. From $r \preccurlyeq^I q$, there is $q' \in Q$ with $q \xrightarrow{a} q'$ and $r' \preccurlyeq^I q'$, and since $\preccurlyeq^I \subseteq \langle P, \mathit{Rel}\rangle$ (Lemma 3.5), there is $E \in \mathit{Rel}(D)$ with $q' \in E$. Moreover, from Lemma 3.4 (whose claim holds also just before entering the for loop on line 11 because lines 4–10 do not influence the induced relation), $E \in \mathit{Rel}(B')$.

We have shown that when entering the for loop on line 11, $q \xrightarrow{a} \bigcup Rel(B')$. Recall that q will not be added into $Remove_a(B')$ during this iteration. Therefore, it has to hold that $q \xrightarrow{a} \bigcup Rel(B')$ also after finishing the for loop on line 11 (otherwise q would be added into $Remove_a(B')$). This is, after finishing the for loop on line 11, there is still some $E' \in Rel(B')$ and $q'' \in E'$ with $q \xrightarrow{a} q''$. Because $q \preccurlyeq^I r$, there is some $r'' \in Q$ with $q'' \preccurlyeq^I r''$ and $r \xrightarrow{a} r''$. Since $\preccurlyeq^I \subseteq \langle P, Rel\rangle$, there is some $F \in Rel(E')$ with $r'' \in F$. But at the end of the for loop on line 11 (i.e. the beginning of the next iteration of the while loop), $(B', F) \notin Rel$ as r was be added into $Remove_a(B')$ within the for loop (Invariant (3.1)). To conclude the proof, observe now that at the beginning of the next iteration of the while loop, states q'', r'' and blocks B', E', F form a situation contradicting Lemma 3.4. □

Lemma 3.7. *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 a state q is in $Remove_a(B)$ after the initialisation, then $q \overset{a}{\nrightarrow} \bigcup Rel_I(B)$. If q is added into $Remove_a(B)$ later on line 17, then it means that the test on line 13 passed, so $q \xrightarrow{a} \bigcup Rel(B)$ was true at that moment[1]. Subsequently, after the update of Rel on line 14, $q \overset{a}{\nrightarrow} \bigcup Rel(B)$. From Lemma 3.1, if once $q \overset{a}{\nrightarrow} \bigcup Rel(B)$, then from that moment on it can never happen that $q \xrightarrow{a} \bigcup Rel(B')$ where B' is a descendant of B. It means that q will never be added to any $Remove_a(B')$ where B' is a descendant of B. To summarise: when a pivot B with nonempty $Remove_a(B)$ is chosen to be processed on line 3, $Remove_a(B)$ is always emptied and none of the states from $Remove_a(B)$ can be added to any $Remove_a(B')$ where B' is a descendant of B again. Thus whenever later some descendant B' of B with $Remove_a(B')$ is being processed, $Remove_a(B) \cap Remove_a(B') = \emptyset$. □

We are now ready to prove Theorem 1.

Proof of Theorem 1. Due to Lemma 3.7, 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 3.3 implies that the relation $R_{\langle P_{fin}, Rel_{fin}\rangle}$ induced by the final partition-relation pair $\langle P_{fin}, Rel_{fin}\rangle$ is a simulation included in I. Lemma 3.5 implies that this simulation is the maximal one. Finally, Lemma 3.6 implies that the resulting partition P_{fin} equals $Q/\cong^I$ and thus $\langle P_{fin}, Rel_{fin}\rangle = \langle P_{sim}, Rel_{sim}\rangle$. □

3.2.2 Implementation and Complexity of the Algorithm

The complexity of the algorithm is equal to that of the original algorithm from [RT07], up to the new factor Σ that is not present in [RT07] (or, equivalently, $|\Sigma| = 1$ in [RT07]). The complexity analysis is based on the similar reasoning

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

as the one in [RT07]. Time complexity strongly depends on use of certain data structures and on some particular implementation techniques that we describe below along the analysis within the proof of Theorem 2.

Theorem 2. *Algorithm 1 runs in time* $\mathcal{O}(|\Sigma||P_{sim}||Q| + |P_{sim}||\delta|)$ *and space* $\mathcal{O}(|\Sigma||P_{sim}||Q|)$.

Proof.

Basic Data Structures. 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.

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 $q \in Q$ contains a list of nonempty $pre_a(q)$ sets, each of them encoded as a list of its members (we use a list rather than an array having an entry for each $a \in \Sigma$ in order to avoid a need to iterate over alphabet symbols for which there is no transition). The partition P 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 is encoded as a Boolean matrix $P \times P$.

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 line 3 then means just scanning the head of the list of blocks.

For each $a \in \Sigma$, a state $q \in Q$ and a block $B \in P$, we maintain a counter $Count_a(q, B)$. Its value within a run of the algorithm records cardinality of the set $\{r \in Q \mid r \in \bigcup Rel_a(B) \wedge q \xrightarrow{a} r\}$. This counters allow us to test whether r is in $pre_b(\bigcup Rel(C))$ on line 16 in constant time—we just ask whether $Count_b(r, C) = 0$. The counters are stored as an $P \times Q$ integer matrix per each $a \in \Sigma$. The way of updating the counters during a computation will be described later.

We attach to each $q \in Q$ an array indexed by symbols of Σ. A cell of the array indexed by $a \in \Sigma$ contains a reference the $pre_a(q)$ list. Using the arrays, we can access the $pre_a(q)$ list for given a and q in constant time (it would be $\mathcal{O}(|\Sigma|)$ time without the arrays).

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

Time Complexity. We first introduce some auxiliary notation. For $B \subseteq Q$ and $a \in \Sigma$, we denote by $in_a(B)$ the set $\{(r, a, q) \in \delta \mid q \in B\}$, 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 δ_a the set of all a-edges of δ. We use $Anc(B)$ to denote the set of all ancestors of B, including also B itself.

We first analyse the initialisation phase of the algorithm preceding the main while loop. The initialisation of the arrays of pointers to the pre_a lists takes

$\mathcal{O}(|\Sigma||Q|)$ time. The *Count* counters are initialised by (1) setting all *Count* 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 $Count_a(r, C)$. This takes $\mathcal{O}(|P_I||\delta|)$ time. The *Remove* sets are initialised by iterating through all $a \in \Sigma, q \in Q, B \in P$ and checking whether $Count_a(q, B) = 0$. If so, then we add (append) q to $Remove_a(B)$. This takes $\mathcal{O}(|\Sigma||P_I||Q|)$ time. Overall, the initialisation can be done in time $\mathcal{O}(|P_I||\delta| + |\Sigma||P_I||Q|)$.

The time complexity analysis of the while loop builds on Lemma 3.7 and Lemma 3.1 proved within the proof of correctness of Algorithm 1. The two lemmas allow us to make the following observations:

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 $|Q|$.

Observation 2. If a pair (C, D) once appears on line 15, then no pair (C', D') such that $C \in Anc(C')$ and $D \in Anc(D')$ can appear on line 15 again.

The $Split(P, Remove)$ operation can be implemented in the following way: 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. This can be done in time $\mathcal{O}(|Remove|)$. 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 with $B' \in Anc(B)$ according to which *Split* is being done is below $|Q|$. Therefore, summed over all symbols of Σ and all blocks of P_{sim}, the overall time complexity of all *Split* operations is $\mathcal{O}(|\Sigma||P_{sim}||Q|)$.

The time complexity analysis of lines 7–10 is based on the fact that it can happen at most $|P_I| - |P_{sim}|$ 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 $a \in \Sigma, B \in P$, $Remove_a(B) \subseteq Q$, and a new block will be generated at most $|P_I| - |P_{sim}|$ times, the overall time of this copying is in $\mathcal{O}(|\Sigma||P_{sim}||Q|)$. (2) Add a row and a column to the *Rel* matrix and copy the entries from those of the parent. This operation takes $\mathcal{O}(|P_{sim}|)$ 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 $\mathcal{O}(|P_{sim}|^2)$. (3) Add and copy the *Count* counters. For one newly generated block, this operation takes an $\mathcal{O}(|\Sigma||Q|)$ time and thus for all generated blocks, it gives time $\mathcal{O}(|\Sigma||P_{sim}||Q|)$.

Lines 13 and 14 are $\mathcal{O}(1)$-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 $\mathcal{O}(|Remove|)$ 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 $\mathcal{O}(1)$ 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 $\mathcal{O}(|in_a(B)|)$ 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 $\mathcal{O}(\sum_{a\in\Sigma}\sum_{B\in P_{sim}} |RL_a(B)||in_a(B)|)$. From the initial observations, we can see that $|RL_a(B)| \leq |P_{sim}|$, and thus we have the overall time complexity of lines 11–14 in $\mathcal{O}(\sum_{a\in\Sigma}\sum_{B\in P_{sim}} |P_{sim}||in_a(B)|) = \mathcal{O}(\sum_{a\in\Sigma} |P_{sim}||\delta_a|) = \mathcal{O}(|P_{sim}||\delta|)$.

Lines 15–17 are implemented as follows. For a single pair (C, D) 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 $Count_a(r, C)$. If $Count_a(r, C) = 0$ after the decrement, we append r to 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 a pair (C', D) such that $q \in D$ and $C' \in Anc(C)$ is being removed from $Rel.$ on line 14. Thus, the contribution of the pair C, q to the time complexity of lines 15–17 is $\mathcal{O}(\sum_{a\in\Sigma} |pre_a(q)|)$. Therefore, the contribution of the C, r pairs for all $r \in Q$ is $\mathcal{O}(|\delta|)$, and hence the overall time complexity of lines 15–17 is $\mathcal{O}(|P_{sim}||\delta|)$.

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

3.3 Conclusions and Future Work

We have presented a modification of the currently fastest algorithm RT [RT07] for computing simulations over Kripke structures, which was at the time of its publication the fastest algorithm for computing simulations over LTS (the currently fastest algorithm is its optimised version from [HŠ09a]). The algorithm has the time complexity $\mathcal{O}(|\Sigma||P_{sim}||Q| + |P_{sim}||\delta|)$ and the space complexity $\mathcal{O}(|\Sigma||P_{sim}||Q|)$, which is slightly worse than $\mathcal{O}(|P_{sim}||\delta|)$ time and $\mathcal{O}(|P_{sim}||Q|)$ space of RT. However, this complexity increase can be to a large degree lowered as we show in [HŠ09a]. We have also presented a proof of correctness that is more straightforward than the one presented in [RT07].

We plan to continue the research by the authors of [RT07] and [CRT09]. We have noticed that the algorithm from [CRT09] that refines RT goes in some sense against the spirit of the original algorithm from [HHK95], which is the main reason of its worse time complexity. We believe that this problem can be circumvented and that it is possible to design an algorithm that matches both the time complexity of the fastest simulation algorithm [RT07] and space complexity of the most space efficient algorithm [CRT09].

4 Simulation-based Reduction of Tree Automata

Tree automata (TA) appear in many areas of computer science such as verification, structured documents processing, natural language processing, and decision procedures of various logics, where they are used for modelling and reasoning about structured objects such as configurations of complex systems, algebraic term representations of data, computations, syntactical trees, XML documents, etc.—see, e.g., [CDG$^+$07].

In many applications of tree automata, it is highly desirable to deal with automata which are as small as possible in order to save memory as well as time. In theory, one can always determinise and minimise any given (bottom-up) tree automaton. However, the determinisation step may lead to an exponential blow-up in the size of the TA and even if the minimal deterministic TA is small, it might not be feasible to compute it in practice because of the expensive determinisation step. Moreover, the minimal deterministic TA may still be bigger than the original non-deterministic TA.

To avoid determinisation, a TA can be reduced (while preserving its language) by identifying and collapsing states that are equal wrt. a suitable equivalence relation. In the case of finite word automata, this method works well with relations that respect language inclusion on states of an automaton (relations $\preceq$ such that $q \preceq r$ implies that the language accepted at the state q is a subset of the language accepted at the state r). The well-known candidates are language inclusion itself, simulation, and bisimulation. Here, a natural trade-off between the computational cost and reduction power arises. Computing language inclusion is PSPACE-complete for FA and even EXPTIME-complete for TA, which is too costly. Bisimulation can be computed very efficiently in time $\mathcal{O}(m \log(n))$ (see [Hop71, PT87, Val09]) where n is the number of states and m the number of transition of the automaton. However, bisimulation relations are usually quite sparse and thus only of a limited reduction capability. Simulation seems to be a good compromise between the above two. It can be efficiently checked in polynomial time $\mathcal{O}(mn)$ (see [GPP03, HHK95, RT07, CRT09]) and often approximates language inclusion relatively well which gives it a good reduction power.

In this chapter, we start by considering a basic notion of tree simulation, called *downward simulation* [ALdR05], corresponding to a natural extension of the usual notion of simulation defined on word automata to automata on ordered trees. This relation can be shown to be compatible with the tree language equivalence.

The second notion of simulation that we consider, called *upward simulation* [ALdR05], corresponds intuitively to a generalisation of the notion of backward simulation to tree automata. 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, the so called *mediated equivalence*, 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. Mediated equivalence is coarser than (or, in the worst case, as coarse as) the downward simulation equivalence and according to our practical experiments, it usually leads to significantly better reductions of the automata.

In this way, we obtain three candidates for simulation-based equivalences for use in automata reduction: mediated equivalence, downward simulation, and upward simulation (Upward simulation may be used alone only when parametrised by the identity. If the inducing relation is nontrivial, it may be used only to form a mediated equivalence.). 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 [RT07], considers simulation relations defined on Kripke structures. The use of this algorithm requires its adaptation to labelled transition systems, which we provided in Chapter 3. 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 this chapter.

We continue the study by looking also at tree automata bisimulations that are in fact special cases of tree automata simulations. Tree automata bisimulations were studied in [HMM07a, AHK07]. They are computationally cheaper than simulations, but they are usually quite sparse and thus have only a limited reduction power. However, it is possible to construct a mediated equivalence by combining any downward relation (simulation, bisimulation, or identity) with any upward relation (simulation, bisimulation, identity). This results in a scale of mediated equivalences suitable for reducing automata that offers a fine choice between reduction power and computation cost. Moreover, the tree automata bisimulation can be computed using the same LTS translations as we have defined for tree automata simulations. The only difference is that after translating a tree automaton into an LTS, we run on it an LTS bisimulation algorithm instead of a simulation one. The algorithms for computing TA bisimulation obtained this way are competitive with the specialised ones presented at [HMM07a].

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 (bi-)simulation reduction framework presented in this chapter really offers a fine choice of relations suitable for reduction tree automata with different reduction powers and computation costs. Especially the coarsest mediated equivalence that arise by combining downward simulation with upward simulation gives much better reduction than the bisimulation-based methods from [AHK07, HMM07a].

Related work. As far as we know, our work [ABH$^+$08c] 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 [ALdR06] 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 [ALdR06]. 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 [AHK07, HMM07a]. The notion of a backward bisimulation used in [HMM07a] corresponds to what can be called a downward bisimulation in our terminology, while backward simulation from [HMM07a] corresponds to the most restrictive variant of our upward simulation when the inducing relation is the identity. The specialised algorithms for computing the bisimulations from [HMM07a] have comparable complexities to our ones.

Outline. In Section 4.1, we give definitions of tree automata downward and upward simulations and state their basic properties. We then discuss tree automata bisimulations viewing them as certain special cases as the tree automata simulations. In Section 4.2, we introduce the principle of combining downward and upward simulations to relations suitable for quotienting tree automata. Then, in Section 4.3, we discuss properties of variants of the combined relations. Section 4.4 presents algorithms for computing all the proposed relations, Section 4.6 describes our experimental results, and Section 4.6 finally concludes the chapter.

4.1 Tree Automata Simulations and Bisimulations

We now present definitions of downward simulation and upward simulation *parametrised* or *induced* by a downward simulation. Subsequently, we present downward and induced upward bisimulation that are important special cases of the simulations. We fix a tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$ for the rest of this section.

4.1.1 Downward and Upward Simulation

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

Lemma 4.1. *The set of all downward simulations on $\mathcal{A}$ is closed under reflexive and transitive closure and under union.*

Proof. Union: Given two downward simulations D_1 and D_2, we want to prove that $D = D_1 \cup D_2$ is also a downward simulation. Let qDr for some $q, r \in Q$, then either qD_1r or qD_2r. Assume without loss of generality that qD_1r. Then, from the definition of downward simulations, whenever $(q_1, \ldots, q_n) \xrightarrow{a} q$, then there is a rule $(r_1, \ldots, r_n) \xrightarrow{a} r$ with $q_i D_1 r_i$ for all $i : 1 \leq i \leq n$. As $D_1 \subseteq D$ gives $q_i D r_i$ for all the positions i, D fulfils the definition of a downward simulation.

Reflexive closure: It can be seen from the definition of downward simulations that the identity is a downward simulation. Thus, the union of the identity and any downward simulation is a downward simulation.

Transitive closure: Let D be a downward simulation and let D_T be its transitive closure. Let qD_Tr and $(q_1^1, \ldots, q_n^1) \xrightarrow{a} q$. From qD_Tr, we have that there are states $q = q^1, \ldots, q^m = r$ such that $q^1 D q^2 D \ldots D q^m$. Therefore, from the definition of downward simulations, there are also rules $(q_1^1, \ldots, q_n^1) \xrightarrow{a} q^1, \ldots, (q_1^m, \ldots, q_n^m) \xrightarrow{a} q^m$ with $q^1 D \ldots D q^m$, and $q_i^1 D \ldots D q_i^m$ for all $i : 1 \leq i \leq n$. Thus, as D_T is the transitive closure of D, we obtain $q_i^1 D_T q_i^m$ for all $i : 1 \leq i \leq n$. We have proved that D_T fulfils the definition of downward simulations. $\square$

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

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

Notice that for any two preorders D_1 and D_2 with $D_1 \subseteq D_2$, an upward simulation induced by D_1 is also an upward simulation induced by D_2.

Lemma 4.2. *Given a preorder D on Q, the set of all upward simulations induced by D is closed under reflexive and transitive closure and under union.*

Proof. Union: Given two upward simulations U_1 and U_2 induced by D, we want to prove that $U = U_1 \cup U_2$ is also an upward simulation induced by D. Let qUr for some $q, r \in Q$, then either qU_1r or qU_2r. Assume without loss of generality that qU_1r. Then, from the definition of upward simulations, whenever $(q_1, \ldots, q_n) \xrightarrow{a} q'$ with $q_i = q$, then there is a rule $(r_1, \ldots, r_n) \xrightarrow{a} r'$ with $q'U_1r'$, $q' \in F \implies r' \in F$, and q_jDr_j for all $j : 1 \leq j \neq i \leq n$. As $U_1 \subseteq U$ gives $q'Ur'$, U fulfils the definition of upward simulations induced by D.

Reflexive closure: It can be seen from the definition that the identity is an upward simulation induced by D for any reflexive downward simulation D. Therefore, from the closure under union, the union of the identity and any upward simulation induced by D is an upward simulation induced by D.

Transitive closure: Let U be an upward simulation induced by D and let U_T be its transitive closure. Let $q^1U_Tq^m$ and $(q_1^1, \ldots, q_n^1) \xrightarrow{a} r^1$ with $q^1 = q_i^1$. From $q^1U_Tq^m$, we have that there are states $q^1, \ldots, q^m$ such that $q^1Uq^2U \ldots Uq^m$. Therefore, there are also rules $(q_1^1, \ldots, q_n^1) \xrightarrow{a} r^1, \ldots, (q_1^m, \ldots, q_n^m) \xrightarrow{a} r^m$ with $q_i^1 = q_i^1, \ldots, q_i^m = q^m$, $r^1U \ldots Ur^m$, $r^1 \in F \implies \cdots \implies r^m \in F$, and $q_j^1D \ldots Dq_j^m$ for all $j : 1 \leq j \neq i \leq n$. Thus, from the definition of U_T, we have $r^1U_Tr^m$, from the transitivity of $\implies$, we have $r^1 \in F \implies r^m \in F$, and from the transitivity of D, we have $q_j^1Dq_j^m$ for all $j : 1 \leq j \neq i \leq n$. We have thus proved that U_T fulfils the definition of an upward simulation induced by D. $\square$

Notice that Lemma 4.1 implies that there is a unique maximal downward simulation D on $\mathcal{A}$ which is a preorder. We call it *the downward simulation preorder on* $\mathcal{A}$ and we call the equivalence $D \cap D^{-1}$ *the downward simulation equivalence on* $\mathcal{A}$. Analogically, Lemma 4.2 implies that for a preorder D on Q, there is a unique maximal upward simulation U induced by D which is a preorder. We call it *the upward simulation preorder on* $\mathcal{A}$ *induced by* D and we call the equivalence $U \cap U^{-1}$ *the upward simulation equivalence on* $\mathcal{A}$ *induced by* D.

4.1.2 Downward and Upward Bisimulation

We first recall the well-known notion of bisimulation on labelled transition systems. Given an LTS $\mathcal{T} = (\Sigma, Q, \delta)$, a relation $R \subseteq Q \times Q$ is a *bisimulation* on $\mathcal{T}$ if for any two states $q, r \in Q$, qRr implies that $q \xrightarrow{a} q'$ for some state q' if and only if $r \xrightarrow{a} r'$ for some r' with $q'Rr'$. In other words, a bisimulation on $\mathcal{T}$ is a simulation on $\mathcal{T}$ such that its inverse is also a simulation on $\mathcal{T}$. Tree automata bisimulations are defined in the same spirit, based on the tree automata simulations.

Downward Bisimulation. A *downward bisimulation* D on $\mathcal{A}$ is a binary relation on Q such that if qDr, then $(q_1, \ldots, q_n) \xrightarrow{a} q$ if and only if $(r_1, \ldots, r_n) \xrightarrow{a} r$ with q_iDr_i for each $i : 1 \leq i \leq n$. In other words, a downward bisimulation on $\mathcal{A}$ is any downward simulation on $\mathcal{A}$ such that its inverse is also a downward simulation on $\mathcal{A}$.

Lemma 4.3. *The set of all downward bisimulations on* $\mathcal{A}$ *is closed under symmetric, reflexive, and transitive closure and under union.*

Proof. The closure properties easily follow from Lemma 4.1 and the fact that D is a downward bisimulation on $\mathcal{A}$ if and only if both D and D^{-1} are a downward simulations on $\mathcal{A}$. Union of two bisimulations D_1 and D_2 is a downward bisimulation, because from Lemma 4.1, $D_1 \cup D_2$ is a downward simulation as well as $D_1^{-1} \cup D_2^{-1}$ and because, obviously, $(D_1 \cup D_2)^{-1} = D_1^{-1} \cup D_2^{-1}$. Reflexive closure $D \cup id$ of a downward bisimulation D is a downward bisimulation because the identity id is apparently a downward bisimulation and downward bisimulations are closed under union. Transitive closure D_T of D is a downward bisimulation since by Lemma 4.1, both D_T and the transitive closure $(D^{-1})_T$ of D^{-1} are downward simulations and $D_T^{-1} = (D^{-1})_T$ (transitive closure of the inverse equals inverse of the transitive closure). Finally, the symmetric closure $D \cup D^{-1}$ is a downward bisimulation since both D and D^{-1} are downward bisimulations and downward bisimulations are closed under union. □

Upward Bisimulation. Let D be a preorder on Q. An *upward bisimulation U on $\mathcal{A}$ induced by D* is a binary relation on Q such that if qUr, then

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

In other words, upward bisimulation on $\mathcal{A}$ induced by D is a relation U such that both U and U^{-1} are upward simulations on $\mathcal{A}$ induced by $D \cap D^{-1}$.

We note that the notion of an upward bisimulation induced by the identity relation corresponds to the notion of a forward bisimulation from [HMM07a].

Lemma 4.4. *Given a preorder D on Q, the set of all upward bisimulations induced by D is closed under symmetric, reflexive, and transitive closure and under union.*

Proof. The lemma follows from Lemma 4.2 and from the fact that U is an upward bisimulation on $\mathcal{A}$ induced by D if and only if both U and U^{-1} are upward simulations on $\mathcal{A}$ induced by $D \cap D^{-1}$. The reasoning is the same as in the proof of Lemma 4.3. □

Lemma 4.3 implies that there is a unique maximal downward bisimulation on $\mathcal{A}$ which is an equivalence. We call it *the downward bisimulation equivalence on $\mathcal{A}$*. Analogically, Lemma 4.4 implies that for a given preorder D on Q, there is a unique maximal upward bisimulation induced by D which is an equivalence. We call it *the upward bisimulation equivalence on $\mathcal{A}$ induced by D*.

The fact the tree automata bisimulations are tree automata simulations of some type such that their inverses are also simulations of the type allows us to simplify some further reasoning by handling bisimulations as special cases of simulations.

4.2 Combined Relations for Quotienting

In this section, we will work towards quotienting tree automata using the tree automata simulations. Quotienting a tree automaton according to an equivalence relations means to collapse equivalent states, as formally defined below.

Quotient Tree Automata. Consider a tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$ and an equivalence relation $\equiv$ on Q. We denote $[q]$ the equivalence class of $\equiv$ containing q. The *quotient* of $\mathcal{A}$ according to $\equiv$ is the TA $\mathcal{A}/\equiv = (\Sigma, Q/\equiv, \Delta/\equiv, \{[q] \mid q \in F\})$ where $\Delta/\equiv = \{(([q_1], \ldots, [q_n]), a, [q]) \mid ((q_1, \ldots, q_n), a, q) \in \Delta\}$. Intuitively, we collapse all states which belong to the same block into one state of the quotient automaton, there is a transition in the quotient automaton iff there is a transition between states in the corresponding blocks in the original TA, and a block is accepting iff it contains a state which is accepting.

Obviously, $L(\mathcal{A}) \subseteq L(\mathcal{A}/\equiv)$ and also for any two equivalences $\equiv_1$ and $\equiv_2$ on Q such that $\equiv_1 \subseteq \equiv_2$, $L(\mathcal{A}/\equiv_1) \subseteq L(\mathcal{A}/\equiv_2)$.

Quotienting tree automata w.r.t. any downward simulation equivalence preserves the language, but, surprisingly, this is not the case for upward simulations (except the cases when the inducing preorder is the identity relation, as we show later). However, an upward simulation U can be still used for quotienting in an indirect manner. We can combine it with its inducing downward simulation D into the so called mediated preorder, which is certain fragment of the relation composition $D \circ U^{-1}$ that include the inducing downward simulation D and a part of the inverted upward simulation U (plus some additional elements). The mediated preorder then yields an equivalence which is suitable for quotienting tree automata while preserving the language.

4.2.1 Runs and Simulations

Before we present the mediated preorders, we will state the basic connections between runs of tree automata and the tree automata simulations, which are in our setting the most essential properties of the simulations. For this, we will make use of the notion of *context*. For the rest of this section, let us fix a tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$.

Contexts. Intuitively, a context is a tree with "holes" instead of leaves. Formally, we consider a special symbol $\square \notin \Sigma$ with rank 0. A *context* over Σ is a tree c over $\Sigma \cup \{\square\}$ such that for all leaves $p \in c$, we have $c(p) = \square$. We extend the notion of runs to contexts. Let c be a context with leaves $v_1, \ldots, v_n$ (in the usual lexicographic order). A *run* π of $\mathcal{A}$ on c from $(q_1, \ldots, q_n)$ is defined in a similar manner to a run on a tree except that for each leaf v_i, we have $\pi(v_i) = q_i$, $1 \leq i \leq n$. In other words, each leaf v_i labelled with $\square$ is annotated by q_i. We use $c\,[q_1, \ldots, q_n] \overset{\pi}{\Longrightarrow} q$ to denote that π is a run of $\mathcal{A}$ on c from $(q_1, \ldots, q_n)$ such that $\pi(\epsilon) = q$. The notation $c\,[q_1, \ldots, q_n] \Longrightarrow q$ is explained in a similar manner to runs on trees.

The following lemma intuitively says that backward simulation implies "downward context language" inclusion. This is, the set of contexts accepted *at* a simulation bigger state is a superset of the set of contexts accepted at a downward simulation smaller state. Moreover, runs ending at the simulation bigger state in a sense downward simulate (on the level of leaves) corresponding runs ending at the smaller state.

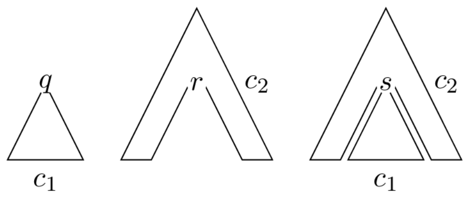
Lemma 4.5. *Let D be a downward simulation on $\mathcal{A}$. If $c[q_1, \ldots, q_n] \Longrightarrow q$ and qDr, then there are states $r_1, \ldots, r_n$ such that for each i with $1 \le i \le n$, $q_i D r_i$, and $c[r_1, \ldots, r_n] \Longrightarrow r$.*

Proof. By induction on the height of c. The base case when c is just a single hole is trivial. For the induction step we consider that c contains more than one node. We know that $c[q_1, \ldots, q_n] \overset{\pi}{\Longrightarrow} q$ for some π. Let $c(\epsilon) = a$. Furthermore, we know that there are $q'_1, \ldots, q'_m$ such that $(q'_1, \ldots, q'_m) \xrightarrow{a} q$, and $\pi(i) = q'_i$ for each i with $1 \le i \le m$. In other words, the run labels the root with q, and labels the children of the root with $q'_1, \ldots, q'_m$, respectively. This means that for all $i : 1 \le i \le m$, $c_i[q_{n_{i-1}+1}, \ldots, q_{n_i}] \overset{\pi_i}{\Longrightarrow} q'_i$ where c_i is the i^{th} subtree of c, π_i is the restriction of π to c_i, $n_0 = 0$, and $n_m = n$. Since qDr, we know that there are $r'_1, \ldots, r'_m$ such that $(r'_1, \ldots, r'_m) \xrightarrow{a} r$ and $q'_i D r'_i$ for each $i, 1 \le i \le m$. By the induction hypothesis, it follows that for each $i, 1 \le i \le m$, there are states $r_{n_{i-1}+1}, \ldots, r_{n_i}$ such that for each $j, n_{i-1} + 1 \le j \le n_i$, $q_j D r_j$ and $c_i[r_{n_{i-1}+1}, \ldots, r_{n_i}] \Longrightarrow r'_i$. Hence $c[r_1, \ldots, r_n] \Longrightarrow r$. $\square$

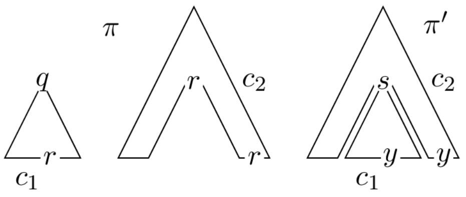
The following lemma is an upward simulation counterpart of Lemma 4.5. Intuitively, it says that upward simulation induced by downward simulations imply "upward context language" inclusion. This is, that the set of contexts accepted *from* an upward simulation bigger state is a superset of the set of contexts accepted from an upward simulation smaller state. Moreover, runs from the simulation bigger state in a sense simulate (downward on the level of leaves and upward on the level of roots) corresponding runs from the simulation smaller state.

Lemma 4.6. *Let U be an upward simulation induced by a downward simulation D. If $c[q_1, \ldots, q_n] \Longrightarrow q$ and $q_i U r_i$ for some $1 \le i \le n$, then there are states $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n, r$ such that $q_j D r_j$ for each $j : 1 \le j \ne i \le n$, qUr, and $c[r_1, \ldots, r_n] \Longrightarrow r$.*

Proof. 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. To simplify the notation, we assume (without loss of generality) that $i = 1$. Suppose that $c[q_1, \ldots, q_n] \overset{\pi}{\Longrightarrow} q$ for some run π and that $q_1 U r_1$. Let p be the parent of the leaf p_1 labelled by q_1 and let $p_1, \ldots, p_m$ be its children. Let c_p be the subtree of c rooted at p. Notice that for all $i, 2 \le i \le m$, $c_i[q_{n_{i-1}+1}, \ldots, q_{n_i}] \overset{\pi}{\Longrightarrow} q'_i$, where c_i is the subtree of c rooted at p_i, $n_1 = 1$ and $n_m = k$ for some $k \le n$. Let $q' = \pi(v)$ and let c' be the context c with the subtrees rooted at $v_1, \ldots, v_m$ deleted. In other words, $dom(c') = dom(c) \setminus \bigcup_{i=1}^{m} \{p_i p' \mid p' \in \mathbb{N}^*\}$, $c'(v') = c(v')$ if $v' \in dom(c')$, and $c'(v) = \square$. Observe



(a) The Mediator



(b) Potential Problems

Figure 4.1: Basic Intuition Behind Mediated Preorder

that $c'[q', q_{k+1}, \dots, q_n] \Longrightarrow q$ and that $(q_1, q'_2, \dots, q'_m) \xrightarrow{a} q'$ for some a. By the definition of the upward simulation and the premise $q_1 U r_1$, it follows that there are $r'_2, \dots, r'_m, r'$ such that $q'_2 D r'_2, \dots, q'_m D r'_m, q' U r'$, and $(r_1, r'_2, \dots, r'_m) \xrightarrow{a} r'$. Since c' is smaller than c, by the induction hypothesis, there are $r_{k+1}, \dots, r_n, r$ such that $q_{k+1} D r_{k+1}, \dots, q_n D r_n$ and $c'[r', r_{k+1}, \dots, r_n] \Longrightarrow r$. For each $i, 2 \leq i \leq m$, we have $q'_i D r'_i$, and thus by Lemma 4.5, there are states $r_{n_{i-1}+1}, \dots, r_{n_i}$ such that for each $j, n_{i-1} + 1 \leq j \leq n_i$, $q'_j D r'_j$ and $c_i[r_{n_{i-1}+1}, \dots, r_{n_i}] \Longrightarrow r'_i$. The claim follows immediately. □

4.2.2 Mediated Preorder

Collapsing states of an automaton wrt. some equivalence allows a run that arrives to some state to *jump* to another equivalent state and continue from there. The equivalence must have the property that the language is not increased even when the jumps are allowed. This is what we aim at when introducing the *mediated equivalence* $\equiv_M$, the largest symmetric fragment of a so called *mediated preorder* M. The mediated preorder M will in particular be defined as a suitable transitive fragment of $D \circ U^{-1}$ in the following, where D is a downward simulation and U an upward simulation induced by D.

The intuition behind allowing a run to jump from a state q to a state r such that $qD \circ U^{-1} r$ is an existence of the so called *mediator*, i.e., a state s such that $qDsU^{-1}r$ (cf. Figure 4.1(a)). By Lemma 4.5, any context c_1 accepted at q can be accepted also at s, and by Lemma 4.6, any context c_2 accepted from r can be accepted from s too (with s appearing at the same position of c_2 as r). Hence, intuitively, the newly allowed run based on the jump from r to q does not add anything to the language because it can anyway be realised through s without jumps.

The relation $D \circ U^{-1}$ is not a preorder, therefore, to build an equivalence for quotienting, we first take some of its transitive fragments. This is natural as if the automaton can safely jump from q_1 to q_2 and from q_2 to q_3, it should be able to safely jump from q_1 to q_3 too.

However, not all of the transitive fragments of $D \circ U^{-1}$ can be used for quotienting due to some phenomena that arise when we allow runs to jump repeatedly (merging two states is equivalent to allowing arbitrary many jumps between them). A sufficient property that guarantees that a preorder can be used for quotienting is downward extensibility—a preorder $M \subseteq D \circ U^{-1}$ is downward extensible if for any three states, it holds that whenever $q_1 M q_2 D q_3$, then $q_1 M q_3$. The intuitive meaning of this requirement is the following. On Figure 4.1(b), we denote π a run on the joined contexts c_1 and c_2 that jumps at some node v from a state q to a state r, and we denote π' a run on the same joined context that labels v by s. It may be the case that such a jump from q to r is in π also done at some node w that appears (i) bellow v or (ii) in the context of v (this my happen if r appears below r or in the context of itself, cf. Figure 4.1(b)). In case (i), the mediated preorder assures that the run π' labels w by some state y that downward simulates r (by Lemma 4.5). Similarly, in the case (ii), the run π' labels w by some state y that downward simulates r (by Lemma 4.6). However, when allowing jumps, then in the case (i), s is not guaranteed to downward simulate q and there may be contexts accepted at q but not accepted at s; and in the case (ii), s is not guaranteed to upward simulate r and there might be contexts accepted from r but not accepted from s. In order to circumvent this problem, we require that if the computation is allowed to jump from q to r, than for any state that can appear in the role of y in our example, this is, for any state y with rDy, the computation is allowed to jump from y to q too. In other words, we require that $qMrDy$ implies qMy, which is our downward extensibility condition.

Combination Operator. We now introduce a *relation combination operator* $\oplus$, which we will use to combine downward and inverted upward simulations into mediated preorders. For the sake of generality, we will define it on arbitrary preorders as follows.

Given two preorders D and U over a set Q, for $x, y \in Q$, $x(D \oplus U)y$ iff (i) $x(D \circ U)y$ and (ii) $\forall z \in Q : yDz \implies x(D \circ U)z$.

The following lemma states properties of the combination operator that show that when using it to combine a downward simulation and an inverted upward simulation, it has all the properties (including downward extensibility) that allow us to use the result as a mediate preorder.

Lemma 4.7. *For any preorders D, U over a set Q, $D \oplus U$ is a preorder and it is a unique maximal preorder satisfying $D \subseteq D \oplus U \subseteq D \circ U$.*

Proof. Let $M = D \oplus U$ and $C = D \circ U$. To make the proof compact, we first prove the following auxiliary claims. For any $x, y, z \in Q$:

1. $xCy \implies xDwUy$ for some $w \in Q$. This follows directly from the definition of C.
2. $xDyCz \implies xCz$. From yCz and (1), we have $yDwUz$ for some $w \in Q$. From $xDyDw$, we have xDw. From $xDwUz$ and from the definition of C, we have xCz.

3. $xMyDz \implies xCz$. This follows directly from the definition of M.

4. $xCyUz \implies xCz$. From xCy and (1), we have $xDwUy$ for some $w \in Q$. From $wUyUz$, we have wUz. From $xDwUz$ and (2), we have xCz.

5. $xMyCz \implies xCz$. From yCz and (1), we have $yDwUz$ for some $w \in Q$. From $xMyDw$ and (3), we have xCw, which together with (1) gives $xDvUw$ for some $v \in Q$. From $vUwUz$, we have vUz and so vCz (as $U \subseteq C$), which together with xDv and (2) gives xCz.

To prove the claim of the lemma, we will first argue that $D \subseteq M \subseteq C$. The second inclusion trivially follows from the definition of $\oplus$. For the first inclusion, we will show that for any $x, y \in Q$ with xDy, xMy holds. As $D \subseteq C$, we have that xCy, which means that Condition (i) from the definition of $\oplus$ is fulfilled. To satisfy Condition (ii), we have to show that for arbitrary $z \in Q$ such yDz, xCz holds. From transitivity of D and from $xDyDz$, we have xDz, which together with $D \subseteq C$ implies that xCz. Thus, since even Condition (ii) is fulfilled, we have xMy, and the first inclusion is proved.

We will now prove that M is a preorder. We first prove by contradiction that M is transitive. Suppose that there exist $x, y, z \in Q$ such that $xMyMz$, but not xMz. Recall that $M \subseteq C$. From (1), we have $xDwUyDvUz$ for some $v, w \in Q$. From $xMyDv$ and (3), we have xCv. From $xCvUz$ and (4), we have xCz. From the definition of $\oplus$, xCz together with not xMz imply that there is some $q \in Q$ such that $xCzDq$, but not xCq. From $yMzDq$ and (3), we get yCq. Then $xMyCq$ and (5) gives xCq, which is a contradiction. We have proved that the relation M is transitive. Showing that M is also reflexive is immediate as we already know that $D \subseteq M$ and that D is reflexive. Thus, we have proved that M is a preorder.

Finally, we will show that M is a unique maximal preorder between D and C, in other words, that any preorder R with $D \subseteq R \subseteq C$ is a subset of M. For the purpose of contradiction assume there are $x, y \in Q$ with xRy and $\neg xMy$. Since $\neg xMy$, there is some z with yDz and $\neg xCz$ (Condition (ii) from the definition of $\oplus$ is not satisfied). The inclusions $D \subseteq R \subseteq C$ together with yD and $\neg xCz$ give yRz and $\neg xRz$. We have $xRyRz$ and $\neg xRz$, which contradicts transitivity of R. $\square$

4.2.3 Quotienting with Mediated Equivalence

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

We will show that quotienting $\mathcal{A}$ with respect to $\equiv_M$ preserves the language. For a state $r \in Q$, a set $B \subseteq Q$ of states, and a relation $R \subseteq Q \times Q$, we write BRr to denote that there is $q \in B$ with qRr.

Lemma 4.8. *For $B_1, \ldots, B_n, B \in Q/{\equiv_M}$ and a context c, if $c[B_1, \ldots, B_n] \Longrightarrow B$, then there exist states $r_1, \ldots, r_n, r \in Q$ with $B_1Dr_1, \ldots, B_nDr_n, BUr$, and $c[r_1, \ldots, r_n] \Longrightarrow r$. Moreover, if $B \cap F \neq \emptyset$, then $r \in F$.*

Proof. The claim is shown by induction on the structure of c. In the base case when 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 then holds obviously by reflexivity of D and U.

For the induction step, we assume that c is not only a single hole. Suppose that $c[B_1, \dots, B_n] \overset{\pi}{\Longrightarrow} B$ for some run π. Let p by some non-leaf node with children $v_k, \dots, v_l, 1 \leq k \leq l \leq n$ that are all leaves (there has to be such a node since the tree $dom(c)$ is finite). Note that $B_k = \pi(v_k), \dots, B_l = \pi(v_l)$. Let $B' = \pi(v)$ and let c' be the context c with the leaves $v_k, \dots, v_l$ deleted. In other words, $dom(c') = dom(c) \setminus \{v_k, \dots, v_l\}$, $c'(v') = c(v')$ provided $v' \in dom(c') \setminus \{v, v_k, \dots, v_l\}$, and $c'(v) = \Box$. Note that $c'[B_1, \dots, B_{k-1}, B', B_{l+1}, \dots, B_n] \Longrightarrow B$. Since c' is smaller than c, we can apply the induction hypothesis and conclude that there are states v, q', and q'_i for each $i \in \{1, \dots, k-1, l+1, \dots, n\}$ with $B_i D q'_i$, $B' D v$, and $B U q'$ such that $c'[q'_1, \dots, q'_{k-1}, v, q'_{l+1}, \dots, q'_n] \Longrightarrow q'$ and, moreover, if $B \cap F \neq \emptyset$, then $q' \in F$. It follows that there are $u \in B'$, $q \in B$ and for each $i \in \{1, \dots, k-1, l+1, \dots, n\}$, $q_i \in B_i$, such that uDv, qUq', and $q_i D q'_i$. By the definition of $\mathcal{A}/{\equiv_M}$, there are states $q_l \in B_l, \dots, q_k \in B_k$, and $z \in B'$ such that $(q_k, \dots, q_l) \xrightarrow{a} z$ for some a. Since $D \subseteq M$ and uDv, we get uMv. Since $u, z \in B'$, it follows that $u \equiv_M z$ and hence zMu. From transitivity of M, we get zMv. From the definition of M, there is a mediator w such that zDw and vUw. By the definition of downward simulation and premises zDw and $(q_k, \dots, q_l) \xrightarrow{a} z$, there are states $r_k, \dots, r_l$ with $q_k D r_k, \dots, q_l D r_l$, and $(r_k, \dots, r_l) \xrightarrow{a} w$. By Lemma 4.6 and premises vUw and $c'[q'_1, \dots, q'_{k-1}, v, q'_{l+1}, \dots, q'_n] \Longrightarrow q'$, there are states r and r_i for each $i \in \{1, \dots, k-1, l+1, \dots, n\}$ such that $q'_i D r_i$, $q' U r$, and and $c'[r_1, \dots, r_{k-1}, w, r_{l+1}, \dots r_n] \Longrightarrow r$. By transitivity of D and U, we get $q_i D r_i$ and qUr. Finally, we know that if $B \cap F \neq \emptyset$, then $q' \in F$ which together with $q' U r$ gives $r \in F$. The claim thus holds. $\Box$

Lemma 4.9. *For a tree t, if $t \Longrightarrow B$, then $t \Longrightarrow w$ for some w with BUw. Moreover, if $B \cap F \neq \emptyset$, then also $w \in F$.*

Proof. Suppose that $t \overset{\pi}{\Longrightarrow} B$ for some π. Let $v_1, \dots, v_n$ be the leafs of t, and let $\pi(v_i) = B_i$ for each $i : 1 \leq i \leq n$. Let c be the context that we get from t by deleting the leaves $v_1, \dots, v_n$. Observe that $c[B_1, \dots, B_n] \overset{\pi}{\Longrightarrow} B$. It follows from Lemma 4.8 that there exist states $r_1, \dots, r_n, r \in Q$ and $q_1 \in B_1, \dots, q_n \in B_n, q \in B$ such that $q_1 D r_1, \dots, q_n D r_n, qUr$, $c[r_1, \dots, r_n] \Longrightarrow r$, and if $B \cap F \neq \emptyset$, then $r \in F$. By the definition of $\mathcal{A}/{\equiv_M}$, it follows that there are $q'_1 \in B_1, \dots, q'_n \in B_n$, and $a_1, \dots, a_n$ such that $\xrightarrow{a_i} q'_i$ for each i such that $1 \leq i \leq n$. We show by induction on i that for each $i : 1 \leq i \leq n$, there are states $u^i_1, \dots, u^i_i, v^i_{i+1}, \dots, v^i_n, w^i$ with $q'_1 D u^i_1, \dots, q'_i D u^i_i, q_{i+1} D v^i_{i+1}, \dots, q_n D v^i_n, r U w^i$, and $c[u^i_1, \dots, u^i_i,\ v^i_{i+1}, \dots, v^i_n] \Longrightarrow w^i$. The base case where $i = 0$ is trivial. We consider the induction step. Since $D \subseteq M$ and $q_{i+1} D v_{i+1}$, we get $q_{i+1} M v_{i+1}$. Since $q_{i+1}, q'_{i+1} \in B_{i+1}$, we have that $q'_{i+1} \equiv_M q_{i+1}$ and hence $q'_{i+1} M q_{i+1}$. By transitivity of M, it follows that $q'_{i+1} M v_{i+1}$. By the definition of M, there is z_{i+1} such that $q'_{i+1} D z_{i+1}$ and $v_{i+1} U z_{i+1}$. By Lemma 4.6, there are $z_1, \dots, z_i, z_{i+2}, \dots, z_n, z$ with $u^i_1 D z_1, \dots, u^i_i D z_i$, $v^i_{i+2} D z_{i+2}, \dots, v^i_n D z_n$, $w^i U z$, and $c[z_1, \dots, z_n] \Longrightarrow z$. By transitivity of D and the premises $q'_j D u^i_j$ and $u^i_j D z_j$, we have $q'_j D z_j$ for each $j : 1 \leq j \leq i$. By transitivity of D and the premises $q_j D v^i_j$ and $v^i_j D z_j$, we have $q_j D z_j$ 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 $c[u^n_1, \dots, u^n_n] \Longrightarrow w^n$. From the definition of the language inclusion preorder and the premises $\xrightarrow{a_i} q'_i$ and $q'_i D u^n_i$, it follows that $\xrightarrow{a_i} u^n_i$ for each $i : 1 \leq i \leq n$. It follows that $t = c[a_1, \dots, a_n] \Longrightarrow w^n$. By the definition of U and the fact that $r \in F$ if $B \cap F \neq \emptyset$, it follows that for all $i : 1 \leq i \leq n$, $w^i \in F$ provided that $B \cap F \neq \emptyset$. Thus, in the claim of the lemma, it suffices to take $w = w^n$. □

Theorem 3. $L(\mathcal{A}/{\equiv_M}) = L(\mathcal{A})$.

Proof. The inclusion $L(\mathcal{A}/{\equiv_M}) \supseteq L(\mathcal{A})$ is trivial. Let $t \in L(\mathcal{A}/{\equiv_M})$, i.e., $t \Longrightarrow B$ for some block B where $B \cap F \neq \emptyset$. Lemma 4.9 implies that $t \Longrightarrow w$ such that $w \in F$. □

Note that the theorem also covers the case of reducing automata using downward simulations (and bisimulations) alone. Indeed, given any downward simulation D, the identity is always an upward simulation induced by D. Then, the combined preorder $D \oplus id^{-1}$ equals D, which means that we can reduce the automaton using $\equiv_D$. In particular, this covers as special cases the proofs of correctness of reducing automata using downward bisimulations and simulation equivalences stated in [ABH+08c].

Corollary 1. $L(\mathcal{A}/{\equiv_D}) = L(\mathcal{A})$ *for the downward simulation equivalence* $\equiv_D$ *on* $\mathcal{A}$.

4.3 Variants of the Combined Relation

Theorem 3 and Lemmas 4.3 and 4.4 allow us to consider a spectrum of relations suitable for reducing tree automata. We now examine properties of the relations from this spectrum that arise when we consider the identity, the downward bisimulation equivalence, and the downward simulation preorder as the inducing relation D for both the upward bisimulation equivalence and upward simulation preorder.

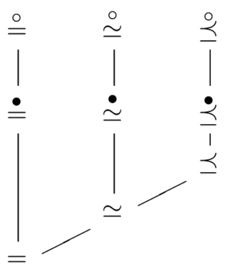
In this section we use a special notation to systematically distinguish various types of mediated preorders. The notation consists of two parts: a relation symbol and an additional symbol above the relation symbol. The relation symbol denotes the type of the inducing preorder. Namely, $=$ denotes the identity, $\simeq$ denotes the downward bisimulation equivalence, and $\preceq$ the downward simulation preorder. The additional symbol then denotes the type of the upward relation. We use $\bullet$ for the upward bisimulation equivalence and $\circ$ for the upward simulation preorder. No additional symbol corresponds to the maximum equivalence embedded in the downward relation itself—the downward (bi-)simulations can be viewed as mediated equivalences where the role of the upward relation is played by the identity. For example, $\overset{\circ}{\preceq}$ denotes the mediated equivalence

$(D \oplus U^{-1}) \cap (D \oplus U^{-1})^{-1}$ where D is the downward simulation preorder and U is the upward simulation preorder induced by D.

Ordering the Mediated Preorders wrt. their Coarseness. From the definition of a combined preorder, it clearly follows that, for a fixed inducing relation D, if we are choosing the type of the upward relations U from the strongest one to the coarsest one, i.e., starting from the identity and going through the upward bisimulation induced by D to the upward simulation induced by D, we obtain coarser and coarser combined preorders $D \oplus U^{-1}$.

On the other hand, if the inducing preorder D is growing, the situation is different. From the definition of the upward simulation and bisimulation, we can see that the upward simulation preorder/bisimulation equivalence U induced by D and thus also the relation $D \circ U^{-1}$ are growing too. But, when computing $D \oplus U^{-1}$ by pruning $D \circ U^{-1}$, the larger relation D means that more pairs are to be pruned since they are violating Condition (ii) from the definition of $\oplus$. In general, having two downward simulations D_1 and D_2 with $D_1 \subseteq D_2$, we are guaranteed that the upward simulation preorder/bisimulation equivalence U_1 induced by D_1 is included in the upward simulation preorder/bisimulation equivalence U_2 induced by D_2. Therefore, we know that $D_1 \circ U_1^{-1} \subseteq D_2 \circ U_2^{-1}$, but the combined preorders $D_1 \oplus U_1^{-1}$ and $D_2 \oplus U_2^{-1}$ can be in any relation w.r.t. set inclusion or incomparable (although, in our experiments, the former one usually *is* included in the latter one).

Based on these observations, we obtain the partial ordering of all the considered types of combined equivalences according to inclusion which is depicted in Figure 4.2. For a tree automaton $\mathcal{A}$, we denote by $\equiv(\mathcal{A})$ the combined equivalence of type $\equiv$ on $\mathcal{A}$. In the figure, the line from $\equiv_1$ up to $\equiv_2$ means that for any automaton $\mathcal{A}$, $\equiv_1(\mathcal{A}) \subseteq \equiv_2(\mathcal{A})$. It is not hard to find an automaton $\mathcal{A}$ showing that all these relationships are strict, i.e., such that for each of the edges in the figure, $\equiv_1(\mathcal{A}) \subsetneq \equiv_2(\mathcal{A})$. We construct such an automaton in Example 1. Most of our tree automata examples in this section use just leaf and unary rules, therefore they may be drawn in the same way as is usual for word automata.



$\overset{\circ}{=}$
$\overset{\circ}{\simeq}$
$\overset{\circ}{\preceq}$
$\overset{\bullet}{=}$
$\overset{\bullet}{\simeq}$
$\overset{\bullet}{\preceq}$
$\preceq$
$\simeq$
$=$

Figure 4.2: Coarseness of various types of combined equivalences

Example 1. *Let $Q = \{q, r, s, t, u, v, w, x, y, z\}$ be a set of states and let Σ be a ranked alphabet such that $\Sigma_0 = \{l\}$ and $\Sigma_1 = \{a, b, c\}$. The automaton $\mathcal{A} = (\Sigma, Q, \Delta_1, \{x\})$ proves strictness of the relations in Figure 4.2. For each*

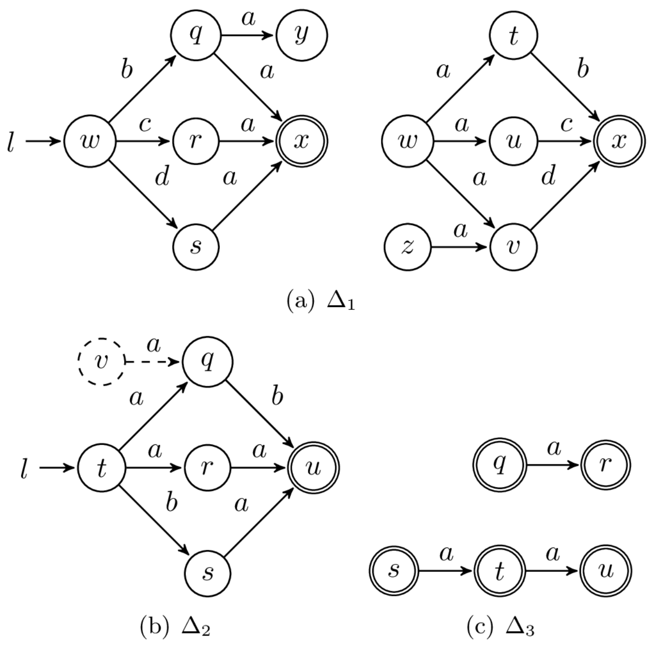


Figure 4.3: Transition relations of automata proving the non-inclusion relationships from Figure 4.2, and of an automaton proving that it is not safe to use mediated equivalences that arise from preorders included in the language inclusion preorder that are not downward simulations.

two types of relations from Figure 4.2 such that $\equiv_2$ is above $\equiv_1$, $\equiv_1(\mathcal{A}) \subsetneq \equiv_2(\mathcal{A})$ holds. The transition relation Δ_1 is depicted in Figure 4.3(a). In the table below, there are listed the appropriate mediated equivalences for all the combinations of the considered types of inducing and induced relations. For each type of combination, we list nontrivial equivalence classes of the resulting mediated equivalence:

$\overset{\circ}{=}$: $\{q,r,s\}$	$\overset{\circ}{\simeq}$: $\{t,u\},\{q,r,s\}$	$\overset{\circ}{\preceq}$: $\{t,u,v\},\{q,r,s\},\{x,z\}$
$\overset{\bullet}{=}$: $\{r,s\}$	$\overset{\bullet}{\simeq}$: $\{t,u\},\{r,s\}$	$\overset{\bullet}{\preceq}$: $\{t,u,v\},\{r,s\}$
$=$:	$\simeq$: $\{t,u\}$	$\preceq$: $\{t,u,v\}$

It is now easy to check that for the automaton $\mathcal{A}$, all the inclusions from Figure 4.2 are strict. □

To complete the picture, we need to show that the types of combined relations that are not connected in Figure 4.2 are really incomparable. In Example 2, we construct automata $\mathcal{A}_1, \mathcal{A}_2$ such that for each pair $\equiv_1, \equiv_2$ of types of mediated equivalences that are not connected in Figure 4.2, neither $\equiv_1(\mathcal{A}_i) \subseteq \equiv_2(\mathcal{A}_i)$ nor $\equiv_1(\mathcal{A}_i) \supseteq \equiv_2(\mathcal{A}_i)$ holds for some $i \in \{1,2\}$.

Example 2. *Let $Q = \{q,r,s,t,u,v\}$ be a set of states and let Σ be a ranked alphabet such that $\Sigma_0 = \{l\}$ and $\Sigma_1 = \{a,b,c\}$. All the incomparability results show up taking automata $\mathcal{A}_1 = (Q \setminus \{v\}, \Sigma, \Delta_2 \setminus \{v \xrightarrow{a} q\}, \{u\})$ and $\mathcal{A}_2 =$*

$(\Sigma, Q, \Delta_2, \{u\})$ where the transition relation Δ_2 is depicted in Figure 4.3(b). One can easily check that $\overset{\bullet}{=}(\mathcal{A}_1)$ and $\overset{\circ}{=}(\mathcal{A}_1)$ define just one nontrivial equivalence class $\{r, s\}$ and thus they are incomparable with $\overset{\circ}{\simeq}(\mathcal{A}_1), \overset{\bullet}{\preceq}(\mathcal{A}_1), \overset{\circ}{\preceq}(\mathcal{A}_1)$ that define only one nontrivial equivalence class $\{q, r\}$. In the case of the automaton $\mathcal{A}_2$, the added transition $v \xrightarrow{a} q$ distinguishes the downward simulation from the downward bisimulation. Analogically as for $\mathcal{A}_1$, we have that $\overset{\bullet}{\simeq}(\mathcal{A}_2)$ and $\overset{\circ}{\simeq}(\mathcal{A}_2)$ define just one nontrivial equivalence class $\{r, s\}$ and thus they are incomparable with $\overset{\bullet}{\preceq}(\mathcal{A}_2)$ and $\overset{\circ}{\preceq}(\mathcal{A}_2)$ that define only one nontrivial equivalence class $\{q, r\}$. This gives all the incomparability relationships. □

According to our experiments presented in Section 6.5, the reduction capabilities are rising when we move in Figure 4.2 not only in the bottom-up direction, but also in the left-right direction. As a trade-off, the computational complexity of constructing the relations is rising in the same way from the bottom to the top and from the left to the right.

Impossibility of Relaxing the Need of Downward Simulations. It is easy to see that when not considering combined relations (and when not thinking of the computational complexity), one can replace the use of downward simulations in reducing the size of tree automata by a use of any preorder which is included in the so called language inclusion preorder $LP^{\subseteq}$ ($(q, r) \in LP^{\subseteq} \iff L(q) \subseteq L(r)$). A natural question comes forward: Is it also possible to induce (and combine by $\oplus$) an upward simulation with any preorder included in $LP^{\subseteq}$? Here, we give a negative answer. Not all preorders included in $LP^{\subseteq}$ can be used within the operator $\oplus$ for reducing automata. We prove this claim by the following counterexample.

Example 3. *Consider an automaton $\mathcal{A} = (\Sigma, Q, \Delta_3 \cup \{\xrightarrow{l} x \mid x \in Q\}, F)$ where $Q = \{q, r, s, t, u\}$, $\Sigma_0 = \{l\}$, $\Sigma_1 = \{a\}$, Δ_3 is depicted in Figure 4.3(c), and $F = Q$. Let us choose the relation $R = id \cup \{(q, r), (r, t), (q, t)\}$, which is apparently contained in $LP^{\subseteq}$, as the inducing preorder. Notice that since we deal here only with unary and leaf symbols, upward simulation and bisimulation do not depend on the inducing relation. We can choose the relation $U = id \cup \{(q, t)\}$ as an reflexive and transitive upward simulation induced by R. Then, we obtain $R \circ U^{-1} = R \cup U^{-1} \cup \{(r, q)\}$. The pair (r, q) is present in $R \circ U^{-1}$ because of rRt and qUt. Let $M = R \oplus U^{-1}$ be the mediated preorder. $R \circ U^{-1}$ itself is already a preorder, and therefore $M = R \circ U^{-1}$. We have obtained an equivalence class $\{q, r\}$ of $M \cap M^{-1}$. This means that the quotient automaton $\mathcal{A}/M \cap M^{-1}$ contains the rule $\{q, r\} \xrightarrow{a} \{q, r\}$. This definitely changes the language, since $\mathcal{A}$ does not contain loops (accepts only finitely many trees), but the language of $\mathcal{A}/M \cap M^{-1}$ is infinite.*

Observe that if we take a downward simulation as the inducing preorder, such a situation does not arise. The problem above is caused by the presence of (r, q) in $R \circ U^{-1}$, which is enabled by rRt. If R was a downward simulation containing (r, t), then R would have to contain even (q, s) from the definition of a downward simulation. So, we would get $r(R \oplus U^{-1})qRs$ which according to Condition (ii) of the definition of $\oplus$ enforces $r(R \circ U^{-1})s$. However, $r(R \circ U^{-1})s$

does not hold for any pair of an inducing downward simulation R and an induced upward simulation U (not even when one considers the maximal ones), and so the pair (r, q) is not present in any mediated preorder, and we are never allowed to collapse q and r. $\square$

Mediated Bisimulation contra Forward Followed by Backward. In [HMM07a], the following straightforward approach to combining downward and upward bisimulations is presented: Compute a quotient automaton w.r.t. downward bisimulation equivalence and reduce it again using upward bisimulation equivalence induced by identity, or alternatively, proceed the other way around (compute quotient w.r.t. upward bisimulation equivalence induced by identity and then compute quotient of the result w.r.t. downward bisimulation equivalence). One could ask whether this approach gives different results from our mediated bisimulation $\overset{\bullet}{\simeq}$ (this question was in fact asked during the presentation of our work [ABH+09] at CIAA'08). We give an answer in Example 4 where we show that all the three techniques are in general incomparable.

Example 4. *For each of the three reduction techniques, we now present an automaton such that reducing it by the particular reduction technique gives better result (i.e., the resulting automaton has less states) than reducing it by the other two techniques. The automata were automatically generated using a random automata generator thus they are perhaps not the smallest possible ones. We have checked validity these of counterexamples using our implementation the three methods. The automaton $A_1 = (Q_1, \Sigma_1, \Delta_1, Q_1)$ can be reduced most when using composed bisimulation, in the case of the automaton $A_2 = (Q_2, \Sigma_2, \Delta_2, Q_2)$, using forward bisimulation reduction followed by forward (downward) bisimulation reduction is the best, and the automaton $A_3 = (Q_3, \Sigma_3, \Delta_3, Q_3)$ can be reduced most using forward (downward) bisimulation reduction followed by forward bisimulation reduction. The sets $\Sigma_1, \Sigma_2, \Sigma_3, Q_1, Q_2, Q_3, \Delta_1, \Delta_2, \Delta_3$ look as follows:*

$$Q_1 = \{q_x, q_0, q_1, q_2, q_3, q_4, r_x, r_0, r_1, r_2, r_3, r_4\}, \Sigma_1 = \{x, a, b, c, d\}$$

$$\begin{array}{llllll}
\Delta_3 : & & & & & \\
q_x \xrightarrow{x} q_x, & q_0 \xrightarrow{d} q_1, & q_2 \xrightarrow{d} q_1, & r_x \xrightarrow{x} r_x, & r_1 \xrightarrow{d} r_0, & r_1 \xrightarrow{d} r_2, \\
q_x \xrightarrow{a} q_4, & q_0 \xrightarrow{d} q_0, & q_2 \xrightarrow{d} q_0, & r_4 \xrightarrow{a} r_x, & r_0 \xrightarrow{d} r_0, & r_0 \xrightarrow{d} r_2, \\
q_x \xrightarrow{a} q_3, & q_1 \xrightarrow{d} q_4, & q_3 \xrightarrow{c} q_4, & r_3 \xrightarrow{a} r_x, & r_4 \xrightarrow{d} r_1, & r_4 \xrightarrow{c} r_3, \\
q_x \xrightarrow{b} q_3, & q_2 \xrightarrow{c} q_3, & q_3 \xrightarrow{c} q_2, & r_3 \xrightarrow{b} r_x, & r_3 \xrightarrow{c} r_2, & r_2 \xrightarrow{c} r_3, \\
q_0 \xrightarrow{c} q_0, & q_2 \xrightarrow{c} q_1, & q_3 \xrightarrow{d} q_2, & r_0 \xrightarrow{c} r_0, & r_1 \xrightarrow{c} r_2, & r_2 \xrightarrow{d} r_3, \\
q_0 \xrightarrow{c} q_3, & q_2 \xrightarrow{c} q_0, & q_3 \xrightarrow{d} q_1, & r_3 \xrightarrow{c} r_0, & r_0 \xrightarrow{c} r_2, & r_1 \xrightarrow{d} r_3, \\
q_0 \xrightarrow{c} q_1, & q_2 \xrightarrow{d} q_3, & q_3 \xrightarrow{d} q_0, & r_1 \xrightarrow{c} r_0, & r_3 \xrightarrow{d} r_2, & r_0 \xrightarrow{d} r_3, \\
q_0 \xrightarrow{d} q_2, & q_2 \xrightarrow{d} q_2, & q_4 \xrightarrow{d} q_4, & r_2 \xrightarrow{d} r_0, & r_2 \xrightarrow{d} r_2, & r_4 \xrightarrow{d} r_4
\end{array}$$

$$Q_2 = Q_3 = \{q_0, q_1, q_2\}, \Sigma_2 = \Sigma_3 = \{a, b, c, d\}$$

$$
\begin{array}{lll|lll}
\Delta_1 : & & & \Delta_2 : & & \\
\xrightarrow{a} q_2, & q_2 \xrightarrow{c} q_1, & q_2 \xrightarrow{d} q_1, & \xrightarrow{a} q_1, & q_1 \xrightarrow{c} q_0, & q_1 \xrightarrow{d} q_2, \\
\xrightarrow{a} q_1, & q_1 \xrightarrow{c} q_2, & q_2 \xrightarrow{d} q_2, & \xrightarrow{a} q_2, & q_0 \xrightarrow{c} q_0, & q_1 \xrightarrow{d} q_1, \\
\xrightarrow{b} q_1, & q_1 \xrightarrow{c} q_1, & q_0 \xrightarrow{d} q_0, & \xrightarrow{b} q_1, & q_2 \xrightarrow{c} q_0, & q_1 \xrightarrow{d} q_0, \\
\xrightarrow{b} q_2, & q_1 \xrightarrow{c} q_0, & q_0 \xrightarrow{d} q_1 & \xrightarrow{b} q_2, & & q_0 \xrightarrow{d} q_0. \\
 & q_0 \xrightarrow{c} q_1, & & & &
\end{array}
$$

□

On the other hand, in all our test cases, backward bisimulation followed by forward bisimulation behaved in a very similar way to composed bisimulation—a more thorough experimental comparison is to be done in the future.

We also note that applying the forward and backward bisimulations in succession has the advantage that the second relation is computed on a smaller input automaton, and that the algorithm for computing upward bisimulation induced by identity from [HMM07a] is asymptotically slightly faster than our algorithm for computing general upward bisimulation (induced by any preorder), presented in Section 4.4.6. On the other hand, our algorithm is more generic and can be very easily implemented within the general framework for computing tree automata simulations and bisimulations presented in Section 4.4.

Upward Simulation Equivalences Cannot be Used For Quotienting. As we have mentioned at the beginning of Section 4.2, induced upward simulation equivalences cannot be always safely used for quotienting. This shows Example 5 where quotienting w.r.t. upward simulation equivalence induced by downward bisimulation equivalence extends the language.

Example 5. *Let $\mathcal{A} = (\Sigma, Q, \Delta, F)$ be an automaton where $Q = \{q, r, s, t, f\}$, $\Sigma = \{a, b, c\}$, $F = \{f\}$ and Δ is given as follows:*

$$
\begin{array}{ll}
\xrightarrow{a} q, & (s, q) \xrightarrow{c} f, \\
\xrightarrow{a} s, & (q, r) \xrightarrow{c} f, \\
\xrightarrow{b} r, & (r, t) \xrightarrow{c} f, \\
\xrightarrow{b} t &
\end{array}
$$

We have that q and s are downward bisimilar and r and t are downward bisimilar. Therefore, q and r are upward simulation equivalent w.r.t. the upward simulation equivalence $\equiv_U$ induced by the downward bisimulation equivalence. Observe that if we merge states q and r, the quotient automaton $\mathcal{A}/{\equiv_U}$ will accept the tree $c(b, a)$, however, $L(\mathcal{A}) = \{c(a, a), c(a, b), c(b, b)\}$ does not contain $c(b, a)$. □

A note on word automata. We note that all the above results (except Example 5) carry over to word automata. The inclusion properties from Figure 4.2 hold for word automata too since they can be seen as a special case of tree automata. Moreover, our automata examples proving strictness of the relationships and incomparability relationships are built using just leaf and unary rules, and so they are valid for word automata as well.

4.4 Computing the Proposed Relations

In this section, we give algorithms for computing the tree automata simulation preorders and bisimulation equivalences. The algorithms are based on translations of tree automata simulation and bisimulation problems into problems of computing maximal simulation or bisimulation on labelled transition systems. This is, we translate tree automata into labelled transition systems and (i) run an LTS simulation algorithm (we use Algorithm 1) in the case of tree automata simulations; and (ii) run an LTS bisimulation algorithm in the case of tree automata bisimulations (we use the algorithm from [Val09] by Valmari). The simulation or bisimulation on the original tree automaton is then obtained directly in the form of simulation or bisimulation on states of the LTS, respectively. Finally, we give a simple procedure for computing mediated preorder induced by a given downward simulation and upward simulation.

4.4.1 Computing Downward Simulation

As mentioned above, our approach to computing downward simulation consists of two parts: (1) we translate the maximal downward simulation problem over tree automata into a corresponding maximal simulation problem over LTS, and (2) we compute the simulation preorder on the obtained LTS using Algorithm 1. Below, we describe how the translation is carried out.

We will work with the set $Lhs_{\mathcal{A}}$ of *left-hand sides* of rules of $\mathcal{A}$, i.e., the set of tuples of the form $(q_1, \ldots, q_n)$ where $(q_1, \ldots, q_n) \xrightarrow{a} q$ for some a and q. We will drop the reference to $\mathcal{A}$ if no confusion may arise. The idea of the translation comes from the following alternative definition of downward simulation.

Extended Downward Simulation. An *Extended downward simulation* is a relation $\bar{D} \subseteq Q \times Q \cup Lhs \times Lhs$ such that

1. for any $q, r \in Q$, if $q\bar{D}r$ and $(q_1, \ldots, q_n) \xrightarrow{a} q$, then $(r_1, \ldots, r_n) \xrightarrow{a} r$ with $(q_1, \ldots, q_n)\bar{D}(r_1, \ldots, r_n)$,
2. for any $(q_1, \ldots, q_n), (r_1, \ldots, r_n) \in Lhs$, if $(q_1, \ldots, q_n)\bar{D}(r_1, \ldots, r_n)$, then $q_i\bar{D}r_i$ for all $1 \leq i \leq n$.

The following proposition is an obvious consequence of this definition.

Proposition 4.1. *A relation $D \subseteq Q \times Q$ is a downward simulation on $\mathcal{A}$ if and only if there is an extended downward simulation on $\mathcal{A}$ such that its restriction to Q is D.*

The translation is based on the observation that when viewing Q and Lhs as two kinds of nodes of an LTS such that for every transition $(q_1, \ldots, q_n) \xrightarrow{a} q$, there is an a-labelled transition from q to $(q_1, \ldots, q_n)$ and for every $i : 1 \leq i \leq n$, there is an i-labelled transition from $(q_1, \ldots, q_n)$ to q_i, then the definition above closely resembles the definition of normal LTS simulation. Formally, we translate a tree automaton $\mathcal{A}$ to the LTS $\mathcal{A}^\bullet = (\Sigma^\bullet, Q^\bullet, \Delta^\bullet)$ where:

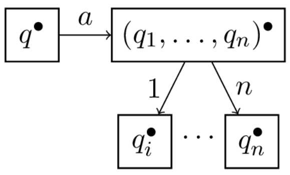


Figure 4.4: The part of $\mathcal{A}^\bullet$ created for a rule $(q_1,\ldots,q_n) \xrightarrow{a} q \in \Delta$.

- 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 $(q_1,\ldots,q_n) \in Lhs$.
- 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 $(q_1,\ldots,q_n) \xrightarrow{a} q$ of $\mathcal{A}$, the set $\Delta^\bullet$ contains both the transition $q^\bullet \xrightarrow{a} (q_1,\ldots,q_n)^\bullet$ 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 translation is illustrated on Figure 4.4. The following theorem shows correctness of the translation.

Theorem 4. *A relation $\bar{D}$ is an extended downward simulation on $\mathcal{A}$ if and only if the relation $D^\bullet = \{(x^\bullet, y^\bullet) \mid xDy\}$ is a simulation on $\mathcal{A}^\bullet$.*

Proof. (if) Assume that $D^\bullet$ is a simulation on $\mathcal{A}^\bullet$. We show that $\bar{D}$ is an extended downward simulation on $\mathcal{A}$. Suppose that there are $q, r \in Q$ with $q\bar{D}r$ and $(q_1,\ldots,q_n) \xrightarrow{a} q$. Since $q\bar{D}r$ we know that $q^\bullet D^\bullet r^\bullet$; and since $(q_1,\ldots,q_n) \xrightarrow{a} q$ we know by definition of $\mathcal{A}^\bullet$ that $q^\bullet \xrightarrow{a} (q_1,\ldots,q_n)^\bullet$. Since $D^\bullet$ is a simulation, there are $r_1,\ldots,r_n \in Q$ with $r^\bullet \xrightarrow{a} (r_1,\ldots,r_n)^\bullet$ and $(q_1,\ldots,q_n)^\bullet D^\bullet (r_1,\ldots,r_n)^\bullet$ and thus $(q_1,\ldots,q_n)\bar{D}(r_1,\ldots,r_n)$.

Suppose now that there are left-hand sides $(q_1,\ldots,q_n), (r_1,\ldots,r_n) \in Lhs$ with $(q_1,\ldots,q_n)\bar{D}(r_1,\ldots,r_n)$. By the definition of $D^\bullet$, $(q_1,\ldots,q_n)^\bullet D^\bullet (r_1,\ldots,r_n)^\bullet$. By definition of $\mathcal{A}^\bullet$ we know that $(q_1,\ldots,q_n)^\bullet \xrightarrow{i} q_i^\bullet$ for each $i : 1 \leq i \leq n$, and that $(q_1,\ldots,q_n)^\bullet$ does not have any other outgoing edges. 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 $q_i^\bullet D^\bullet r_i^\bullet$. This means that $q_i\bar{D}r_i$ for each $i : 1 \leq i \leq n$.

(only if) Suppose that $\bar{D}$ is an extended downward simulation on $\mathcal{A}$. We prove that $D^\bullet$ is a simulation on $\mathcal{A}$. Suppose that for some $q, r \in Q$, $q^\bullet D^\bullet r^\bullet$ and $q^\bullet \xrightarrow{a} (q_1,\ldots,q_n)^\bullet$. Since $q^\bullet D^\bullet r^\bullet$, we know that $q\bar{D}r$, and since $q^\bullet \xrightarrow{a} (q_1,\ldots,q_n)^\bullet$, we know by definition of $\mathcal{A}^\bullet$ that $(q_1,\ldots,q_n) \xrightarrow{a} q$. Since $\bar{D}$ is an extended downward simulation, there are $r_1,\ldots,r_n \in Q$ with $(r_1,\ldots,r_n) \xrightarrow{a} r$ and $(q_1,\ldots,q_n)\bar{D}(r_1,\ldots,r_n)$. By definitions of $D^\bullet$ and $\mathcal{A}^\bullet$, we obtain that $(q_1,\ldots,q_n)^\bullet D^\bullet (r_1,\ldots,r_n)^\bullet$ and $r^\bullet \xrightarrow{a} (r_1,\ldots,r_n)^\bullet$.

Finally, suppose that $(q_1,\ldots,q_n)^\bullet D^\bullet (r_1,\ldots,r_n)^\bullet$ for some left-hand sides $(q_1,\ldots,q_n), (r_1,\ldots,r_n) \in Lhs$. By definition of $\mathcal{A}^\bullet$, we have $(q_1,\ldots,q_n)^\bullet \xrightarrow{i} q_i^\bullet$

and $(r_1, \dots, r_n)^\bullet \xrightarrow{i} r_i^\bullet$ for each $i : 1 \leq i \leq n$ and we know that there are no other outgoing arcs of $(r_1, \dots, r_n)^\bullet$ and $(q_1, \dots, q_n)^\bullet$. Therefore, since $(q_1, \dots, q_n)^\bullet D^\bullet (r_1, \dots, r_n)^\bullet$, it follows that $q_i^\bullet D^\bullet r_i \bullet$ and hence $q_i \bar{D} r_i$ by the definition of $D^\bullet$. Therefore, $(q_1, \dots, q_n) \bar{D} (r_1, \dots, r_n)$ by the definition of $\bar{D}$. □

Due to Theorem 4, the maximal extended downward simulation $\preceq_D$ on $\mathcal{A}$ corresponds to the simulation preorder on $\mathcal{A}^\bullet$ which can be computed by constructing the LTS $\mathcal{A}^\bullet$ and running Algorithm 1 on it with the initial partition-relation pair being simply $\langle P^\bullet, Rel^\bullet \rangle = (\{Q^\bullet\}, \{(Q^\bullet, Q^\bullet)\})$ (this is, we initially consider all states of the LTS $\mathcal{A}^\bullet$ equal, and hence they form a single class of $P^\bullet$, which is related to itself in $Rel^\bullet$). By Proposition 4.1, the downward simulation preorder on $\mathcal{A}$ is then obtained simply by restricting $\preceq_D$ to Q.

4.4.2 Complexity of Computing Downward Simulation

The complexity of computing the downward simulation preorder on $\mathcal{A}$ naturally consists of the price of compiling the tree automaton $\mathcal{A}$ into its corresponding LTS $\mathcal{A}^\bullet$, the price of building the initial partition-relation pair $\langle P^\bullet, Rel^\bullet \rangle$, and the price of running Algorithm 1 on $\mathcal{A}^\bullet$ and $\langle P^\bullet, Rel^\bullet \rangle$.

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. Under this assumption, we can use the inequalities $|Q| - 1 \leq |Lhs| \leq |\Delta|$ when deriving complexity of our algorithms. Further, we expect the input automaton $\mathcal{A}$ to be encoded as a list of states $q \in Q$ and a list of the left-hand sides $l = (q_1, \dots, q_n) \in Lhs$. Each left-hand side l is encoded by an array of (pointers to) the states $q_1, \dots, q_n$, plus a list containing a pointer to the so-called a-list for each $a \in \Sigma$ such that there is an a transition from l in Δ. Each a-list is then a list of (pointers to) all the states $q \in Q$ such that $l \xrightarrow{a} q$. We call this representation the *lhs-list* automata encoding. Then, the complexity of preparing the input for computing the downward simulation on $\mathcal{A}$ via Algorithm 1 is given by the following lemma.

Lemma 4.10. *The LTS $\mathcal{A}^\bullet$ and the partition-relation pair $\langle P^\bullet, Rel^\bullet \rangle$ can be derived in time and space $\mathcal{O}(\hat{r}|Q| + |\Delta| + (\hat{r} + |\Sigma|)|Lhs|)$.*

Proof. The state-list encoding of the LTS $\mathcal{A}^\bullet$ that Algorithm 1 takes as its input (c.f. Chapter 3) can be obtained from the lhs-list encoding of $\mathcal{A}$ in the following steps:

1. For all $q \in Q$, add $q^\bullet$ into the state-list encoding of $\mathcal{A}^\bullet$ (and also create an additional pointer from q to $q^\bullet$, which we will need later on).
2. For each $l = (q_1, \dots, q_n) \in Lhs$,
    a) add $l^\bullet$ into the state-list encoding of $\mathcal{A}^\bullet$,
    b) for each $a \in \Sigma$ and each right-hand side r in the a-list of l, add $r^\bullet$ into $pre_a(l^\bullet)$, i.e. add the $r^\bullet \xrightarrow{a} l^\bullet$ 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 (we use the pointers from q to $q^\bullet$ introduced within step 1.).

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

After creating the temporary arrays, we traverse the lhs-list representation of $\mathcal{A}$ in time $\mathcal{O}(|Q| + |\Delta| + \hat{r}|Lhs|)$ while building the state-list representation (with arrays used instead of state-lists) of $\mathcal{A}^\bullet$ 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 $\mathcal{A}$ while creating the a-labelled transitions of $\mathcal{A}^\bullet$ for $a \in \Sigma$, and $\hat{r}|Lhs|$ to traversing the left-hand sides while creating the i-labelled transitions of $\mathcal{A}^\bullet$ for $1 \leq i \leq \hat{r}$. 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 $\mathcal{A}^\bullet$ is done in time $\mathcal{O}(\hat{r}|Q| + |\Delta| + (\hat{r} + |\Sigma|)|Lhs|)$.

The space complexity corresponds to the size of the temporary arrays and the size of the resulting LTS $\mathcal{A}^\bullet$, which is $\mathcal{O}(|Q| + |\Delta| + \hat{r}|Lhs|)$. Indeed, we need space $O(|Q|)$ to represent states, $O(|\Delta|)$ to represent the a-labelled transitions of $\mathcal{A}^\bullet$ for symbols $a \in \Sigma$, and $O(\hat{r}|Lhs|)$ to represent the i-labelled transitions of $\mathcal{A}^\bullet$ for symbols $1 \leq i \leq \hat{r}$. In total, we obtain the same formula as in the case of the time complexity, i.e. $\mathcal{O}(\hat{r}|Q| + |\Delta| + (|\Sigma| + |\hat{r}|)|Lhs|)$.

Finally, the creation of $\langle P^\bullet, Rel^\bullet \rangle$ is trivial, and its complexity is apparently covered by the complexity of creating $\mathcal{A}^\bullet$. $\square$

In order to instantiate the complexity of running Algorithm 1 for $\mathcal{A}^\bullet$, let us denote $\equiv_D$ the maximal equivalence included in the maximal extended downward simulation on $\mathcal{A}$.

Lemma 4.11. *Algorithm 1 computes the simulation preorder on $\mathcal{A}^\bullet$ for the initial partition-relation pair $\langle P^\bullet, Rel^\bullet \rangle$ in the time $\mathcal{O}((|\Sigma| + \hat{r})|Lhs||Lhs/{\equiv_D}| + |\Delta||Lhs/{\equiv_D}|)$ and the space $\mathcal{O}((|\Sigma| + \hat{r})|Lhs||Lhs/{\equiv_D}|)$.*

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

$$\mathcal{O}((|\Sigma| + \hat{r})(|Q| + |Lhs|)(|Q/{\equiv_D}| + |Lhs/{\equiv_D}|) \\ + ((|\Delta| + \hat{r}Lhs)(|Q/{\equiv_D}| + |Lhs/{\equiv_D}|))).$$

Recall that $|Q| - 1 \leq |Lhs| \leq |\Delta|$ since 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. Therefore, the time complexity amounts to

$$\mathcal{O}((|\Sigma| + \hat{r})|Lhs||Lhs/{\equiv_D}| + |\Delta||Lhs/{\equiv_D}|)$$

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

$$\mathcal{O}((|\Sigma| + \hat{r})|Lhs||Lhs/{\equiv_D}|).$$

$\square$

The complexity of computing the downward simulation for the tree automaton $\mathcal{A}$ via the LTS $\mathcal{A}^\bullet$ can now be obtained by simply summing the complexities of computing $\mathcal{A}^\bullet$ and $\langle P^\bullet, Rel^\bullet\rangle$ and of running Algorithm 1 on them.

Theorem 5. *The downward simulation preorder on $\mathcal{A}$ can be computed in time $\mathcal{O}((|\Sigma|+\hat{r})|Lhs||Lhs/{\equiv_D}|+|\Delta||Lhs/{\equiv_D}|)$ and space $\mathcal{O}((|\Sigma|+\hat{r})|Lhs||Lhs/{\equiv_D}|+|\Delta|)$.*

Note that in the special case of $\hat{r} = 1$ (corresponding to a word automaton viewed as a tree automaton), we have $|Lhs| = |Q|$, which leads to the same complexity as Algorithm 1 has when applied directly on word automata.

4.4.3 Computing Upward Simulation

Given a preorder D on Q, we want to compute the upward simulation preorder induced by D. We will need the notion of *environment*, which is a tuple of the form $((q_1, \ldots, q_{i-1}, \square, q_{i+1}, \ldots, q_n), a, q)$ obtained by removing a state q_i, $1 \leq i \leq n$, from the i^{th} position of the left hand side of a rule $((q_1, \ldots, q_{i-1}, q_i, q_{i+1}, \ldots, q_n), a, q)$, and by replacing it by a special symbol $\square \notin Q$ (called a *hole* below). Like for transition rules, we write $[(q_1, \ldots, \square_i, \ldots, q_n) \xrightarrow{a} q]$ provided $((q_1, \ldots, q_{i-1}, q_i, q_{i+1}, \ldots, q_n), a, q) \in \Delta$ for some $q_i \in Q$. We denote the set of all environments of $\mathcal{A}$ by $Env_\mathcal{A}$ and we will drop the reference to $\mathcal{A}$ if no confusion may arise.

We proceed in a similar manner as in Section 4.4.1 with downward simulation. First, we extend the definition of upward simulation to the set of environments.

Extended Upward Simulation. Given a preorder D on Q, an *extended upward simulation $\bar{U}$ induced by D* is a binary relation on $Q \cup Env$ such that if $q\bar{U}r$, then

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

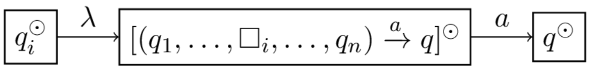


Figure 4.5: The part of $\mathcal{A}^\odot$ created for a rule $(q_1,\ldots,q_n)\xrightarrow{a}q\in\Delta$.

The following proposition follows directly from this definition.

Proposition 4.2. *A relation U is an upward simulation on $\mathcal{A}$ induced by D if and only if there is an extended upward simulation on $\mathcal{A}$ induced by D such that its restriction to Q is U.*

Analogically as in Section 4.4.1, we notice that we can view the extended upward simulation as a simulation on a labelled transition system with two types of nodes, one corresponding to states and the other corresponding to environments of $\mathcal{A}$. Formally, we define the LTS $\mathcal{A}^\odot=(\Sigma^\odot,Q^\odot,\Delta^\odot)$ as follows:

- The set $Q^\odot$ contains a state $q^\odot$ for each $q\in Q$, and it also contains a state $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q]^\odot$ for each environment $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q]\in Env$.
- The set $\Sigma^\odot$ contains each symbol $a\in\Sigma$ and also a special symbol $\lambda\notin\Sigma$.
- For each transition rule $(q_1,\ldots,q_n)\xrightarrow{a}q$ of $\mathcal{A}$ and for each $i:1\le i\le n$, the set $\Delta^\odot$ contains the transition $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q]^\odot\xrightarrow{a}q^\odot$ and the transition $q_i^\odot\xrightarrow{\lambda}[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q]^\odot$.
- The sets $Q^\odot$, $\Sigma^\odot$, and $\Delta^\odot$ do not contain any other elements.

The translation is illustrated on Figure 4.5. We also have to take into account the inducing preorder D. Therefore, we define the initial relation I_D to be the smallest binary relation on $Q^\odot$ containing all pairs of states of $\mathcal{A}$, i.e., all pairs $(q_1^\odot,q_2^\odot)$ for each $q_1,q_2\in Q$ and also all pairs of environments $([(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q]^\odot,[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a}r]^\odot)$ such that q_jDr_j for each $j:1\le j\ne i\le n$. The following theorem shows correctness of the translation.

Theorem 6. *A relation $\bar{U}$ is an extended upward simulation on $\mathcal{A}$ induced by D if and only if the relation $U^\odot=\{(x^\odot,y^\odot)\mid x\bar{U}y\}$ is a simulation on $\mathcal{A}^\odot$ included in I_D.*

Proof. Assume that $U^\odot$ is a simulation on $\mathcal{A}^\odot$ included in I_D. We will show that $\bar{U}$ is an extended upward simulation induced by D. Let $q\bar{U}r$ and $(q_1,\ldots,q_n)\xrightarrow{a}q'$ where $q_i=q$. We know that $q^\odot U^\odot r^\odot$, and since $(q_1,\ldots,q_n)\xrightarrow{a}q'$, $q_i^\odot\xrightarrow{\lambda}[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q']^\odot$ by definition of $\mathcal{A}^\odot$. Since $U^\odot$ is a simulation, there are $r_1,\ldots,r_{i-1},r_{i+1},\ldots,r_n,r'\in Q$ with $r^\odot\xrightarrow{\lambda}[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a}r']^\odot$ and $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q']^\odot U^\odot[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a}r']^\odot$. Therefore, by the definition of $U^\odot$, we have $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a}q']\bar{U}[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a}r']$, and by the definition of $\mathcal{A}^\odot$, $(r_1,\ldots,r_n)\xrightarrow{a}r$.

Assume that there are two elements of *Env* such that $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]\bar{U}[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]$. By the definition of $U^\odot$, $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot U^\odot[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot$. Since $U^\odot \subseteq I_D$, $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot I_D[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot$ and hence for each j such that $1\le j\neq i\le n$, q_jDr_j. By definition of $\mathcal{A}^\odot$, the only transitions from $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot$ resp. $[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot$ are $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot\xrightarrow{a} q^\odot$ resp. $[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot\xrightarrow{a} r^\odot$. Consequently, it must be the case that $q^\odot U^\odot r^\odot$. This means that $q\bar{U}r$.

(only if) Assume that $\bar{U}$ is an extended upward simulation induced by D. We will show that $U^\odot$ is a simulation on $\mathcal{A}^\odot$ included in I_D. Suppose that $q^\odot U^\odot r^\odot$ and $q^\odot\xrightarrow{\lambda}[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q']^\odot$. Since $q^\odot U^\odot r^\odot$, we know that $q\bar{U}r$; and since $q^\odot\xrightarrow{\lambda}[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q']^\odot$ we know by definition of $\mathcal{A}^\odot$ that $(q_1,\ldots,q_n)\xrightarrow{a} q'$ where $q=q_i$. Since $\bar{U}$ is an extended upward simulation induced by D, there are $r_1,\ldots,r_n,r'\in Q$ with $(r_1,\ldots,r_n)\xrightarrow{a} r$ with $r_i=r$ and $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q']\bar{U}[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r']$. Therefore, $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q']^\odot U^\odot[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r']^\odot$.

Now, suppose that $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot U^\odot[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot$. By definition of $\mathcal{A}$, $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]^\odot\xrightarrow{a} q^\odot$ and $[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]^\odot\xrightarrow{a} r^\odot$. Moreover, $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]\bar{U}[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]$ by definition of $U^\odot$. Since $\bar{U}$ is an extended upward simulation included in D, we have (i) $q\bar{U}r$; and (ii) q_jDr_j for all $j: 1\le j\neq i\le j$. This implies that $q^\odot U^\odot r^\odot$ and $[(q_1,\ldots,\Box_i,\ldots,q_n)\xrightarrow{a} q]I_D[(r_1,\ldots,\Box_i,\ldots,r_n)\xrightarrow{a} r]$, respectively. Hence $U^\odot$ is a simulation on $\mathcal{A}^\odot$ included in I_D. □

The relation I_D is clearly a preorder. Due to Theorem 6, the maximal extended upward simulation $\preceq_U$ on $\mathcal{A}$ induced by D corresponds to the simulation preorder on $\mathcal{A}^\odot$ included in I_D which can be computed by running Algorithm 1 on the LTS $\mathcal{A}^\odot$ with the initial partition-relation pair $\langle P^\odot, Rel^\odot\rangle$ inducing I_D, i.e., $P^\odot = Q^\odot/I_D\cap I_D^{-1}$ and $Rel^\odot=\{(B,C)\in P^\odot\times P^\odot \mid B\times C\subseteq I_D\}$. By Proposition 4.2, the upward simulation preorder on $\mathcal{A}$ induced by D is then obtained as the restriction of $\preceq_U$ to Q.

4.4.4 Complexity of Computing Upward Simulation

Once the inducing preorder D on a $\mathcal{A}$ is computed, the complexity of computing the upward simulation preorder induced by D naturally consists of the price of compiling $\mathcal{A}$ into its corresponding LTS $\mathcal{A}^\odot$, the price of building the initial partition-relation pair $\langle P^\odot, Rel^\odot\rangle$, and the price of running Algorithm 1 on $\mathcal{A}^\odot$ and $\langle P^\odot, Rel^\odot\rangle$. We use $\equiv_U$ to denote the maximal equivalence included in the maximal extended upward simulation induced by D.

We assume the automaton $\mathcal{A}$ to be encoded in the format of lhs-list, this is, in the same way as in the case of computing the downward simulation (c.f. Section 4.4.2). 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* wrt. I_D, which is a subset of the partition $P^\odot$ (formally, $P_e^\odot = P^\odot\cap 2^{Env}$). If the computation of $P_e^\odot$ is done naïvely (i.e., based on comparing each environment with every other environment), it can introduce a factor of $|Env|^2$ into the overall complexity of the procedure. This would dominate the complexity of computing the simulation on $\mathcal{A}^{\odot}$ where, as we will see, $|Env|$ is only multiplied by $|Env/{\equiv_U}|$.

Fortunately, this complexity blowup can be to a large degree avoided by exploiting the partition $Lhs/{\equiv_D}$. Notice that in the case when D is the downward simulation preorder, $Lhs/{\equiv_D}$ was anyway computed when computing D. We first give the basic ideas, the detailed algorithm for computing $P_e^{\odot}$ is rather technical and is presented within the proof of Lemma 4.12.

For each $i : 1 \leq i \leq \hat{r}$, we define an i-weakened version D_i of D on left-hand sides of $\mathcal{A}$ that does not take into account states on the i-th position. Formally, $D_i = ((q_1, \ldots, q_n), (r_1, \ldots, r_m)) \in D_i \iff n = m \geq i \land (\forall 1 \leq j \leq n.\ j \neq i \implies q_j D r_j)$. Clearly, each D_i is a preorder, 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 blocks of $P_e^{\odot}$ and blocks of $Lhs/{\approx_i}$. Namely, we have that $L \in Lhs/{\approx_i}$ iff for each $a \in \Sigma$, there is a block $E_{L,a} \in P_e^{\odot}$, such that $E_{L,a} = \{[(q_1, \ldots, \Box_i, \ldots, q_n) \xrightarrow{a} q] \mid \exists q_i, q \in Q.\ (q_1, \ldots, q_i, \ldots, q_n) \in L \land (q_1, \ldots, q_i, \ldots, q_n) \xrightarrow{a} q\}$.

The idea of computing $P_e^{\odot}$ is now to first compute blocks of $Lhs/{\approx_i}$ and then to derive from them the blocks of $P_e^{\odot}$. The key advantage here is that the computation of the $\approx_i$-blocks can be done on blocks of $Lhs/{\equiv_D}$ instead of directly on elements of Lhs.[1] This is because, for each i, blocks of $Lhs/{\equiv_D}$ are sub-blocks of blocks of $Lhs/{\approx_i}$. Moreover, for any blocks K, L of $Lhs/{\equiv_D}$, the test on $K \times L \subseteq D_i$ can simply be done by choosing any two representatives $k \in K$ and $l \in L$ and testing whether $(k, l) \in D_i$. Therefore, all $\approx_i$-blocks can be computed in time $\mathcal{O}(\hat{r}|Lhs/{\equiv_D}|^2)$, as we will show within the proof of Lemma 4.12.

From each block $L \in Lhs/{\approx_i}$, one block $E_{L,a}$ of $P_e^{\odot}$ is generated for each symbol $a \in \Sigma$. The $E_{L,a}$ 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 a, and adding a right-hand side state $q \in Q$ which together with l form a transition $l \xrightarrow{a} q$ of $\mathcal{A}$. This can be done efficiently using the lhs-list encoding of $\mathcal{A}$. An additional factor $|\Delta| \log |Env|$ 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 $\log |Env|$ 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 $\langle P^{\odot}, Rel^{\odot} \rangle$ is then summarised in the below lemma.

Lemma 4.12. *Given the partition $Lhs/{\equiv_D}$, the LTS $\mathcal{A}^{\odot}$ and the partition-relation pair $\langle P^{\odot}, Rel^{\odot} \rangle$ can be derived in time $\mathcal{O}(|\Sigma||Q| + \hat{r}(|Lhs| + |Lhs/{\equiv_D}|^2) + \hat{r}^2|\Delta| \log |Env| + |P^{\odot}|^2)$ and in space $\mathcal{O}(|\Sigma||Q| + |Env| + \hat{r}|Lhs| + |Lhs/{\equiv_D}|^2 + |P^{\odot}|^2)$.*

[1] If D is the downward simulation preorder, $Lhs/{\equiv_D}$ was anyway computed within computation of D.

Proof. We assume to start with the lhs-list representation of $\mathcal{A}$. We need to derive the LTS $\mathcal{A}^{\odot}$ in the state-list format and the partition-relation pair $\langle P^{\odot}, Rel^{\odot}\rangle$. 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 $Lhs/{\equiv_D}$. In the case when D is the downward simulation preorder, $Lhs/{\equiv_D}$ is constructed within the computation of the downward simulation on $\mathcal{A}$. The state-list representation of LTS $\mathcal{A}^{\odot}$ 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. For an i-compatible subset E of Env, we define the set of their left-hand side generators as $\{(q_1, \ldots, q_n) \in Lhs \mid [(q_1, \ldots, \square_i, \ldots, q_n) \xrightarrow{a}_i q] \in E\}$.

Algorithm 2: Upward Initialisation

```
   Input: a tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$ and a partition $Lhs/{\equiv_D}$
   Data: for each $1 \le i \le \hat{r}$, a relation $Rel_i \subseteq Lhs/{\equiv_D} \times Lhs/{\equiv_D}$
   Output: the partition-relation pair $\langle P^{\odot}, Rel^{\odot}\rangle$ and the LTS
           $\mathcal{A}^{\odot} = (\Sigma^{\odot}, Q^{\odot}, \Delta^{\odot})$
1  forall $K, L \in Lhs/{\equiv_D}$ do
2      forall $1 \le i \le \hat{r}$ 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 $1 \le i \le \hat{r}$ do
6      foreach equivalence class $\{L_1, \ldots, L_m\} \in (Lhs/{\equiv_D})/(Rel_i \cap Rel_i^{-1})$ do
7          merge $L_j$s into a new block of $Lhs/{\approx_i}$, 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 $\mathcal{A}^{\odot}$ within this procedure. Then add $E$ into $P_e^{\odot}$;
9  forall $1 \le i \le \hat{r}$ 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 $\langle P^{\odot}, Rel^{\odot}\rangle \leftarrow \langle P_e^{\odot} \cup \{\{q^{\odot} \mid q \in Q\}\}, Rel^{\odot} \cup (\{q^{\odot} \mid q \in Q\}, \{q^{\odot} \mid q \in Q\})\rangle$;
```

Lines 1–3. At the first step (lines 1–3) we compute for each $1 \le i \le \hat{r}$ a binary relations Rel_i on blocks of $Lhs/{\equiv_D}$ such that the partition-relation pair $(Lhs/{\equiv_D}, Rel_i)$ 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 $Lhs/{\equiv_D}$, 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 ($(k, l) \in \equiv_D$)

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 $K, L \in Lhs/{\equiv_D}$ with respect the $K \times L \subseteq D_i$ inclusions.

From these properties follows that given two blocks $K, L \in Lhs/{\equiv_D}$, the tests $K \times L \subseteq D_i$ can be done for all i in time $\mathcal{O}(\hat{r})$ 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, line 3 can be done in (amortised) constant time and thus the for loop on lines 1–3 can be finished in time $\mathcal{O}(\hat{r}|Lhs/{\equiv_D}|^2)$. Furthermore, encoding of all the Rel_i relations takes only $\mathcal{O}(|Lhs/{\equiv_D}|^2)$ space.

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

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

On line 8 we generate all the environments of E and update $\mathcal{A}^\odot$. 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 $\mathcal{A}$, i.e. each l is connected to an array indexed by symbols from Σ where the a-indexed element contains the list of all states q such that $l \xrightarrow{a} q$. Thus for each $l \in B$, we can effectively iterate through all rules of the form $l \xrightarrow{a} q$ and for each of them we: (1.) create a new environment; and (2.) update $\mathcal{A}^\odot$ in the following way:

(1.) We create a representation of environment e consisting of a pointer on l, symbol a, 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 e created, we test whether 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) $\hat{r}|\Delta|$ times. In the end (when $S = Env$), we get $|Env|$ different environments. We can assume that testing equality of two environments takes $\mathcal{O}(\hat{r})$ time and that we use a set representation with a logarithmic membership test and addition. Thus, in total, the time $\mathcal{O}(\hat{r}^2|\Delta| \log |Env|)$ is spent by testing membership of environments in S and by extending S by the new environments.

(2.) Having a representation of an environment $e = [(q_1, \dots, \Box_i, \dots, q_n) \xrightarrow{a} q]$ 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 $pre_a(q)$. Then, regardless on the result of the $e \in S$ test, 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 $\mathcal{A}^{\odot}$. 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.) Since creating a state $e^{\odot}$ and adding an element into a *pre* set are constant time, the overall complexity of these updates of $\mathcal{A}^{\odot}$ is covered by the complexity of the above procedure for creating the environments in the E blocks.

Lines 9–10. On lines 9-10 we compute the main part of relation $Rel^{\odot}$. We exploit 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 $L, K \in Lhs/{\equiv_D}$ 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 its 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 $\mathcal{O}(|P_e^{\odot}|^2)$.

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

We instantiate the complexity of running Algorithm 1 for $\mathcal{A}^{\odot}$ and $\langle P^{\odot}, Rel^{\odot} \rangle$ within the following theorem.

Lemma 4.13. *Algorithm 1 with input $\mathcal{A}^{\odot}$ and $\langle P^{\odot}, Rel^{\odot} \rangle$ terminates in time $\mathcal{O}(\hat{r}|\Delta||Env/{\equiv_U}| + |\Sigma||Env||Env/{\equiv_U}|)$ and space $\mathcal{O}(|\Sigma||Env||Env/{\equiv_U}|)$.*

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

$$\mathcal{O}(|\Sigma|(|Q| + |Env|)(|Q/{\equiv_U}| + |Env/{\equiv_U}|) + \hat{r}|\Delta|(|Q/{\equiv_U}| + |Env/{\equiv_U}|)).$$

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

$$\mathcal{O}(|\Sigma||Env||Env/{\equiv_U}| + \hat{r}|\Delta||Env/{\equiv_U}|)$$

and, as the space complexity in Theorem 2 equals the first summand of the time complexity formula, we get the space complexity $\mathcal{O}(|\Sigma||Env||Env/{\equiv_U}|)$. $\Box$

The complexity of computing upward simulation preorder on $\mathcal{A}$ induced by D can now be obtained by simply summing the price of computing D and $Lhs/{\equiv_D}$, the price of computing $\mathcal{A}^\odot$ and $\langle P^\odot, Rel^\odot\rangle$, and the price of running Algorithm 1 on $\mathcal{A}^\odot$ and $\langle P^\odot, Rel^\odot\rangle$.

Theorem 7. *Let $T_D(\mathcal{A})$ and $S_D(\mathcal{A})$ denote the time and space needed for computing the preorder D and on $\mathcal{A}$ and $Lhs/{\equiv_D}$. Then, the upward simulation preorder on $\mathcal{A}$ induced by D can be computed in time*

$$\mathcal{O}((|\Sigma||Env| + \hat{r}|\Delta|)|Env/{\equiv_U}| + \hat{r}^2|\Delta|\log|Env| + T_D(\mathcal{A}))$$

and in space $\mathcal{O}(|\Sigma||Env||Env/{\equiv_U}| + S_D(\mathcal{A}))$.

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

4.4.5 Computing Downward Bisimulation Equivalences

In [HMM07a], Högberg, Maletti, and May propose an algorithm for computing downward bisimulation with running time $\mathcal{O}(\hat{r}^2|\Delta|\log(|Q|))$ (in [HMM07a], downward bisimulation is called backward bisimulation). Our approach based on translating tree automata to LTS that we use for simulations can be also used and yields an algorithm with the same asymptotic complexity. In particular, we use the same LTS $\mathcal{A}^\bullet$ as for downward simulation. Downward bisimulation equivalence is then obtained in the form of the standard LTS bisimulation equivalence on states of $\mathcal{A}^\bullet$. This can easily be proved using the results of Section 4.4.1 and the fact that downward bisimulations are exactly downward simulations such that their inverses are also downward simulations. For this we need to extend the definition of downward bisimulation to left-hand sides of $\mathcal{A}$ analogically as we have extended the definition of downward simulation.

Extended downward bisimulation. An *Extended downward bisimulation on $\mathcal{A}$* is any extended downward simulation on $\mathcal{A}$ such that its inverse is also an extended downward simulation on $\mathcal{A}$.

Proposition 4.3. *A relation D is a downward bisimulation if and only if there is an extended downward bisimulation $\bar{D}$ such that its restriction to Q is D.*

Proof. Let D be the restriction of an extended downward bisimulation $\bar{D}$ to Q. We know that both $\bar{D}$ and $\bar{D}^{-1}$ are extended downward simulations, therefore both D and D^{-1} are downward simulations by Proposition 4.1, and thus D is a downward bisimulation by definition.

Let D be a downward bisimulation on $\mathcal{A}$ and let us define $\bar{D}$ as the relation $D \cup \{((q_1,\dots,q_n),(r_1,\dots,r_n)) \in Lhs \times Lhs \mid \forall 1 \leq i \leq n : q_i D r_i\}$. Since D is a simulation, $\bar{D}$ is apparently an extended downward simulation. Then, define analogically $\bar{D}' = D^{-1} \cup \{((q_1,\dots,q_n),(r_1,\dots,r_n)) \in Lhs \times Lhs \mid \forall 1 \leq i \leq n : q_i D^{-1} r_i\}$. Again, since D^{-1} is a downward simulation, $\bar{D}'$ is and extended downward simulation. Now, it is not hard to see that $\bar{D}^{-1} = \bar{D}'$ and therefore, as both $\bar{D}$ and $\bar{D}^{-1}$ are extended downward simulations, $\bar{D}$ is an extended downward bisimulation on $\mathcal{A}$. $\square$

Theorem 8. *A relation $\bar{D}$ is an extended downward bisimulation on $\mathcal{A}$ if and only if $D^\bullet = \{(x^\bullet, y^\bullet) \mid x\bar{D}y\}$ is a bisimulation on $\mathcal{A}^\bullet$.*

Proof. $\bar{D}$ is an extended downward bisimulation on $\mathcal{A}$ iff both $\bar{D}$ and $\bar{D}^{-1}$ are extended downward simulations on $\mathcal{A}$ which holds (by Theorem 4) iff both $D^\bullet$ and $(D^\bullet)^{-1}$ are simulations on $\mathcal{A}^\bullet$, and by the definition of bisimulation, this holds iff $D^\bullet$ is a bisimulation on $\mathcal{A}^\bullet$. $\square$

To compute the bisimulation equivalence on $\mathcal{A}^\bullet$, we can use the algorithm recently proposed by Valmari in [Val09] that on $\mathcal{A}^\bullet$ runs in time $\mathcal{O}(|\Delta^\bullet| \log(Q^\bullet))$. The sizes of the parameters of $\mathcal{A}^\bullet$ can be bounded as follows: $|Q^\bullet| \in \mathcal{O}(Lhs) \subseteq \mathcal{O}(|Q|^{\hat{r}})$, and $|\Delta^\bullet| \leq |\Delta| + \hat{r}|Lhs| \leq \hat{r}|\Delta|$. Therefore, the time complexity of running the Valmari's algorithm on $\mathcal{A}^\bullet$ is $\mathcal{O}(\hat{r}|\Delta| \log(|Q|^{\hat{r}})) = \mathcal{O}(\hat{r}^2|\Delta| \log(|Q|))$, which is indeed the same complexity as the one of the algorithm from [HMM07a].

4.4.6 Computing Upward Bisimulation Equivalences

Let us fix a preorder D on Q. Our algorithm for computing the upward bisimulation equivalence induced by D is again based on translating tree automata into labelled transition systems. The same transition system $\mathcal{A}^\odot$ as for upward simulations can be used. To prove this easily using the results of Section 4.4.3, we first extend upward bisimulation to environments of $\mathcal{A}$.

Extended Upward Bisimulation. An *extended upward bisimulation U on $\mathcal{A}$ induced by D* is an extended upward simulation induced by $D \cap D^{-1}$ such that its inverse is also an extended upward simulation on $\mathcal{A}$ induced by $D \cap D^{-1}$.

Proposition 4.4. *A relation U is an upward bisimulation induced by D iff there is an extended upward bisimulation on $\mathcal{A}$ induced by D such that its restriction to Q is U.*

Proof. Let U be the restriction of an extended upward bisimulation $\bar{U}$ induced by D to Q. We know that both $\bar{U}$ and $\bar{U}^{-1}$ are extended upward simulations induced by $D \cap D^{-1}$, therefore both U and U^{-1} are upward simulations induced by D by Proposition 4.2, and hence, by definition, U is an upward bisimulation induced by D.

Let U be an upward bisimulation on $\mathcal{A}$ induced by D. Define $\bar{U} = U \cup \{([(q_1, \ldots, \square_i, \ldots, q_n) \xrightarrow{a} q], [(r_1, \ldots, \square_i, \ldots, r_n) \xrightarrow{a} r]) \in \mathit{Env} \times \mathit{Env} \mid \forall j : 1 \leq j \neq i \leq n.\ q_j D \cap D^{-1} r_j\}$. Since U is an upward simulation induced by $D \cap D^{-1}$, $\bar{U}$ is apparently an extended upward simulation induced by $D \cap D^{-1}$. Then, define analogically $\bar{U} = U^{-1} \cup \{([(q_1, \ldots, \square_i, \ldots, q_n) \xrightarrow{a} q], [(r_1, \ldots, \square_i, \ldots, r_n) \xrightarrow{a} r]) \in \mathit{Env} \times \mathit{Env} \mid \forall j : 1 \leq j \neq i \leq n.\ q_j D \cap D^{-1} r_j\}$. Again, since U^{-1} is an upward simulation induced by $D \cap D^{-1}$, $\bar{U}'$ is an extended upward simulation induced by $D \cap D^{-1}$. Now, it is not hard to see that $\bar{U}^{-1} = \bar{U}'$ and therefore, as both $\bar{U}$ and $\bar{U}^{-1}$ are extended upward simulations induced by $D \cap D^{-1}$, $\bar{U}$ is an extended upward bisimulation on $\mathcal{A}$ induced by D. $\square$

We define $I_{\equiv_D} = I_D \cap I_D$, this is, $I_{\equiv_D}$ is the smallest binary relation on $Q^\odot$ containing all pairs of states of the automaton $\mathcal{A}$, 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 $([(q_1, \dots, \Box_i, \dots, q_n) \xrightarrow{a} q]^\odot, [(r_1, \dots, \Box_i, \dots, r_n) \xrightarrow{a} r]^\odot)$ such that $q_j D \cap D^{-1} r_j$ for each $j : 1 \leq j \neq i \leq n$.

Theorem 9. *A relation $\bar{U}$ is an extended upward bisimulation on $\mathcal{A}$ induced by D if and only if $U^\odot = \{(x^\odot, y^\odot) \mid x\bar{U}y\}$ is a bisimulation on $\mathcal{A}^\odot$ included in $I_{\equiv_D}$.*

Proof. $\bar{U}$ is an extended upward bisimulation on $\mathcal{A}$ iff both $\bar{U}$ and $\bar{U}^{-1}$ are extended upward simulations on $\mathcal{A}$ induced by $D \cap D^{-1}$ which holds (by Theorem 9) iff both $U^\odot$ and $(U^\odot)^{-1}$ are simulations on $\mathcal{A}^\odot$ included in $I_{\equiv_D}$, and by definition of bisimulation, this holds iff $U^\odot$ is a bisimulation on $\mathcal{A}^\odot$ included in $I_{\equiv_D}$. □

The Valmaris algorithm [Val09] computes the bisimulation equivalence included in $I_{\equiv_D}$ on $\mathcal{A}$ in time $\mathcal{O}(|\Delta^\odot| \log(|Q^\odot|))$. Since $|Q^\odot| \in \mathcal{O}(|Q| + \hat{r}|\Delta|) = \mathcal{O}(\hat{r}|\Delta|)$ and $|\Delta^\odot| \in \mathcal{O}(\hat{r}|\Delta|)$, the running time of Valmaris algorithm on $\mathcal{A}^\odot$ amounts to $\mathcal{O}(\hat{r}|\Delta| \log(\hat{r}|\Delta|)) \subseteq \mathcal{O}(\hat{r}|\Delta| \log(\hat{r}|Q|^{\hat{r}}|\Sigma|)) = \mathcal{O}(\hat{r}^2 |\Delta| \log|Q| + \hat{r}|\Delta| \log|\Sigma|)$.

We note that in [HMM07a] is presented a specialised algorithm for computing upward bisimulation equivalence induced by identity (called forward bisimulation in [HMM07a]). The algorithm runs in time $\mathcal{O}(\hat{r}|\Delta| \log(|Q|))$, which is better than the complexity of our algorithm (however, the algorithm from [HMM07a] is not designed for computing bisimulations induced by nontrivial preorders). Still, it suggests that there might be a space for improving our method.

4.4.7 Computing the Combined Relations

Given an inducing downward simulation D and an upward simulation U induced by D, the combined preorder $M = D \oplus U^{-1}$ can be easily computed by simply following its definition. It is sufficient to start by computing the relation $C = D \circ U^{-1}$ and then just erase all the elements of C that break Condition (ii) from the definition of $\oplus$. Using suitable data structures, this computation starting from the relations U and D can be implemented to run in time $\mathcal{O}(\min\{|D||Q|, |U||Q|\})$ as follows.

We encode a relation ρ on Q as an array indexed by elements of Q of lists of elements of Q. A state q is present in a list with index r iff $(r, q) \in \rho$. Note that given a Boolean matrix representation of the relation, the "array of lists"-representation can be derived in time $\mathcal{O}(|Q|^2)$. Note also that as U and D are reflexive, we have that $|U|, |D| \geq |Q|$ and thus $|Q|^2 \leq \min\{|D||Q|, |U||Q|\}$. Let arrays of lists $\mathsf{D}, \mathsf{U}^{-1}$ encode relations D, U^{-1}.

The relation $C = D \circ U^{-1}$ represented by a Boolean matrix C can be computed in the following way: (1) Initialise all entries of C to *false*. (2) For each $q \in Q$, pass through all elements of the list $\mathsf{D}[q]$, and for each $r \in \mathsf{D}[q]$, pass through all elements s of $\mathsf{U}^{-1}[r]$, and set $\mathsf{C}[q, s]$ to *true*. This procedure takes time $\mathcal{O}(|\{(q, r, s) \mid (q, r) \in D \wedge (r, s) \in U^{-1}\}|) \subseteq \mathcal{O}(\min\{|D||Q|, |U||Q|\})$.

Then we compute a Boolean matrix representation M of the relation $M = D \oplus U^{-1}$ as follows: (3) We initialise M as a copy of the matrix C (representing $D \circ U^{-1}$), and in the subsequent Step (4), we erase from M all the pairs of elements of Q that break Condition (ii) from the definition of $\oplus$. In Step (4), we proceed in the following way: For all $q \in Q$, for all $r \in \mathsf{D}[q]$, for all $s \in \mathsf{U}^{-1}[r]$, if not $\mathsf{C}[q, s]$ (i.e., $(q, s) \notin D \circ U^{-1}$), then $\mathsf{M}[q, s] = \mathit{false}$. This gives us the set $D \oplus U^{-1}$ represented by the matrix M. The complexity of Steps (3), (4) is in $\mathcal{O}(|\{(q, r, s) \mid (q, r) \in D \wedge (r, s) \in U^{-1}\}| + |\{(q, r, s) \mid (q, r) \in U^{-1} \wedge (r, s) \in D\}|)$, which is again in $\mathcal{O}(\min\{|D||Q|, |U||Q|\})$.

4.5 Experiments

We have implemented our algorithms in a prototype tool written in Java. We have used the tool on a number of tree automata from the frameworks of *regular tree model checking* (RTMC) and *abstract regular tree model checking* (ARTMC) [BT02, AJMd02, BHRV06a, BHRV06a]. These techniques were shortly discussed in Chapter 1 and we will explain them in a more detail in Chapter 5. Most of the algorithms in the frameworks of both RTMC and ARTMC rely crucially on efficient automata reduction methods since the size of the generated automata often explodes, making computations infeasible without a reduction.

Our experimental evaluation was carried out on an AMD Athlon 64 X2 2.19GHz PC with 2.0 GB RAM. We have compared the size of tree automata after reducing them with all the different reduction techniques considered in this thesis. Table 4.1 shows the computation time and the reduction (in percent) for the different relations within the considered framework and illustrates that we have really obtained a wide spectrum of relations differing in their reduction capabilities and computational complexity. As can be seen from the results, $\overset{\circ}{\preceq}$ gives the best reduction in all experiments, but it also suffers from a high computation time. Combining simulations and bisimulations does not give the same amount of reduction as the combined simulation, but the computation time is lower and the reduction is better than $\overset{\bullet}{\simeq}$. Note that no attempt to optimise the implementation of any of the relations was done, and therefore the computation times could probably be much lower with an optimised implementation for all of them.

Another set of experiments proving significance of the mediated equivalence $\overset{\circ}{\preceq}$ in practice was done in [BHH+08b]. We have implemented our reduction methods within an ARTMC-based verification tool which was tested on various benchmarks, mostly short but complex pointer manipulating programs. Together with a new method for testing language inclusion of nondeterministic tree automata presented in [BHH+08b], quotienting allowed us to greatly improve performance of the ARTMC tool. We comment more on this set of experiments in Chapter 5.

Table 4.1: The obtained reduction in percent and the computation time in seconds for the various considered relations applied for reducing TA obtained from RTMC and ARTMC case studies. The size of the TA is the number of their states plus the number of their transition rules.

TA		$\preceq$		$\overset{\circ}{=}$		$\overset{\circ}{\simeq}$		$\overset{\circ}{\preceq}$	
origin	size	red.	time	red.	time	red.	time	red.	time
ARTMC	195	18%	0.5s	2%	0.5s	23%	0.5s	61%	1.0s
RTMC	613	27%	3.5s	19%	2.0s	19 %	2.5s	88%	5.1s
RTMC	909	52%	3.6s	72%	3.1s	82%	3.4s	89%	35.1s
ARTMC	2029	10%	27.0s	37%	26.0s	33%	29.0s	93%	39.0s
RTMC	2403	26%	31.0s	0%	25.0s	0%	34.0s	82%	37.1s

TA		$\simeq$		$\overset{\bullet}{=}$		$\overset{\bullet}{\simeq}$		$\overset{\bullet}{\preceq}$	
origin	size	red.	time	red.	time	red.	time	red.	time
ARTMC	195	18%	0.1s	2%	0.5s	23%	0.2s	23%	0.6s
RTMC	613	0%	0.3s	0%	0.4s	0%	0.8s	27%	3.7s
RTMC	909	14%	0.6s	72%	0.4s	82%	0.8s	83%	4.1s
ARTMC	2029	10%	1.7s	14%	1.4s	19%	3.1s	44%	29.0s
RTMC	2403	0%	0.3s	0%	0.6s	0%	0.7s	27%	31.0s

4.6 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 an 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, the so called mediated equivalence, derived from compositions of downward and upward simulation. Mediated equivalences according to our experiments usually give a much better reduction of the size of automata than downward or upward simulations alone.

We have also considered upward and downward bisimulations on tree automata, that may be seen as special cases of tree automata simulations. Bisimulations are much stronger relations than simulations, therefore, quotienting using bisimulations reduces automata less. On the other hand, bisimulations are considerably computationally cheaper. We show that our approach for computing tree automata simulations via translations to labelled transition systems can be easily used also for computing tree automata bisimulations. Particularly, we translate downward or upward bisimulation problems in the same way as downward or upward simulation problems, respectively, and then run a standard LTS bisimulation algorithm on the resulting LTS. This uniform framework yields tree automata simulation and bisimulation algorithms that are efficient and can be implemented with a relatively small effort.

Moreover, our combination operator can be used to combine any downward simulation or bisimulation with any induced upward simulation or bisimulation, which yields a spectrum of mediated equivalences. We have established a partial ordering of the obtained mediated equivalences according to their reduction capabilities and showed that some of them are also incomparable. Moreover, we have performed a number of experiments with automata from the area of (abstract) regular tree model checking that show a practical applicability of the obtained relations and allow us to conclude that the considered relations really offer a fine choice of balance in the trade-off between reduction capabilities and computational requirements.

There are several possible directions of future work. Since the proposed framework is built on quite general principles, we believe that it can be extended to more advanced types of automata such as guided tree automata, nested word automata, or hedge automata that find their use in many applications in formal verification, decision procedures of various logics, structured document processing, or natural language processing. Reduction of automata from some of such classes has already been considered in the literature (e.g., in [Buc08], the author proposes a bisimulation-based minimisation of weighted word automata, and a use of bisimulations for reducing weighted tree automata is considered in [HMM07b]). In Chapter 6, we present a nontrivial extension of our framework to alternating Büchi automata. From the practical point of view, it is also interesting to investigate more efficient techniques of computing the (bi-)simulation relations, e.g., by computing them in a symbolic way (for symbolically encoded automata). Furthermore, it can be interesting to explore more deeply the principles of the proposed combination of downward and upward (bi-)simulation relations. One can, for instance, think of defining still weaker types of relations preserving the language of tree automata by using the combined relations repeatedly as inducing relations.

5 Language Inclusion and Universality of Finite (Tree) Automata

The language inclusion problem for regular languages is important in many application domains, e.g., formal verification. Many verification problems can be formulated as a language inclusion problem. For example, one may describe the actual behaviours of an implementation in an automaton $\mathcal{A}$ and all of the behaviours permitted by the specification in another automaton $\mathcal{B}$. Then, the problem of whether the implementation meets the specification is equivalent to the problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Other applications include checking whether a fixpoint of a symbolic automata-based incremental reachability computation was reached or checking implication in automata-based decision procedures. The universality problem is a simpler variant of the language inclusion problem. Even though it is less useful in practice, it is important from the theoretical point of view. A good solution for the universality problem often leads to a good solution for language inclusion problem while the simpler setting of the former problem makes the principles of the method easier to master.

Methods for proving language inclusion can be categorised into two types: those based on *simulation* (e.g., [DHWT91]) and those based on the *subset construction* (e.g., [Brz62, Hop71, MS72, Møl04]). Simulation-based approaches first compute a simulation relation on the states of two automata $\mathcal{A}$ and $\mathcal{B}$ and then check whether all initial states of $\mathcal{A}$ can be simulated by some initial state of $\mathcal{B}$. Since simulation can be computed in polynomial time, simulation-based methods are usually very efficient. Their main drawback is that they are incomplete since simulation implies language inclusion, but not vice-versa.

On the other hand, methods based on the subset construction are complete but inefficient because in many cases they will cause an exponential blow up in the number of states. Recently, De Wulf et al. in [WDHR06] proposed the *antichain-based* approach for nondeterministic finite word automata. To the best of our knowledge, it was the most efficient one among all of the methods based on the subset construction. Although the antichain-based method significantly outperforms the classical subset construction, in many cases, it (unavoidably) still sometimes suffers from the exponential blow up problem.

This chapter presents result that were published in two works, [BHH$^+$08b] and [ACH$^+$10a]. In [BHH$^+$08b], we generalise the results on FA from [WDHR06] also for tree automata and we show how a combination of the antichain-based tree automata inclusion checking with the reduction techniques from Chapter 4 allows to greatly improve efficiency of abstract regular tree model checking method. In [ACH$^+$10a], we present a new approach for both word and tree automata universality and inclusion checking that nicely combines the simulation-based and the antichain-based approaches. A computed simulation relation is used for pruning out unnecessary search paths of the antichain-based method and also to efficiently encode the stored state-space. To distinguish the approaches from [WDHR06, BHH$^+$08b] from the one of [ACH$^+$10a], we will refer to the former ones as to the *pure antichain approach* and to the latter ones as to the *simulation-enhanced antichain approach.* In this chapter, we describe mostly the results from [ACH$^+$10a], this is, the simulation enhanced antichain algorithms for FA and TA since the pure antichain TA algorithms that we present in [BHH$^+$08b] can be seen as they simpler instances. As for experimental results, we present both the results from [BHH$^+$08b] and from [ACH$^+$10a].

To simplify the presentation, we first consider the problem of checking universality for a word automaton $\mathcal{A}$. In a similar manner to the classical subset construction, we start from the set of initial states and search for sets of states (here referred to as *macro-states*) which are not accepting (i.e., we search for a counterexample of universality). The key idea is to define an "easy-to-check" ordering $\preceq$ on the states of $\mathcal{A}$ which implies language inclusion (i.e., $p \preceq q$ implies that the language of the state p is included in the language of the state q). From $\preceq$, we derive an ordering on macro-states which we use in two ways to optimise the subset construction: (1) searching from a macro-state needs not continue in case a smaller macro-state has already been analysed; and (2) a given macro-state is represented by (the subset of) its maximal elements. In this work, we take the ordering $\preceq$ to be the simulation preorder on the automaton $\mathcal{A}$. In fact, the antichain algorithms of [WDHR06] coincide with the special case where the ordering $\preceq$ is the identity relation. Subsequently, we describe how to generalise the above approach to the case of checking language inclusion between two automata $\mathcal{A}$ and $\mathcal{B}$, by extending the ordering to pairs consisting of a state of $\mathcal{A}$ and a macro-state of $\mathcal{B}$.

We then generalise our algorithms to the case of tree automata. We first formally define a notion of a language accepted from tuples of states of the tree automaton as a set of contexts. We identify here a new application of the upward simulation relation from Chapter 4. We show that it implies (context) language inclusion, and we describe how we can use it to optimise existing algorithms for checking the universality and language inclusion properties.

We have implemented our algorithms and carried out an extensive experimentation. Particularly, in [BHH$^+$08b], we compare performance of the classical tree automata subset construction based algorithms with the pure antichain-based algorithms (so far not using simulation optimisations) developed in the spirit of [WDHR06]. We have tested the algorithms on tree automata generated with a scale of different settings of a random automata generator designed according to framework by Tabakov and Vardi [TV05]. We have also considered tree-automata derived from intermediate steps of abstract regular tree model checking. The obtained results are consistent with the ones from [WDHR06] on FA and lead to a definite conclusion that the antichain tree automata algorithms vastly outperform the classical ones. Our inclusion checking algorithms together with the reduction techniques from Chapter 4 also greatly improve the overall performance of the abstract regular tree model checking method.

In [ACH$^+$10a], we have carried out experiments comparing the pure antichain-based algorithms for both FA and TA with their simulation-improved variants. In the case of FA, we obtained our experimental data from several different sources. The experiments show that simulation enhanced antichain approach significantly outperforms the pure antichain-based approach in almost all of the considered cases.

We note that simultaneously with [ACH$^+$10a], Doyen and Raskin published their recent work [DR10] where they present basically the same main idea as is the one of [ACH$^+$10a] (this is, using simulation to improve the antichain algorithms). However, even though the two works have significant overlaps, both of them contain original unique contributions. We will briefly compare the two works in the following two paragraphs.

Doyen and Raskin in [DR10] study more systematically theoretical aspects of simulation optimisations of antichain algorithms. They present a framework where they consider also the backward algorithms for FA that were presented in [WDHR06] and show how they can be optimised with backward simulation. These backward algorithms are dual to the forward ones and they utilise backward simulation instead of forward simulation. They also consider a conceptually different approach where one utilises forward simulation within backward algorithms and backward simulation within forward algorithms. Apart from that, Doyen and Raskin also show other applications of their framework to problems such as emptiness of alternating automata.

On the other hand, our paper [ACH$^+$10a] comes with the following. Contrary to [DR10], we provide extensive experimental results showing practical applicability of the algorithms. We also design algorithms that are carefully optimised not to explore unnecessary search paths which also notably improves their efficiency. Then, except using simulations to prune unpromising macro-states, we use them also to reduce the internal representations of reached macro-states. We study in detail both universality and language inclusion problem (while Raskin and Doyen concentrate mostly only on universality) where not all the optimisations that we propose are covered by the framework from [DR10] (in particular, in the case of inclusion checking, we utilise also simulation between states of the two automata). Finally, we also present an extension of the technique to tree automata.

Outline. The remainder of the chapter is organised as follows. We begin Section 5.1 by applying our idea to solve the universality problem for FA. The problem is simpler than the language inclusion problem and thus we believe that presenting our universality checking algorithm first makes it easier for the reader to grasp the idea. We continue the section by discussing our language inclusion checking algorithm for FA. In Section 5.2, we present the algorithms for checking universality and language inclusion for tree automata that are extensions of the FA algorithms from Section 5.1. Section 5.3 describes experimental results on comparing pure antichain-based algorithms for TA with the classical subset construction-based algorithms, and also experiments on testing impact of applying our algorithms in abstract regular tree model checking. In Section 5.4, we present experiments on comparing pure antichain-based algorithms for both FA and TA with their versions improved with simulations. Finally, in Section 5.5, we conclude the chapter and discuss further research directions.

5.1 Universality and Language Inclusion of FA

In this section, we describe our simulation improvements of the antichain algorithms for testing language inclusion and universality of FA from [WDHR06]. Basically, we will show how to utilise simulation on states of an automaton (that is computed in advance) within a language inclusion/universality checking algorithm.

Let $\mathcal{A} = (\Sigma, Q, \delta, I, F,)$ be a finite automaton. For convenience, we call a set of states in $\mathcal{A}$ a *macro-state*, i.e., a macro-state is a subset of Q. A macro-state is *accepting* if it contains at least one accepting state, otherwise it is *rejecting*. For a macro-state P, define $L(\mathcal{A})(P) := \bigcup_{p \in P} L(\mathcal{A})(p)$. We say that a macro-state P is universal if $L(\mathcal{A})(P) = \Sigma^*$. For two macro-states P and R, we write $P \preceq^{\forall\exists} R$ as a shorthand for $\forall p \in P . \exists r \in R : p \preceq r$. We define the post-image of a macro-state $Post(P) := \{P' \mid \exists a \in \Sigma : P' = \{p' \mid \exists p \in P : (p, a, p') \in \delta\}\}$. We use $\mathcal{A}^{\subseteq}$ to denote the set of relations over the states of $\mathcal{A}$ that imply language inclusion, i.e., if $\preceq \in \mathcal{A}^{\subseteq}$, then we have $p \preceq r \implies L(\mathcal{A})(p) \subseteq L(\mathcal{A})(r)$.

Let $\mathcal{A} = (\Sigma, Q_\mathcal{A}, \delta_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (\Sigma, Q_\mathcal{B}, \delta_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B})$ be two FA. Define their union automaton $\mathcal{A} \cup \mathcal{B} := (\Sigma, Q_\mathcal{A} \cup Q_\mathcal{B}, \delta_\mathcal{A} \cup \delta_\mathcal{B}, I_\mathcal{A} \cup I_\mathcal{B}, F_\mathcal{A} \cup F_\mathcal{B})$.

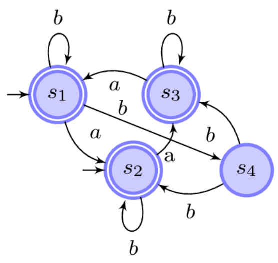
5.1.1 Universality of FA

The *universality problem* for an FA $\mathcal{A} = (\Sigma, Q, \delta, I, F)$ is to decide whether $L(\mathcal{A}) = \Sigma^*$. The problem is PSPACE-complete. The classical algorithm for the problem first determinises $\mathcal{A}$ with the subset construction and then checks if every reachable macro-state is accepting. The algorithm is inefficient since in many cases the determinisation will cause a very fast growth in the number of states. Note that for universality checking, we can stop the subset construction immediately and conclude that $\mathcal{A}$ is not universal whenever a rejecting macro-state is encountered. An example of a run of this algorithm is given in Fig. 5.1. The automaton $\mathcal{A}$ used in Fig. 5.1 is universal because all reachable macro-states are accepting.

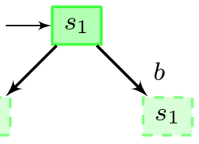
In this section, we propose a more efficient approach to universality checking. In a similar manner to the classical algorithm, we run the subset construction procedure and check if any rejecting macro-state is reachable. However, our algorithm augments the subset construction with two optimisations, henceforth referred to as *Optimisation 1* and *Optimisation 2*, respectively.

Optimisation 1 is based on the fact that if the algorithm encounters a macro-state R whose language is a superset of the language of a visited macro-state P, then there is no need to continue the search from R. The intuition behind this is that if a word is not accepted from R, then it is also not accepted from P. For instance, in Fig. 5.1(b), the search needs not continue from the macro-state $\{s_2, s_3\}$ since its language is a superset of the language of the initial macro-state $\{s_1, s_2\}$. However, in general it is difficult to check if $L(\mathcal{A})(P) \subseteq L(\mathcal{A})(R)$ before the resulting deterministic FA is completely built. Therefore, we suggest to use an easy-to-compute alternative based on the following lemma.

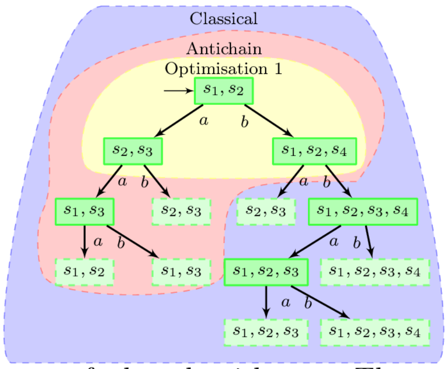
Lemma 5.1. *Let P, R be two macro-states, $\mathcal{A}$ be an FA, and $\preceq$ be a relation in $\mathcal{A}^{\subseteq}$. Then, $P \preceq^{\forall\exists} R$ implies $L(\mathcal{A})(P) \subseteq L(\mathcal{A})(R)$.*



(a) Source FA $\mathcal{A}$



(c) Optimisation 1 and 2



(b) A run of the algorithms. The areas labelled “Optimisation 1”, “Antichain”, “Classical” are the macro-states generated by our simulation enhanced antichain approach with the maximal simulation and Optimisation 1, the antichain-based approach, and the classical approach, respectively.

Figure 5.1: Universality Checking Algorithms

Note that in Lemma 5.1, $\preceq$ can be any relation on the states of $\mathcal{A}$ that implies language inclusion. This includes any simulation relation (Lemma 2.1). When $\preceq$ is the maximal simulation or the identity relation, it can be efficiently obtained from $\mathcal{A}$ before the subset construction algorithm is triggered and used to prune out unnecessary search paths.

An example of how the described optimisation can help is given in Fig. 5.1(b). If $\preceq$ is the identity, the universality checking algorithm will not continue the search from the macro-state $\{s_1, s_2, s_4\}$ because it is a superset of the initial macro-state. In fact, the pure antichain-based approach [WDHR06] can be viewed as a special case of our simulation enhanced antichain approach when $\preceq$ is the identity. Notice that, in this case, only 7 macro-states are generated (the classical algorithm generates 13 macro-states). When $\preceq$ is the maximal simulation, we do not need to continue from the macro-state $\{s_2, s_3\}$ either because $s_1 \preceq s_3$ and hence $\{s_1, s_2\} \preceq^{\forall\exists} \{s_2, s_3\}$. In this case, only 3 macro-states are generated. As we can see from the example, a better reduction of the number of generated states can be achieved when a weaker relation (e.g., the maximal simulation) is used.

Optimisation 2 is based on the observation that $\mathcal{L}(\mathcal{A})(P) = \mathcal{L}(\mathcal{A})(P \setminus \{p_1\})$ if there is some $p_2 \in P$ with $p_1 \preceq p_2$. This fact is a simple consequence of Lemma 5.1 (note that $P \preceq^{\forall\exists} P \setminus \{p_1\}$). Since the two macro-states P and $P \setminus \{p_1\}$ have the same language, if a word is not accepted from P, it is not accepted from $P \setminus \{p_1\}$ either. On the other hand, if all words in Σ^* can be accepted from P, then they can also be accepted from $P \setminus \{p_1\}$. Therefore, it is safe to replace the macro-state P with $P \setminus \{p_1\}$.

Consider the example in Fig. 5.1. If $\preceq$ is the maximal simulation relation, we can remove the state s_2 from the initial macro-state $\{s_1, s_2\}$ without changing its language, because $s_2 \preceq s_1$. This change will propagate to all the searching

Algorithm 3: *Universality Checking*

```
Input: An FA $\mathcal{A} = (\Sigma, Q, \delta, I, F)$ and a relation $\preceq \in \mathcal{A}^{\subseteq}$.
Output: TRUE if $\mathcal{A}$ is universal. Otherwise, FALSE.
1  if $I$ is rejecting then return FALSE;
2  $Processed$:=$\emptyset$;
3  $Next$:=$\{Minimize(I)\}$;
4  while $Next \neq \emptyset$ do
5      Pick and remove a macro-state $R$ from $Next$ and move it to $Processed$;
6      foreach $P \in \{Minimize(R') \mid R' \in Post(R)\}$ do
7          if $P$ is rejecting then return FALSE;
8          else if $\neg\exists S \in Processed \cup Next$ s.t. $S \preceq^{\forall\exists} P$ then
9              Remove all $S$ from $Processed \cup Next$ s.t. $P \preceq^{\forall\exists} S$;
10             Add $P$ to $Next$;
11 return TRUE
```

paths. With this optimisation, our approach will only generates 3 macro-states, all of which are singletons. The result after apply the two optimisations are applied is shown in Fig. 5.1(c).

Algorithm 3 describes our approach in pseudocode. In this algorithm, the function *Minimize*(R) implements Optimisation 2. The function does the following: it chooses a new state r_1 from R, removes r_1 from R if there exists a state r_2 in R such that $r_1 \preceq r_2$, and then repeats the procedure until all of the states in R are processed. Lines 8–10 of the algorithm implement Optimisation 1. Overall, the algorithm works as follows. Till the set *Next* of macro-states waiting to be processed is non-empty (or a rejecting macro-state is found), the algorithm chooses one macro-state from *Next*, and moves it to the *Processed* set. Moreover, it generates all successors of the chosen macro-state, minimises them, and adds them to *Next* unless there is already some $\preceq^{\forall\exists}$-smaller macro-state in *Next* or in *Processed*. If a new macro-state is added to *Next*, the algorithm at the same time removes all $\preceq^{\forall\exists}$-bigger macro-states from both *Next* and *Processed*. Note that the pruning of the *Next* and *Processed* sets together with checking whether a new macro-state should be added into *Next* can be done within a single iteration through *Next* and *Processed*. We discuss correctness of the algorithm in the next section.

5.1.2 Correctness of the Optimised Universality Checking

In this section, we prove correctness of Algorithm 3. We first introduce some definitions and notations that will be used in the proof. For a macro-state P, define $Dist(P) \in \mathbb{N} \cup \{\infty\}$ as the length of the shortest word in Σ^* that is not in $L(\mathcal{A})(P)$ (if $L(\mathcal{A})(P) = \Sigma^*$, $Dist(P) = \infty$). For a set of macro-states *MStates*, the function $Dist(MStates) \in \mathbb{N} \cup \{\infty\}$ returns the length of the shortest word in Σ^* that is not in the language of some macro-state in *MStates*. More precisely, if $MStates = \emptyset$, $Dist(MStates) = \infty$, otherwise, $Dist(MStates) = \min_{P \in MStates} Dist(P)$. The predicate $Univ(MStates)$ is true if and only if all the macro-states in *MStates* are universal, i.e., $\forall P \in \mathit{MStates} : L(\mathcal{A})(P) = \Sigma^*$.

The lemma bellow follows from the fact that if $L(\mathcal{A})(P) \subseteq L(\mathcal{A})(R)$, then the shortest word rejected by R is also rejected by P.

Lemma 5.2. *Let P and R be two macro-states such that $L(\mathcal{A})(P) \subseteq L(\mathcal{A})(R)$. We have $\mathit{Dist}(P) \leq \mathit{Dist}(R)$.*

Lemma 5.3 describes the invariants used to prove the partial correctness of Algorithm 3.

Lemma 5.3. *The below two loop invariants hold in Algorithm 3:*

1. $\neg\,\mathit{Univ}(\mathit{Processed} \cup \mathit{Next}) \implies \neg\,\mathit{Univ}(\{I\})$.

2. $\neg\,\mathit{Univ}(\{I\}) \implies \mathit{Dist}(\mathit{Processed}) > \mathit{Dist}(\mathit{Next})$.

Proof. It is trivial to see that the invariants hold at the entry of the loop, taking into account Lemma 5.1 covering the effect of the *Minimize* function. We show that the invariants continue to hold when the loop body is executed from a configuration of the algorithm in which the invariants hold. We use $\mathit{Processed}^{old}$ and Next^{old} to denote the values of *Processed* and *Next* when the control is on line 4 before executing the loop body and we use $\mathit{Processed}^{new}$ and Next^{new} to denote their values when the control gets back to line 4 after executing the loop body once. We assume that $\mathit{Next}^{old} \neq \emptyset$.

Let us start with Invariant 1. Assume first that $\mathit{Univ}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$ holds. Then, the macro-state R picked on line 5 must be universal, which holds also for all of its successors and, due to Lemma 5.1, also for their minimised versions, which may be added to *Next* on line 10. Hence, $\mathit{Univ}(\mathit{Processed}^{new} \cup \mathit{Next}^{new})$ holds after executing the loop body, and thus Invariant 1 holds too. Now assume that $\neg\,\mathit{Univ}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$ holds. Then, $\neg\,\mathit{Univ}(\{I\})$ holds, and hence Invariant 1 must hold for $\mathit{Processed}^{new}$ and Next^{new} too.

We proceed to Invariant 2 and we assume that $\neg\,\mathit{Univ}(\{I\})$ holds (the other case being trivial). Hence, $\mathit{Dist}(\mathit{Processed}^{old}) > \mathit{Dist}(\mathit{Next}^{old})$ holds. We distinguish two cases:

1. $\mathit{Dist}(R) = \infty$ or $\exists Q \in \mathit{Processed}^{old} : \mathit{Dist}(Q) \leq \mathit{Dist}(R)$. In this case, $\mathit{Dist}(\mathit{Processed})$ will not decrease on line 5. From $\mathit{Dist}(\mathit{Processed}^{old}) > \mathit{Dist}(\mathit{Next}^{old})$, there exists some macro-state R' in Next^{old} s.t. $\mathit{Dist}(R') = \mathit{Dist}(\mathit{Next}^{old}) < \mathit{Dist}(\mathit{Processed}^{old}) \leq \mathit{Dist}(Q) \leq \mathit{Dist}(R)$. Therefore, $\mathit{Dist}(\mathit{Next})$ will not change on line 5 either. Moreover, for any macro-state P, removing Q s.t. $P \preceq^{\forall\exists} Q$ from *Next* and *Processed* on line 9 and then adding P to *Next* on line 10 cannot invalidate $\mathit{Dist}(\mathit{Processed}^{new}) > \mathit{Dist}(\mathit{Next}^{new})$ since $\mathit{Dist}(P) \leq \mathit{Dist}(Q)$ due to Lemmas 5.1 and 5.2. Hence, Invariant 2 must hold for $\mathit{Processed}^{new}$ and Next^{new} too.

2. $\mathit{Dist}(R) \neq \infty$ and $\neg\exists Q \in \mathit{Processed}^{old} : \mathit{Dist}(Q) \leq \mathit{Dist}(R)$. In this case, the value of $\mathit{Dist}(\mathit{Processed})$ decreases to $\mathit{Dist}(R)$ on line 5. Clearly, $\mathit{Dist}(R) \neq 0$ or else we would have terminated before. Then there must be some successor R' of R which is either rejecting (and the loop stops without getting back to line 4) or one step closer to rejection, meaning

that $Dist(R') < Dist(R)$. Moreover, R' either appears in $Next^{new}$ or there already exists some $R'' \in Next^{old}$ such that $R'' \preceq^{\forall\exists} R'$, meaning that $Dist(Processed^{new}) > Dist(Next^{new})$. It is impossible that $\exists R'' \in Processed^{old} : R'' \preceq^{\forall\exists} R'$, because $\forall R'' \in Processed^{old} : Dist(R'') > Dist(R) > Dist(R')$ and from Lemmas 5.1 and 5.2, $R'' \preceq^{\forall\exists} R'$ implies $Dist(R'') < Dist(R')$. Furthermore, if some macro-state is removed from *Processed* on line 9, $Dist(Processed)$ can only grow, and hence we are done.

□

Due to the finite number of macro-states, we can show that Algorithm 3 eventually terminates.

Lemma 5.4 (Termination)**.** *Algorithm 3 eventually terminates.*

Proof. For the algorithm not to terminate, it would have to be the case that some macro-state is repeatedly added into *Next*. However, once some macro-state R is added into *Next*, there will always be some macro-state $Q \in Processed \cup Next$ such that $Q \preceq^{\forall\exists} R$. This holds since R either stays in *Next*, moves to *Processed*, or is replaced by some Q such that $Q \preceq^{\forall\exists} R$ in each iteration of the loop. Hence, R cannot be added to *Next* for the second time since a macro-state is added to *Next* on line 10 only if there is no $Q \in Processed \cup Next$ such that $Q \preceq^{\forall\exists} R$. □

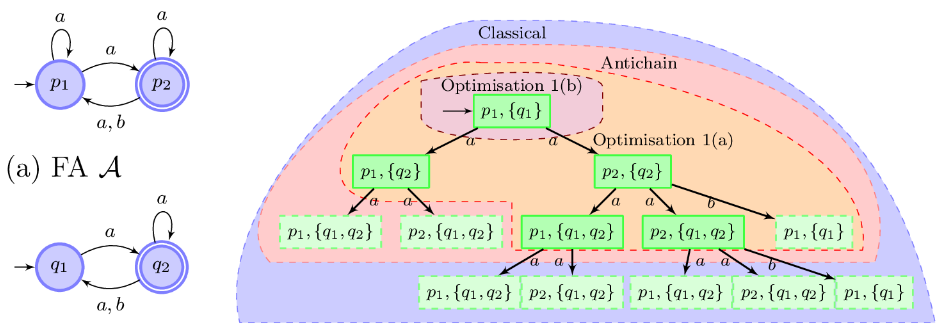
We can now easily prove the main theorem.

Theorem 10. *Algorithm 3 always terminates, and returns* TRUE *iff the input automaton $\mathcal{A}$ is universal.*

Proof. From Lemma 5.4, the algorithm eventually terminates. It returns FALSE only if either the set of initial states is rejecting, or the minimised version R' of some successor S of a macro-state R chosen from *Next* on line 5 is found rejecting. In the latter case, due to Lemma 5.1, S is also rejecting. Then R is non-universal, and hence $Univ(Processed \cup Next)$ is false. By Lemma 5.3 (Invariant 1), we have $\mathcal{A}$ is not universal. The algorithm returns TRUE only when *Next* becomes empty. When *Next* is empty, $Dist(Processed) > Dist(Next)$ is not true. Therefore, by Lemma 5.3 (Invariant 2), $\mathcal{A}$ is universal. □

5.1.3 The FA Language Inclusion Problem

The technique described in Section 5.1.1 can be generalised to solve the *language-inclusion problem.* Let $\mathcal{A}$ and $\mathcal{B}$ be two FA. The *language inclusion problem* for $\mathcal{A}$ and $\mathcal{B}$ is to decide whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$. This problem is also PSPACE-complete. The classical algorithm for solving this problem builds on-the-fly the product automaton $\mathcal{A} \times \overline{\mathcal{B}}$ of $\mathcal{A}$ and the complement of $\mathcal{B}$ and searches for an accepting state. A state in the product automaton $\mathcal{A} \times \overline{\mathcal{B}}$ is a pair (p, P) where p is a state in $\mathcal{A}$ and P is a macro-state in $\mathcal{B}$. For convenience, we call such a pair (p, P) a *product-state.* A product-state is accepting iff p is an accepting state in $\mathcal{A}$ and P is a rejecting macro-state in $\mathcal{B}$. We use $L(\mathcal{A}, \mathcal{B})(p, P)$ to denote the language of the product-state (p, P) in $\mathcal{A} \times \overline{\mathcal{B}}$. The language of $\mathcal{A}$ is not contained



(a) FA $\mathcal{A}$

(b) FA $\mathcal{B}$

(c) A run of the algorithms while checking $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

Figure 5.2: Language Inclusion Checking Algorithms

in the language of $\mathcal{B}$ iff there exists some accepting product-state (p, P) reachable from some initial product-state. Indeed, $L(\mathcal{A}, \mathcal{B})(p, P) = L(\mathcal{A})(p) \setminus L(\mathcal{B})(P)$, and the language of $\mathcal{A} \times \overline{\mathcal{B}}$ consists of words which can be used as witnesses of the fact that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ does not hold. In a similar manner to universality checking, the algorithm can stop the search immediately and conclude that the language inclusion does not hold whenever an accepting product-state is encountered. An example of a run of the classical algorithm is given in Fig. 5.2. We find that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is true and the algorithm generates 13 product-states (Fig. 5.2(c), the area labelled "Classical").

Optimisation 1 that we use for universality checking can be generalised for language inclusion checking as follows. Let $\mathcal{A} = (\Sigma, Q_\mathcal{A}, \delta_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (\Sigma, Q_\mathcal{B}, \delta_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B})$ be two FA such that $Q_\mathcal{A} \cap Q_\mathcal{B} = \emptyset$. We denote by $\mathcal{A} \cup \mathcal{B}$ the FA $(\Sigma, Q_\mathcal{A} \cup Q_\mathcal{B}, \delta_\mathcal{A} \cup \delta_\mathcal{B}, I_\mathcal{A} \cup I_\mathcal{B}, F_\mathcal{A} \cup F_\mathcal{B})$. Let $\preceq$ be a relation in $(\mathcal{A} \cup \mathcal{B})^{\subseteq}$. During the process of constructing the product automaton and searching for an accepting product-state, we can stop the search from a product-state (p, P) if (a) there exists some visited product-state (r, R) such that $p \preceq r$ and $R \preceq^{\forall\exists} P$, or (b) $\exists p' \in P : p \preceq p'$. Optimisation 1(a) is justified by Lemma 5.5, which is very similar to Lemma 5.1 for universality checking.

Lemma 5.5. *Let $\mathcal{A}$, $\mathcal{B}$ be two FA, (p, P), (r, R) be two product-states where p, r are states in $\mathcal{A}$ and P, R are macro-states in $\mathcal{B}$, and $\preceq$ be a relation in $(\mathcal{A} \cup \mathcal{B})^{\subseteq}$. Then, $p \preceq r$ and $R \preceq^{\forall\exists} P$ implies $L(\mathcal{A}, \mathcal{B})(p, P) \subseteq L(\mathcal{A}, \mathcal{B})(r, R)$.*

By the above lemma, if a word takes the product-state (p, P) to an accepting product-state, it will also take (r, R) to an accepting product-state. Therefore, we do not need to continue the search from (p, P).

Let us use Fig. 5.2(c) to illustrate Optimisation 1(a). As we mentioned, the pure antichain-based approach can be viewed as a special case of our simulation enhanced antichain approach when $\preceq$ is the identity. When $\preceq$ is the identity, we do not need to continue the search from the product-state $(p_2, \{q_1, q_2\})$ because $\{q_2\} \subseteq \{q_1, q_2\}$. In this case, the algorithm generates 8 product-states (Fig. 5.2(c), the area labelled "Antichain"). In the case that $\preceq$ is the maximal simulation, we do not need to continue the search from product-states $(p_1, \{q_2\})$, $(p_1, \{q_1, q_2\})$, and $(p_2, \{q_1, q_2\})$ because $q_1 \preceq q_2$ and the algorithm already visited the product-states $(p_1, \{q_1\})$ and $(p_2, \{q_2\})$. Hence, the algorithm generates only 6 product-states (Fig. 5.2(c), the area labelled "Optimisation 1(a)").

If the condition of Optimisation 1(b) holds, we have that the language of p (w.r.t. $\mathcal{A}$) is a subset of the language of P (w.r.t. $\mathcal{B}$). In this case, for any word that takes p to an accepting state in $\mathcal{A}$, it also takes P to an accepting macro-state in $\mathcal{B}$. Hence, we do not need to continue the search from the product-state (p, P) because all of its successor states are rejecting product-states. Consider again the example in Fig. 5.2(c). With Optimisation 1(b), if $\preceq$ is the maximal simulation on the states of $\mathcal{A} \cup \mathcal{B}$, we do not need to continue the search from the first product-state $(p_1, \{q_1\})$ because $p_1 \preceq q_1$. In this case, the algorithm can conclude that the language inclusion holds immediately after the first product-state is generated (Fig. 5.2(c), the area labelled "Optimisation 1(b)").

Observe that from Lemma 5.5, it holds that for any product-state (p, P) such that $p_1 \preceq p_2$ for some $p_1, p_2 \in P$, $L(\mathcal{A}, \mathcal{B})(p, P) = L(\mathcal{A}, \mathcal{B})(p, P \setminus \{p_1\})$ (as $P \preceq^{\forall\exists} P \setminus \{p_1\}$). Optimisation 2 that we used for universality checking can therefore be generalised for language inclusion checking too.

We give the pseudocode of our optimised inclusion checking in Algorithm 4, which is a straightforward extension of Algorithm 3. In the algorithm, the definition of the *Minimize*(R) function is the same as what we have defined in Section 5.1.1. The function *Initialize*$(PStates)$ applies Optimisation 1 on the set of product-states *PStates* to avoid unnecessary searching. More precisely, it returns a maximal subset of *PStates* such that (1) for any two elements (p, P), (q, Q) in the subset, $p \not\preceq q \vee Q \not\preceq^{\forall\exists} P$ and (2) for any element (p, P) in the subset, $\forall p' \in P : p \not\preceq p'$. We define the post-image of a product-state $Post((p, P)) := \{(p', P') \mid \exists a \in \Sigma : (p, a, p') \in \delta, P' = \{p'' \mid \exists p \in P : (p, a, p'') \in \delta\}\}$.

Algorithm 4: *Language Inclusion Checking*

```
Input: FA $\mathcal{A} = (\Sigma, Q_\mathcal{A}, \delta_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$, $\mathcal{B} = (\Sigma, Q_\mathcal{B}, \delta_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B})$. A relation
       $\preceq \in (\mathcal{A} \cup \mathcal{B})^{\subseteq}$.
Output: TRUE if $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Otherwise, FALSE.
1  if there is an accepting product-state in $\{(i, I_\mathcal{B}) \mid i \in I_\mathcal{A}\}$ then return
   FALSE;
2  $Processed := \emptyset$;
3  $Next := Initialize(\{(i, Minimize(I_\mathcal{B})) \mid i \in I_\mathcal{A}\})$;
4  while $Next \neq \emptyset$ do
5      Pick and remove a product-state $(r, R)$ from $Next$ and move it to
       $Processed$;
6      foreach $(p, P) \in \{(r', Minimize(R')) \mid (r', R') \in Post((r, R))\}$ do
7          if $(p, P)$ is an accepting product-state then return FALSE;
8          else if $\neg\exists p' \in P$ s.t. $p \preceq p'$ then
9              if $\neg\exists (s, S) \in Processed \cup Next$ s.t. $p \preceq s \wedge S \preceq^{\forall\exists} P$ then
10                 Remove all $(s, S)$ from $Processed \cup Next$ s.t. $s \preceq p \wedge P \preceq^{\forall\exists} S$;
11                 Add $(p, P)$ to $Next$;
12 return TRUE
```

Correctness: Define $Dist(P) \in \mathbb{N} \cup \{\infty\}$ as the length of the shortest word in the language of the product-state P or ∞ if the language of P is empty. The value $Dist(PStates) \in \mathbb{N} \cup \{\infty\}$ is the length of the shortest word in the language of some product-state in *PStates* or ∞ if *PStates* is empty. The predicate $Incl(PStates)$ is true iff for all product-states (p, P) in *PStates*, $L(\mathcal{A})(p) \subseteq L(\mathcal{B})(P)$. The correctness of Algorithm 4 can now be proved in a very similar way to Algorithm 3, using the invariants below:

1. $\neg Incl(Processed \cup Next) \implies \neg Incl(\{(i, I_\mathcal{B}) \mid i \in I_\mathcal{A}\})$.
2. $\neg Incl(\{(i, I_\mathcal{B}) \mid i \in I_\mathcal{A}\}) \implies Dist(Processed) > Dist(Next)$.

Theorem 11. *Algorithm 4 terminates, and returns* TRUE *iff* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

5.2 Universality and Language Inclusion of Tree Automata

To optimise universality and inclusion checking on word automata, we used relations that imply language inclusion. For the case of universality and inclusion checking on tree automata, we now propose to use relations that imply inclusion of languages of contexts (context is the notion of a tree with "holes" instead of (all) leaves defined in Chapter 4) that are accepted from tuples of tree automata states. As we will see, a relation that fits here best is upward simulation induced by identity introduced in Chapter 4. Notice that in contrast to the notion of a language accepted from a state of a word automaton, which refers to possible "futures" of the state, the notion of a language accepted at a state of a TA refers to possible "pasts" of the state. Our notion of languages of contexts accepted from tuples of tree automata states speaks again about the future of states, which turns out useful when trying to optimise the (antichain-based) subset construction for TA. Below, we state formal definitions of the notions needed within this chapter.

The language of $\mathcal{A}$ accepted from a tuple $(q_1, \ldots, q_n)$ of states is the set of contexts $\mathcal{L}^\Box(\mathcal{A})(q_1, \ldots, q_n) = \{t \in T^\Box \mid t(q_1, \ldots, q_n) \Longrightarrow q \text{ for some } q \in F\}$. We define the language accepted from a tuple of macro-states $(P_1, \ldots, P_n) \subseteq Q^n$ as the set $\mathcal{L}^\Box(\mathcal{A})(P_1, \ldots, P_n) = \bigcup\{\mathcal{L}^\Box(\mathcal{A})(q_1, \ldots, q_n) \mid (q_1, \ldots, q_n) \in P_1 \times \ldots \times P_n\}$. We define $Post_a(q_1, \ldots, q_n) := \{q \mid (q_1, \ldots, q_n) \xrightarrow{a} q\}$. For a tuple of macro-states, we let $Post_a(P_1, \ldots, P_n) := \bigcup\{Post_a(q_1, \ldots, q_n) \mid (q_1, \ldots, q_n) \in P_1 \times \cdots \times P_n\}$.

Let us use $t^\Box$ to denote the context that arises from a tree $t \in T(\Sigma)$ by replacing all the leaf symbols of t by $\Box$ and let for every leaf symbol $a \in \Sigma$, $I_a = \{q \mid \xrightarrow{a} q\}$ is the so called a-initial macro-state. Languages accepted *at* final states of $\mathcal{A}$ correspond to the languages accepted *from* tuples of initial macro-states of $\mathcal{A}$ as stated in Lemma 5.6.

Lemma 5.6. *Let t be a tree over Σ with leaves labelled by $a_1, \ldots, a_n$. Then $t \in \mathcal{L}(\mathcal{A})$ if and only if $t^\Box \in \mathcal{L}^\Box(\mathcal{A})(I_{a_1}, \ldots, I_{a_n})$.*

5.2.1 The Role of Upward Simulation

We now work towards defining suitable relations on states of TA allowing us to optimise the universality and inclusion checking. We extend relations $\preceq \in Q \times Q$ on states to tuples of states such that $(q_1, \dots, q_n) \preceq (r_1, \dots, r_n)$ iff $q_i \preceq r_i$ for each $1 \leq i \leq n$. We define the set $\mathcal{A}^{\subseteq}$ of relations that imply inclusion of languages of tuples of states such that $\preceq \in \mathcal{A}^{\subseteq}$ iff $(q_1, \dots, q_n) \preceq (r_1, \dots, r_n)$ implies $\mathcal{L}^{\square}(\mathcal{A})(q_1, \dots, q_n) \subseteq \mathcal{L}^{\square}(\mathcal{A})(r_1, \dots, r_n)$.

A relation that satisfies the above property is the upward simulation induced by identity defined in Chapter 4. For convenience, in this chapter, we will call it simply upward simulation. We note that it can be equivalently defined in a non-parametric way as follows: An *upward simulation* on $\mathcal{A}$ is a relation $\preceq \subseteq Q \times Q$ such that if $q \preceq r$, then (1) $q \in F \implies r \in F$ and (2) if $(q_1, \dots, q_n) \xrightarrow{a} q'$ where $q = q_i$, then $(q_1, \dots, q_{i-1}, r, q_{i+1}, \dots, q_n) \xrightarrow{a} r'$ where $q' \preceq r'$. [1]

Lemma 5.7. *For the maximal upward simulation $\preceq$ on $\mathcal{A}$, we have $\preceq \in \mathcal{A}^{\subseteq}$.*

Proof. We first show that the maximal upward simulation $\preceq$ has the following property: If $(q_1, \dots, q_n) \xrightarrow{a} q'$ in $\mathcal{A}$, then for every $(r_1, \dots, r_n)$ with $(q_1, \dots, q_n) \preceq (r_1, \dots, r_n)$, there is $r' \in Q$ such that $q' \preceq r'$ and $(r_1, \dots, r_n) \xrightarrow{a} r'$. From $(q_1, \dots, q_n) \xrightarrow{a} q'$ and $q_1 \preceq r_1$, we have that there is some rule $(r_1, q_2, \dots, q_n) \xrightarrow{a} s_1$ such that $q' \preceq s_1$. From the existence of $(r_1, q_2, \dots, q_n) \xrightarrow{a} s_1$ and from $q_2 \preceq r_2$, we then get that there is some rule $(r_1, r_2, q_3, \dots, q_n) \xrightarrow{a} s_2$ such that $s_1 \preceq s_2$, etc. Since the maximal upward simulation is transitive [ABH+08c], we obtain the property mentioned above. This in turn implies Lemma 5.7. □

5.2.2 Tree Automata Universality Checking

We now show how upward simulations can be used for optimised universality checking on tree automata. Let $\mathcal{A} = (\Sigma, Q, \Delta, F)$ be a tree automaton. We define $T_n^{\square}(\Sigma)$ as the set of all contexts over Σ with n leaves. We say that an n-tuple $(q_1, \dots, q_n)$ of states of $\mathcal{A}$ is universal if $\mathcal{L}^{\square}(\mathcal{A})(q_1, \dots, q_n) = T_n^{\square}(\Sigma)$, this is, all contexts with n leaves constructable over Σ can be accepted from $(q_1, \dots, q_n)$. A set of macro-states *MStates* is universal if all tuples in *MStates** are universal. From Lemma 5.6, we can deduce that $\mathcal{A}$ is universal (i.e., $\mathcal{L}(\mathcal{A}) = T(\Sigma)$) if and only if $\{I_a \mid a \in \Sigma_0\}$ is universal.

The following Lemma allows us to design a new TA universality checking algorithm in a similar manner to Algorithm 3 using Optimisations 1 and 2 from Section 5.1.1.

Lemma 5.8. *For a given $\preceq \in \mathcal{A}^{\subseteq}$ and two tuples of macro-states of $\mathcal{A}$, if $(R_1, \dots, R_n) \preceq^{\forall\exists} (P_1, \dots, P_n)$, then $\mathcal{L}^{\square}(\mathcal{A})(R_1, \dots, R_n) \subseteq \mathcal{L}^{\square}(\mathcal{A})(P_1, \dots, P_n)$.*

Algorithm 5 describes our simulation enhanced antichain approach to checking universality of tree automata in pseudocode. It resembles closely Algorithm 3. There are two main differences: (1) The initial value of the *Next* set is the

[1] Upward simulations parametrised by a downward simulation greater than the identity cannot be used in our framework since they do not generally imply inclusion of languages of tuples of states.

Algorithm 5: *Tree Automata Universality Checking*

```
Input: A tree automaton $\mathcal{A} = (\Sigma, Q, \Delta, F)$ and a relation $\preceq \in \mathcal{A}^{\subseteq}$.
Output: TRUE if $\mathcal{A}$ is universal. Otherwise, FALSE.
1  if $\exists a \in \Sigma_0$ such that $I_a$ is rejecting then return FALSE;
2  Processed:=$\emptyset$;
3  Next:= Initialize{Minimize($I_a$) | $a \in \Sigma_0$};
4  while Next $\neq \emptyset$ do
5      Pick and remove a macro-state $R$ from Next and move it to Processed;
6      foreach $P \in \{Minimize(R') \mid R' \in Post(Processed)(R)\}$ do
7          if $P$ is a rejecting macro-state then return FALSE;
8          else if $\neg\exists Q \in Processed \cup Next$ s.t. $Q \preceq^{\forall\exists} P$ then
9              Remove all $Q$ from Processed $\cup$ Next s.t. $P \preceq^{\forall\exists} Q$;
10             Add $P$ to Next;
11 return TRUE
```

result of applying the function *Initialize* to the set $\{Minimize(I_a) \mid a \in \Sigma_0\}$. *Initialize* returns the set of all macro-states in $\{Minimize(I_a) \mid a \in \Sigma_0\}$, which are minimal w.r.t. $\preceq^{\forall\exists}$ (i.e., those macro states with the best chance of finding a counterexample to universality). (2) The computation of the *Post*-image of a set of macro-states is a bit more complicated. More precisely, for each symbol $a \in \Sigma_n, n \in \mathbb{N}$, we have to compute the post image of each n-tuple of macro-states from the set. We design the algorithm such that we avoid computing the *Post*-image of a tuple more than once. We define the *Post*-image $Post(MStates)(R)$ of a set of macro-states *MStates* w.r.t. a macro-states $R \in MStates$. It is the set of all macro-states $P = Post_a(P_1, \ldots, P_n)$ where $a \in \Sigma_n, n \in \mathbb{N}$ and R occurs at least once in the tuple $(P_1, \ldots, P_n) \in MStates^*$. Formally, $Post(MStates)(R) = \bigcup_{a\in\Sigma}\{Post_a(P_1, \ldots, P_n) \mid n = \#(a), P_1, \ldots, P_n \in MStates, R \in \{P_1, \ldots, P_n\}\}$.

5.2.3 Correctness of the TA Universality Checking

In this section, we prove correctness of Algorithm 5 in a very similar way to Algorithm 3, using suitably modified notions of distances and ranks. Let $\mathcal{A} = (Q, \Sigma, \Delta, F)$ be a TA. For $n \geq 0$ and an n-tuple of macro-states $(Q_1, \ldots, Q_n)$ where $Q_i \subseteq Q$ for $1 \leq i \leq n$, we let $\mathbf{Dist}(Q_1, \ldots, Q_n) = 0$ iff $Q_i \cap F = \emptyset$ for some $i \in \{1, \ldots, n\}$. We define $\mathbf{Dist}(Q_1, \ldots, Q_n) = k \in \mathbb{N}^+ \cup \{\infty\}$ iff $Q_i \subseteq F$ for all $i \in \{1, \ldots, n\}$ and $k = \min(\{|t| \mid t \in T_n^{\square}(\Sigma) \wedge t \notin \mathcal{L}^{\square}(\mathcal{A})(Q_1, \ldots, Q_n)\})$. Here, $|t|$ is the number of nodes of t and we assume $\min(\emptyset) = \infty$. For a set *MStates* of macro-states over Q, we define the measure $\mathbf{Rank}(MStates) = \min(\{\mathbf{Dist}(Q_1, \ldots, Q_n) \mid n \geq 1 \wedge \forall 1 \leq i \leq n : Q_i \in MStates\})$ and the predicate $\mathbf{Univ}(MStates) \iff \mathbf{Rank}(MStates) = \infty$.

Lemma 5.9. *The below two loop invariants hold in Algorithm 5:*

1. $\neg\mathbf{Univ}(Processed \cup Next) \implies \neg\mathbf{Univ}(\{I_a \mid a \in \Sigma_0\})$.

2. $\neg\mathbf{Univ}(\{I_a \mid a \in \Sigma_0\}) \implies \mathbf{Rank}(\mathit{Processed}) > \mathbf{Rank}(\mathit{Processed} \cup \mathit{Next})$*.*

Proof. It is trivial to see that the invariants hold at the entry of the loop, taking into account Lemma 5.8. We show that the invariants continue to hold when the loop body is executed from a configuration of the algorithm in which the invariants hold. We use $\mathit{Processed}^{old}$ and Next^{old} to denote the values of *Processed* and *Next* when the control is on line 4 before executing the loop body and we use $\mathit{Processed}^{new}$ and Next^{new} to denote their values when the control gets back to line 4 after executing the loop body once. We assume that $\mathit{Next}^{old} \neq \emptyset$.

Let us start with Invariant 1. Assume first that $\mathbf{Univ}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$ holds. Then, the macro-state R can appear within tuples constructed over $\mathit{Processed}^{old} \cup \mathit{Next}^{old}$ which are universal only. In such a case, all macro-states Q reachable from all tuples T built over $\mathit{Processed}^{old} \cup \mathit{Next}^{old}$ are such that when we add them to $\mathit{Processed}^{old} \cup \mathit{Next}^{old}$, the resulting set will still allow building universal tuples only. Otherwise, one could take a non-universal tuple containing some of the newly added macro-states Q, replace Q by the tuple T from which it arose, and obtain a non-universal tuple over $\mathit{Processed}^{old} \cup \mathit{Next}^{old}$, which is impossible. Hence, the possibility of adding the new macro-states to *Next* on line 10 cannot cause non-universality of $\mathit{Processed}^{new} \cup \mathit{Next}^{new}$, which due to Lemma 5.8 holds when adding the minimised macro-states too. Moreover, removing elements from *Next* or *Processed* cannot cause non-universality either. Hence, Invariant 1 holds over $\mathit{Processed}^{new}$ and Next^{new} in this case. Next, let us assume that $\neg\mathbf{Univ}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$ holds. Then, $\neg\mathbf{Univ}(\{I_a \mid a \in \Sigma_0\})$ holds, and hence Invariant 1 must hold for $\mathit{Processed}^{new}$ and Next^{new} too.

We proceed to Invariant 2 assuming that $\neg\mathbf{Univ}(\{I_a \mid a \in \Sigma_0\})$ holds (the other case is trivial). Hence, $\mathbf{Rank}(\mathit{Processed}^{old}) > \mathbf{Rank}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$ holds. We distinguish two cases:

1. In order to build a tuple T over $\mathit{Processed}^{old}$ and Next^{old} that is of **Dist** equal to $\mathbf{Rank}(\mathit{Processed}^{old} \cup \mathit{Next}^{old})$, one needs to use a macro-state Q in $\mathit{Next}^{old} \setminus \{R\}$. The macro-state Q stays in Next^{new} or is replaced by a $\preceq^{\forall\exists}$-smaller macro-state added to *Next* on line 10 that, due to Lemma 5.8, can only allow to build tuples of the same or even smaller **Dist**. Likewise, the macro-states accompanying Q in T stay in Next^{new} or $\mathit{Processed}^{new}$ or are replaced by $\preceq^{\forall\exists}$-smaller macro-states added to *Next* on line 10 allowing to build tuples of the same or smaller **Dist**, due to Lemma 5.8. Hence, moving R to *Processed* on line 5 cannot cause the invariant to break. Moreover, adding some further macro-states to *Next* on line 10 can only cause $\mathbf{Rank}(\mathit{Processed} \cup \mathit{Next})$ to decrease while removing macro-states from *Processed* on line 9 can only cause $\mathbf{Rank}(\mathit{Processed})$ to grow. Finally, replacing a macro-state in *Next* by a $\preceq^{\forall\exists}$-smaller one as a combined effect of lines 9 and 10 can again just decrease $\mathbf{Rank}(\mathit{Processed} \cup \mathit{Next})$, due to Lemma 5.8. Hence, in this case, Invariant 2 must hold over $\mathit{Processed}^{new}$ and Next^{new}.

2. One can build some tuple T over $\mathit{Processed}^{old}$ and Next^{old} that is of **Dist**

equal to **Rank**($\mathit{Processed}^{old} \cup \mathit{Next}^{old}$) using $\mathit{Processed}^{old} \cup \{R\}$ only. In this case, there must be tuples constructable over $\mathit{Processed}^{old} \cup \{R\}$ and containing R that are not universal. We can distinguish the following subcases:

a) From some of the tuples built over $\mathit{Processed}^{old} \cup \{R\}$ and containing R, a non-accepting macro-state is reached via a single transition of $\mathcal{A}$, and the algorithm stops without getting back to line 4.

b) Otherwise, some macro-states that appear in $\mathit{Post}(\mathit{Processed}, R)$ and that will be added in the minimised form to *Next* must allow one to construct tuples which are of **Dist** smaller than those based on R. This holds since if a macro-state Q is reached from some tuple T containing R by a single transition, we can replace T in larger tuples leading to non-acceptation by Q, and hence decrease the size of the context needed to reach non-acceptation. Taking into account Lemma 5.8 to cover the effect of the minimisation and using a similar reasoning as above for covering the effect of lines 9 and 10, it is then clear that Invariant 2 will remain to hold in this case.

□

We can now prove Lemma 5.10 and Theorem 12 below in a very similar way as Lemma 5.4 and Theorem 10, respectively.

Lemma 5.10. *Algorithm 5 eventually terminates.*

Theorem 12. *Algorithm 5 always terminates, and returns* TRUE *if and only if the input tree automaton $\mathcal{A}$ is universal.*

5.2.4 Downward Universality Checking with Antichains

The *upward universality* introduced above tree automata automata conceptually corresponds to the *forward* universality checking of finite word automata of [WDHR06, DR10] where also a dual *backward* universality checking is introduced. The backward universality algorithm from [WDHR06, DR10] is based on computing the *controllable predecessors* of the set of non-final states. Controllable predecessors are the predecessors that can be forced by an input symbol to continue into a given set of states. Then, the automaton is non-universal iff the controllable predecessors of the non-final states cover the set of initial states.

Downward universality checking for tree automata as a dual approach to upward universality checking is problematic since the controllable predecessors of a set of states $s \subseteq Q$ of an TA $\mathcal{A} = (Q, \Sigma, F, \Delta)$ do not form a set of states, but a set of *tuples* of states, i.e., for $a \in \Sigma$, $\mathit{CPre}_a(s) = \{(q_1, \ldots, q_n) \mid n \in \mathbb{N} \wedge \forall q \in Q : (q_1, \ldots, q_n) \xrightarrow{a} q \in s\}$. Note that if we flatten the set $\mathit{CPre}_a(s)$ to the set $\mathit{FCPre}_a(s)$ of states that appear in some of the tuples of $\mathit{CPre}_a(s)$ and check that starting from leaf rules the computation can be forced into some subset of $\mathit{FCPre}_a(s)$, then this does not imply that the computation can be forced into some state of s. That is because for any rule $(q_1, \ldots, q_n) \xrightarrow{a} q$, $q \in s$, not all of the states $q_1, \ldots, q_n$ may be reached. Moreover, it is too strong to require that starting from leaf rules, it must be possible to force the computation into all states of $FCPre_a(s)$. Clearly, it is enough if the computation starting from leaf rules can be forced into s via some of the vectors in $CPre_a(s)$, not necessarily all of them. Also, if we keep $CPre_a(s)$ for $s \subseteq Q$ as a set of vectors, we also have to define the notion of controllable predecessors for sets of vectors of states, which is a set of vectors of vectors of states, etc. Clearly, such an approach is not practical and does not even terminate. Yet, we feel that some further research on ways possibly circumventing this problems can be interesting as we discuss in Section 5.5.

5.2.5 Tree Automata Language Inclusion Checking

We are interested in testing language inclusion of two tree automata $\mathcal{A} = (\Sigma, Q_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (\Sigma, Q_\mathcal{B}, \Delta_\mathcal{B}, F_\mathcal{B})$. From Lemma 5.6, we have that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ if and only if for every tuple $a_1, \ldots, a_n$ of leaf symbols from Σ_0, $\mathcal{L}^\square(\mathcal{A})(I^\mathcal{A}_{a_1}, \ldots, I^\mathcal{A}_{a_n}) \subseteq \mathcal{L}^\square(\mathcal{B})(I^\mathcal{B}_{a_1}, \ldots, I^\mathcal{B}_{a_n})$. In other words, for any $a_1, \ldots, a_n \in \Sigma_0$, every context that can be accepted from a tuple of states from $I^\mathcal{A}_{a_1} \times \ldots \times I^\mathcal{A}_{a_n}$ can also be accepted from a tuple of states from $I^\mathcal{B}_{a_1} \times \ldots \times I^\mathcal{B}_{a_n}$. This justifies a similar use of the notion of product-states as in Section 5.1.3. We define the language of a tuple of product-states as $\mathcal{L}^\square(\mathcal{A}, \mathcal{B})((q_1, P_1), \ldots, (q_n, P_n)) := \mathcal{L}^\square(\mathcal{A})(q_1, \ldots, q_n) \setminus \mathcal{L}^\square(\mathcal{B})(P_1, \ldots, P_n)$. Observe that we obtain that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ iff the language of every n-tuple (for any $n \in \mathbb{N}$) of product-states from the set $\{(i, I^\mathcal{B}_a) \mid a \in \Sigma_0, i \in I^\mathcal{A}_a\}$ is empty.

Our algorithm for testing language inclusion of tree automata will check whether it is possible to reach a product-state of the form (q, P) with $q \in F_\mathcal{A}$ and $P \cap F_\mathcal{B} = \emptyset$ (that we call accepting) from a tuple of product-states from $\{(i, I^\mathcal{B}_a) \mid a \in \Sigma_0, i \in I^\mathcal{A}_a\}$. The following lemma allows us to use Optimisation 1(a) and Optimisation 2 from Section 5.1.3.

Lemma 5.11. *Given $\preceq \in (\mathcal{A} \cup \mathcal{B})^\subseteq$, two tuples of states and two tuples of product-states with $(p_1, \ldots, p_n) \preceq (r_1, \ldots, r_n)$ and $(R_1, \ldots, R_n) \preceq^{\forall\exists} (P_1, \ldots, P_n)$, it holds that $\mathcal{L}^\square(\mathcal{A}, \mathcal{B})((p_1, P_1), \ldots, (p_n, P_n)) \subseteq \mathcal{L}^\square(\mathcal{A}, \mathcal{B})((r_1, R_1), \ldots, (r_n, R_n))$.*

It is also possible to use Optimisation 1(b) where we stop searching from product-states of the form (q, P) such that $q \preceq r$ for some $r \in P$. However, note that this optimisation is of limited use for tree automata. Under the assumption that the automata $\mathcal{A}$ and $\mathcal{B}$ do not contain useless states, the reason is that for any $q \in Q_\mathcal{A}$ and $r \in Q_\mathcal{B}$, if q appears at a left-hand side of some rule of arity more than 1, then no reflexive relation from $\preceq \in (\mathcal{A} \cup \mathcal{B})^\subseteq$ allows $q \preceq r$.[2]

Algorithm 6 describes our method for checking language inclusion of TA in pseudocode. It closely follows Algorithm 4. It differs in two main points. First, the initial value of the *Next* set is the result of applying the function *Initialize* on the set $\{(i, \mathit{Minimize}(I^\mathcal{B}_a)) \mid a \in \Sigma_0, i \in I^\mathcal{A}_a\}$ where *Initialize* is the same function as in Algorithm 4. Second, the computation of the *Post* image of a set

[2]To see this, assume that a context tree t is accepted from $(q_1, \ldots, q_n) \in Q^n_\mathcal{A}, q = q_i, 1 \leq i \leq n$. If $q \preceq r$, then by the definition of $\preceq$, $t \in \mathcal{L}^\square(\mathcal{A} \cup \mathcal{B})(q_1, \ldots, q_{i-1}, r, q_{i+1}, \ldots, q_n)$. However, that cannot happen, as $\mathcal{A} \cup \mathcal{B}$ does not contain any rules with left hand sides containing both states from $\mathcal{A}$ and states from $\mathcal{B}$.

Algorithm 6: *Tree Automata Language Inclusion Checking*

```
Input: TA $\mathcal{A}$ and $\mathcal{B}$ over an alphabet $\Sigma$. A relation $\preceq \in (\mathcal{A} \cup \mathcal{B})^{\subseteq}$.
Output: TRUE if $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Otherwise, FALSE.
1  if there exists an accepting product-state in $\bigcup_{a\in\Sigma_0}\{(i, I_a^{\mathcal{B}}) \mid i \in I_a^{\mathcal{A}}\}$ then
   return FALSE;
2  Processed:=$\emptyset$;
3  Next:=Initialize($\bigcup_{a\in\Sigma_0}\{(i, Minimize(I_a^{\mathcal{B}})) \mid i \in I_a^{\mathcal{A}}\}$);
4  while Next $\neq \emptyset$ do
5      Pick and remove a product-state $(r, R)$ from Next and move it to
       Processed;
6      foreach $(p, P) \in \{(r', Minimize(R')) \mid (r', R') \in Post(Processed)(r, R)\}$
       do
7          if $(p, P)$ is an accepting product-state then return FALSE;
8          else if $\neg\exists p' \in P$ s.t. $p \preceq p'$ then
9              if $\neg\exists(q, Q) \in Processed \cup Next$ s.t. $p \preceq q \wedge Q \preceq^{\forall\exists} P$ then
10                 Remove all $(q, Q)$ from $Processed \cup Next$ s.t.
                   $q \preceq p \wedge P \preceq^{\forall\exists} Q$;
11                 Add $(p, P)$ to Next;
12 return TRUE
```

of product-states means that for each symbol $a \in \Sigma_n, n \in \mathbb{N}$, we construct the $Post_a$-image of each n-tuple of product-states from the set. Like in Algorithm 5, we design the algorithm such that we avoid computing the $Post_a$-image of a tuple more than once. We define the post image $Post(PStates)(r, R)$ of a set of product-states $PStates$ w.r.t. a product-state $(r, R) \in PStates$. It is the set of all product-states (q, P) such that there is some $a \in \Sigma, \#(a) = n$ and some n-tuple $((q_1, P_1), \ldots, (q_n, P_n))$ of product-states from $PStates$ that contains at least one occurrence of (r, R) where $q \in Post_a(q_1, \ldots, q_n)$ and $P = Post_a(P_1, \ldots, P_n)$.

Correctness of the TA Language Inclusion Checking. We prove correctness of Algorithm 6 in a very similar way to Algorithm 4, using suitably modified notions of distances and ranks.

Let $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$ be two tree automata. Given $n \geq 0$ and an n-tuple of macro-states $((q_1, P_1), \ldots, (q_n, P_n))$, we define $\mathbf{Dist}((q_1, P_1), \ldots, (q_n, P_n)) = 0$ iff $\epsilon \in \mathcal{L}^{\Box}(\mathcal{A}, \mathcal{B})((q_1, P_1), \ldots, (q_n, P_n))$. Otherwise we define $\mathbf{Dist}((q_1, P_1), \ldots, (q_n, P_n)) = k \in \mathbb{N}^+ \cup \{\infty\}$ iff $k = \min(\{|t| \mid t \in T_n^{\Box}(\Sigma) \wedge t \in \mathcal{L}^{\Box}(\mathcal{A}, \mathcal{B})((q_1, P_1), \ldots, (q_n, P_n))\})$. Here, we assume $\min(\emptyset) = \infty$. For a set $PStates$ of product-states, we let $\mathbf{Rank}(PStates) = \min(\{\mathbf{Dist}((q_1, P_1), \ldots, (q_n, P_n)) \mid n \geq 1 \wedge \forall 1 \leq i \leq n : (q_i, P_i) \in PStates\})$. The predicate $Incl(PStates)$ is defined to be true iff $\mathbf{Rank}(PStates) = \infty$.

Lemma 5.12. *The following two loop invariants hold in Algorithm 6:*

1. $\neg Incl(Processed \cup Next) \implies \neg Incl(\bigcup_{a\in\Sigma_0}\{(i, I_a^{\mathcal{B}}) \mid i \in I_a^{\mathcal{A}}\})$.

2. $\neg Incl(\bigcup_{a\in\Sigma_0}\{(i, I_a^{\mathcal{B}}) \mid i \in I_a^{\mathcal{A}}\}) \implies \mathbf{Rank}(Processed) > \mathbf{Rank}(Next \cup Processed)$.

The proof is similar to that of Lemma 5.9. With the invariants in hand, we can now prove Lemma 5.13 and Theorem 13 below in a very similar way as Lemma 5.4 and Theorem 10, respectively.

Lemma 5.13. *Algorithm 6 eventually terminates.*

Theorem 13. *Algorithm 6 terminates, and returns* TRUE *iff* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

5.3 Experiments with Classical versus Pure Antichain Algorithms for Tree Automata

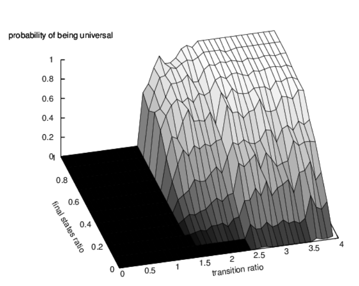
In this section, we describe the experimental results obtained in [BHH+08b] where we compare classical subset construction based algorithms for tree automata with pure antichain based ones. The pure antichain algorithms may be seen as special cases of Algorithms 4 and 6, where the role of simulation relation is played by the identity relation.

We have implemented the above pure antichain approach for testing universality and inclusion of tree automata in a prototype based on the Timbuk tree automata library [GVT03]. We give the results of our experiments run on an Intel Xeon processor at with 2.7GHz and 16GB of memory in Fig. 5.3. We ran our tests on randomly generated automata and on automata obtained from abstract regular tree model checking applied in verification of several pointer-manipulating programs.

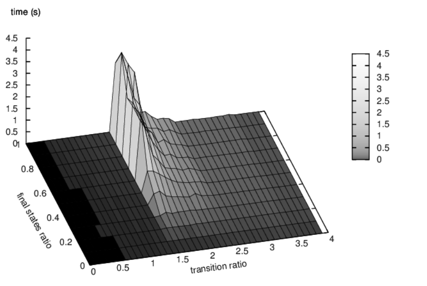
In the random tests, we use an approach for systematic generating random automata with different parameters inspired by the approach proposed by Tabakov and Vardi in [TV05] (which was also used in [WDHR06]). The parameters of the generated automata are *number of states*, *density of their transitions* (the average number of different right-hand side states for a given left-hand side of a transition rule, i.e., $|\Delta|/|\{a(q_1, \ldots, q_n) \mid\in \Sigma, q \in Q : (q_1, \ldots, q_n) \xrightarrow{a} q\}|$) and the *density of their final states* (i.e., $|F|/|Q|$).

5.3.1 Experiments with Antichain-based Universality Checking

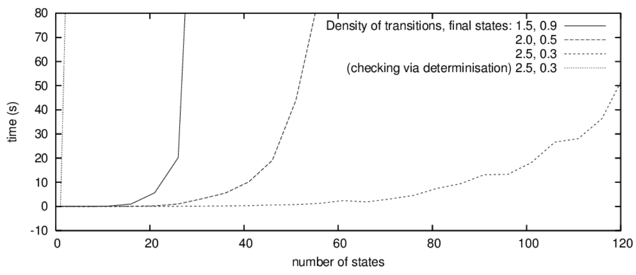
For experiments with the pure antichain tree automata universality algorithm, we used automata with 20 states and varied transition density and density of final states. Fig. 5.3(a) shows the probability of such tree automata being universal, and Fig. 5.3(b) the average times needed for checking their universality using our antichain-based approach. The difficult instances are those where the probability of being universal is about one half. In Fig. 5.3(c), we show how the running times change for some selected instances of the problem (in terms of some chosen densities of transitions and final states, including those for which the problem is the most difficult) when the number of states of the automata grows. We also show the time needed when universality is checked using determinisation, complement, and emptiness checking. We see that the



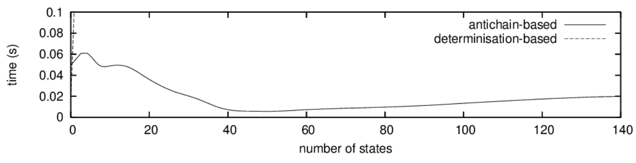
(a) Probability that a tree automaton (TA) with 20 states and some density of transitions and final states is universal



(b) Average times of antichain-based universality checking on TA with 20 states and some density of transitions and final states



(c) Universality checking via determinisation and antichains on TA with selected densities of transitions and final states



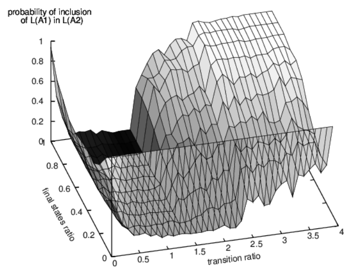
(d) Determinisation-based and antichain-based universality checking on TA from abstract regular tree model checking

Figure 5.3: Experiments with universality checking on tree automata

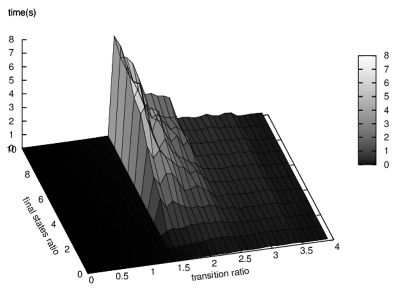
antichain-based approach behaves in a significantly better way. The same conclusion can also be drawn from the results of Fig. 5.3(d) obtained on automata from experimenting with abstract regular tree model checking applied for verifying various procedures manipulating trees presented in Section 5.3.3.

5.3.2 Experiments with Antichain-based Inclusion Checking

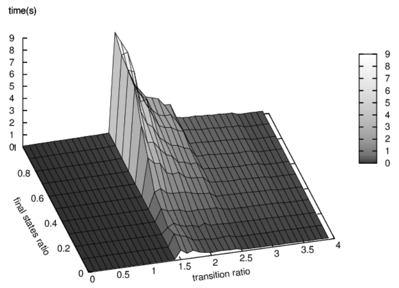
Below, in Fig. 5.4 and Fig. 5.5, we present the results that we have obtained from experimenting with pure antichain-based inclusion checking for tree automata. We first ran our tests on pairs of randomly generated automata having 10 states and different possible densities of transitions and final states. The probability that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds for randomly generated tree automata $\mathcal{A}$ and $\mathcal{B}$ (both having the same densities of transitions and final states) is shown in Fig. 5.4(a). Fig. 5.4(b) then shows how the antichain-based inclusion checking behaves on such automata. We see that its time consumption is naturally growing for automata where the probability of whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds is neither too low nor too high.



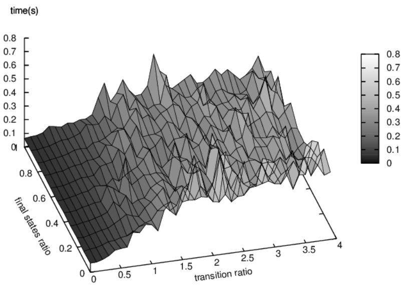
(a) Probability of $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ for tree automata (TA) with 10 states and some density of transitions and final states



(b) Average times of antichain-based inclusion checking on TA with some density of transitions and final states



(c) Antichain-based inclusion checking on TA, $\mathcal{A}$ random, $\mathcal{B}$ with some density of transitions and final states

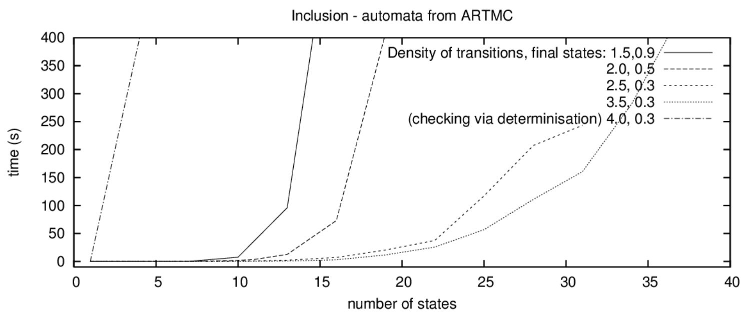


(d) Antichain-based inclusion checking on TA, $\mathcal{B}$ random, $\mathcal{A}$ with some density of transitions and final states

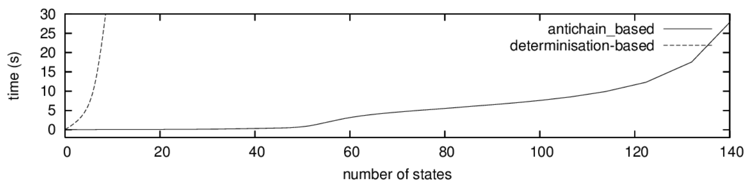
Figure 5.4: Experiments with inclusion checking on tree automata

Fig. 5.4(c) and Fig. 5.4(d) show what happens if either $\mathcal{A}$ or $\mathcal{B}$ is left completely random, and only $\mathcal{B}$ or $\mathcal{A}$, respectively, follows a given density of transitions and final states. The fact that the results in Fig. 5.4(c) follow Fig. 5.4(b), whereas the time consumption in Fig. 5.4(d) is roughly implied by the size of $\mathcal{A}$ (in terms of transitions), implies that the time consumption of the antichain-based inclusion checking is—as expected—influenced much more by the automaton $\mathcal{B}$.

Finally, in Fig. 5.5(a), we show how the running times change for some selected instances of the problem (in terms of some selected densities of transitions and final states, including those for which the problem is the most difficult) when the number of states of the automata starts growing. The figure also shows the time needed when the inclusion checking is based on determinising and complementing $\mathcal{B}$ and checking emptiness of the language $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{B})}$. We see that the antichain-based approach really behaves in a very significantly better way. The same conclusion can then be drawn also from the results shown in Fig. 5.5(b) that we obtained on automata saved from experimenting with abstract regular tree model checking applied for verifying various real-life procedures manipulating trees (cf. Section 5.3.3). In fact, the antichain-based inclusion checking allowed us to implement an abstract regular tree model checking framework entirely based on nondeterministic tree automata which is significantly more efficient than the framework based on deterministic automata.



(a) Determinisation-based and antichain-based inclusion checking on TA with selected densities of transitions and final states



(b) Determinisation-based and antichain-based inclusion checking on TA from abstract regular tree model checking

Figure 5.5: Further experiments with inclusion checking on tree automata

5.3.3 Experiments with Regular Tree Model Checking

We now present our experiments with regular tree model checking that illustrate practical applicability of the language inclusion testing algorithms and the tree automata reduction algorithms from Chapter 4. We will show how the two techniques allow us to build the (abstract) regular tree model checking on nondeterministic tree automata instead of on deterministic ones which greatly improves efficiency of the method.

Nondeterministic Abstract Regular Tree Model Checking. As is clear from the definition of $\hat{\tau}$ in Section 2.5, ARTMC was originally defined for and tested on *minimal deterministic* tree automata (DTA). However, the various experiments done showed that the determinisation step is a significant bottleneck. To avoid it and to implement ARTMC using nondeterministic tree automata (TA), we need the following operations over TA: (1) application of the transition relation τ, (2) union, (3) abstraction and its refinement, (4) intersection with the set of bad configurations, (5) emptiness, and (6) inclusion checking (needed for testing if the abstract reachability computation has reached a fixpoint). Finally, (7) a method to reduce the size of the computed TA is also desirable—$\hat{\tau}(\mathcal{A})$ is then redefined to be the reduced version of the TA obtained from an application of τ on an TA $\mathcal{A}$. We note that the method would in theory work without reduction methods too. However, often hundreds of the steps (1) to (6) are performed within a single verification run, and most of them increases the size of automata[3]. Therefore, good reduction techniques are in fact crucial since the size of automata tends to explode which reduces scalability of the method.

An implementation of Points (1), (2), (4), and (5) is easy. Moreover, concerning Point (3), the abstraction mechanisms of [BHRV06a] can be lifted to work on TA in a straightforward way while preserving their guarantees to be finitary, overapproximating, and the ability to exclude spurious counterexamples. Furthermore, Chapter 4 gives efficient algorithms for reducing TA based on computing suitable simulation equivalences on their states, which covers Point (7). Hence, the last obstacle for implementing nondeterministic ARTMC was Point (6), i.e., the need to efficiently check inclusion on TA. We have solved this problem by Algorithm 6, which allowed us to implement a nondeterministic ARTMC framework in a prototype tool and test it on suitable examples. Below, we present the first very encouraging results that we have achieved. We note that we were so far considering only the pure antichains where the role of simulation within Algorithm 6 is played only by the identity relation[4].

Experiments with Nondeterministic ARTMC. We have implemented the version of ARTMC framework based on nondeterministic tree automata using the Timbuk tree automata library [GVT03] and compared it with an ARTMC implementation based on the same library, but using DTA. In particular, the deterministic ARTMC framework uses determinisation and minimisation after computing the effect of each forward or backward step to try to keep the automata as small as possible and to allow for easy fixpoint checking: The fixpoint checking on DTA is not based on inclusion, but identity checking on the obtained automata (due to the fact that the computed sets are only growing and minimal DTA are canonical). For TA, the tree automata reduction from Chapter 4 that we use does not yield canonical automata, and so the antichain-based inclusion checking is really needed.

We have applied the framework to verify several procedures manipulating dynamic tree-shaped data structures linked by pointers. The trees being manipulated are encoded directly as the trees handled in ARTMC, each node is labelled by the data stored in it and the pointer variables currently pointing to it. All program statements are encoded as (possibly non-structure preserving)

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

[4]We have not yet managed to incorporate simulation enhanced antichain algorithms into the framework of ARTMC. We plan to use them in the further prototype tools that we mention in Section 5.5. We believe that the overall impact of the simulation subsumption technique will be positive, judging from the experience that we have gathered and that is presented in Section 5.4.

Table 5.1: Running times (in sec.) of det. and nondet. ARTMC applied for verification of various tree manipulating programs ($\times$ denotes a too long run or a failure due to a lack of memory)

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

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

tree transducers. The encoding is fully automated. The only allowed destructive pointer updates (i.e., pointer manipulating statements changing the shape of the tree) are tree rotations [CLR89] and addition of new leaf nodes.

We have in particular considered verification of the depth-first tree traversal and the standard procedures for rebalancing red-black trees after insertion or deletion of a leaf node [CLR89]. We have verified that the programs do not manipulate undefined and null pointers in a faulty way. For the procedures on red-black trees, we have also verified that their result is a red-black tree (without taking into account the non-regular balancedness condition). In general, the set of possible input trees for the verified procedures as well as the set of correct output trees were given as tree automata. In the case of the procedure for rebalancing red-black trees after an insertion, we have also used a generator program preceding the tested procedure which generates random red-black trees and a tester program which tests the output trees being correct. Here, the set of input trees contained just an empty tree, and the verification was reduced to checking that a predefined error location is unreachable. The size of the programs ranges from 10 to about 100 lines of pure pointer manipulations.

The results of our experiments on an Intel Xeon processor at 2.7GHz with 16GB of available memory (as in Section 5.3) are summarised in Table 5.1. The predicate abstraction proved to give much better results (therefore we do not consider the finite-height abstraction here). The abstraction was either applied after firing each statement of the program ("full abstraction") or just when reaching a loop point in the program ("restricted abstraction"). The results we have obtained are very encouraging and show a significant improvement in the efficiency of ARTMC based on nondeterministic tree automata. Indeed, the ARTMC framework based on deterministic tree automata has either been significantly slower in the experiments (up to 25-times) or has completely failed (a too long running time or a lack of memory)—the latter case being quite frequent.

5.4 Experiments with Pure versus Simulation Enhanced Antichain Algorithms.

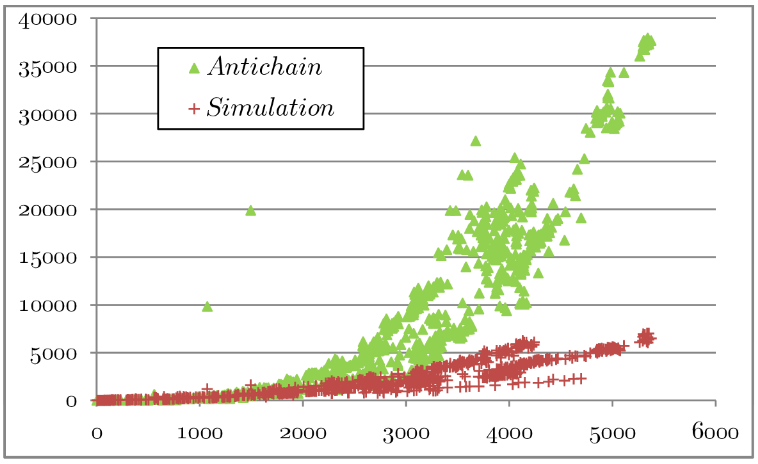
In this section, we describe the experimental result obtained in [ACH$^+$10a] where we compare pure antichain algorithms for FA and TA with simulation enhanced antichain algorithms. Recall that by pure antichain algorithms we mean algorithms published in [WDHR06] for FA and in [BHH$^+$08b] for TA that may be seen as special cases of Algorithms 3, 4, 5, and 6 where the role of simulation relation is played by the identity relation. Notice that in this case, only Optimisation 1 comes to play within Algorithms 3 and 5 for checking universality, and only Optimisation 1(a) applies within Algorithms 4 and 6 for checking language inclusion. Since $\preceq$ is the identity relation, Checking the relation $\preceq^{\forall\exists}$ on sets of states is then replaced be checking subset inclusion.

We concentrated on experiments with inclusion checking, since it is more common than universality checking in various symbolic verification procedures, decision procedures, etc. We compared our approach, parametrised by maximal simulation (or, for tree automata, maximal upward simulation), with the previous pure antichain-based approach of [WDHR06, BHH$^+$08b], and with classical subset-construction-based approach. We implemented all the above in OCaml. We used the algorithm in [HŠ09a] for computing maximal simulations. In order to make the figures easier to read, we often do not show the results of the classical algorithm. The reason is that in all of the experiments, the classical algorithm performed much worse than the other two approaches that these experiments are primarily directed to compare.

We note that we have also done some preliminary experiments with random automata generated according to the framework by Vardi and Tabakov in the same way as in the previous section. Sadly, for this type of automata, the simulation optimisation give almost no speedup. It seems that for the hard areas of the space of settings of parameters of the generator, simulation is very sparse and the speedup that it gives hardly compensates the time needed for computing the simulation itself. On the other hand, for the easy settings, pure antichain algorithms finish too fast and the time needed for computing simulation dominates. Therefore, we decided to perform more experiments with automata that have more structure such as those from the sources described above and which are also closer too real life applications than the random ones. As we will see, for these automata the simulation optimisations really help.

5.4.1 Experiments on FA

For language inclusion checking of FA, we compared the simulation enhanced approach that corresponds to Algorithm 4 against the former pure antichain approach that corresponds to the same algorithm but with the simulation relation being identity. We tested the two on examples generated from the intermediate steps of a tool for abstract regular model checking [BHV04]. In total, we have 1069 pairs of FA generated from different verification tasks, which included verifying a version of the bakery algorithm, a system with a parametrised number of producers and consumers communicating through a double-ended queue, the



(a) Detailed results

Size			Antichain	Simulation
0	-	1000	0.059	0.099
1000	-	2000	1.0	0.7
2000	-	3000	3.6	1.69
3000	-	4000	11.2	3.2
4000	-	5000	20.1	4.79
5000	-		33.7	6.3

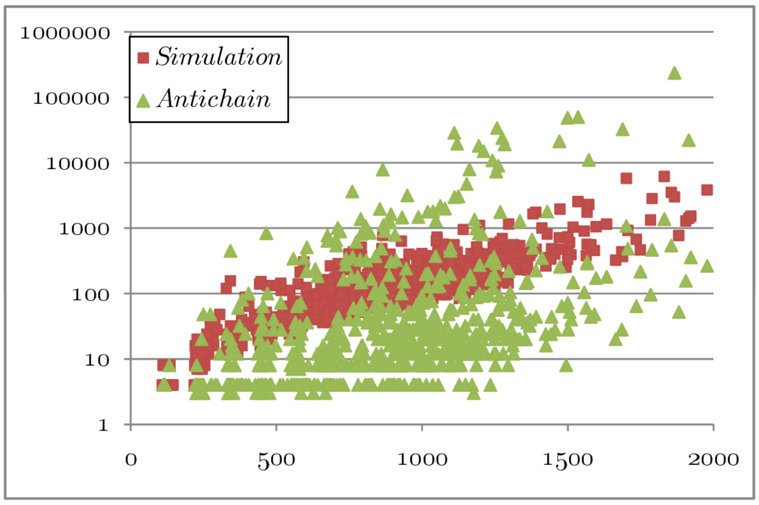
(b) Average execution time for different FA pair sizes (in seconds)

Figure 5.6: Language inclusion checking on FA generated from a regular model checker

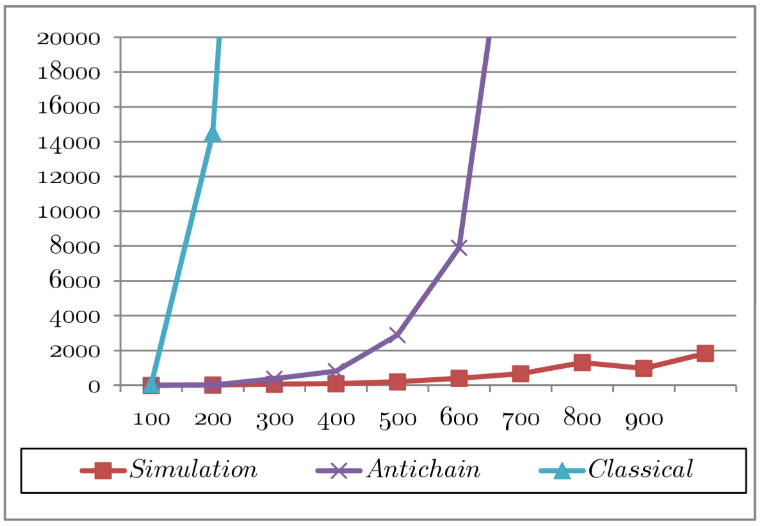
bubble sort algorithm, an algorithm that reverses a circular list, and a Petri net model of the readers/writers protocol (cf. [BHV04, BHMV05] for a detailed description of the verification problems). In Fig. 5.6 (a), the horizontal axis is the sum of the sizes of the pairs of automata[5] whose language inclusion we check, and the vertical axis is the execution time (the time for computing the maximal simulation is included). Each point denotes a result from inclusion testing for a pair of FA. Fig. 5.6 (b) shows the average results for different FA sizes. From the figure, one can see that our approach has a much better performance than the antichain-based one. Also, the difference between our approach and the antichain-based approach becomes larger when the size of the FA pairs increases. If we compare the average results on the smallest 1000 FA pairs, our approach is 60% slower than the the antichain-based approach. For the largest FA pairs (those with size larger than 5000), our approach is 5.32 times faster than the the antichain-based approach. We note that the time needed for computing simulation is always included in the overall running time of the simulation enhanced algorithm.

We also tested our approach using FA generated from random regular ex-

[5]We measure the size of the automata as the number of their states.



(a) Language inclusion does not always hold



(b) Language inclusion always holds

Figure 5.7: Language inclusion checking on FA generated from regular expressions

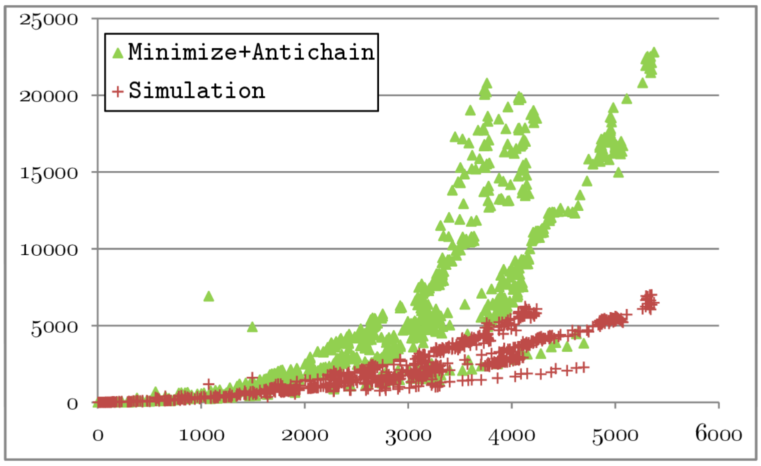


Figure 5.8: Compare the performance of our approach with minimise + antichain

pressions. We have two different tests: (1) language inclusion does not always hold and (2) language inclusion always holds[6]. The result of the first test is in Fig. 5.7(a). In the figure, the horizontal axis is the sum of the sizes of the pairs of automata whose language inclusion we check, and the vertical axis is the execution time (the time for computing the maximal simulation is included). From Fig. 5.7(a), we can see that the performance of our approach is much more stable. It seldom produces extreme results. In all of the cases we tested, it always terminates within 10 seconds. In contrast, the antichain-based approach needs more than 100 seconds in the worst case. The result of the second test is in Fig. 5.7(b) where the horizontal axis is the length of the regular expression and the vertical axis is the average execution time of 30 cases in milliseconds. From Fig. 5.7(b), we observe that our approach has a much better performance than the antichain-based approach if the language inclusion holds. When the length of the regular expression is 900, our approach is almost 20 times faster than the antichain-based approach.

When the maximal simulation relation $\preceq$ is given, a natural way to accelerate the language inclusion checking is to use $\preceq$ to minimise the size of the two input automata by merging $\preceq$-equivalent states. In this case, the simulation relation becomes sparser. A question arises whether our approach has still a better performance than the antichain-based approach in this case. Therefore, we also evaluated our approach under this setting. Here again, we used the FA pairs generated from abstract regular model checking [BHV04]. The results presented at Figure 5.8 show that although the antichain-based approach gains some speed-up (compare with Figure 5.6) when combined with minimisation, it is still slower than our approach. The main reason is that in many cases, simulation holds only in one direction, but not in the other. Our approach can also utilise this type of relation. In contrast, the minimisation algorithm merges only simulation equivalent states.

[6]To get a sufficient number of tests for the second case, we generate two FA $\mathcal{A}$ and $\mathcal{B}$ from random regular expressions, build their union automaton $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$, and test $L(\mathcal{A}) \subseteq L(\mathcal{C})$.

Size	Antichain ($sec.$)	Simulation ($sec.$)	Diff.	# of Pairs
0 – 200	1.05	0.75	140%	29
200 – 400	11.7	4.7	246%	15
400 – 600	65.2	19.9	328%	14
600 – 800	3019.3	568.7	531%	13
800 – 1000	4481.9	840.4	533%	5
1000 – 1200	11761.7	1720.9	683%	10

Table 5.2: Language inclusion checking on TA

5.4.2 Experiments on TA

For language inclusion checking of TA, we tested our approach on 86 tree automata pairs generated from the intermediate steps of a regular tree model checker from Section 5.3.3 while verifying the algorithm of rebalancing red-black trees after insertion or deletion of a leaf node. We were again comparing simulation enhanced antichain approach that corresponds to Algorithm 6 with the pure antichain approach that corresponds to the same algorithm but with the simulation relation being the identity. The results are given in Table 5.2. Our approach has a much better performance when the size of a TA pair is large. For TA pairs of size smaller than 200, our approach is on average 1.39 times faster than the antichain-based approach. However, for those of size above 1000, our approach is on average 6.8 times faster than the antichain-based approach.

5.5 Conclusions and Future Work

We presented algorithms for finite word and tree automata universality and language inclusion checking that combine the antichain principle from [WDHR06] with a use of simulation relations (forward simulation in the case of FA and upward simulation parametrised by identity in the case of TA). The algorithms have been thoroughly tested both on randomly generated automata and on automata obtained from various verification runs of the ARTMC framework. The new algorithms are significantly more efficient than the pure antichain algorithms from [WDHR06] and [BHH$^+$08b].

In the case of TA, we also presented experimental results from our previous work [BHH$^+$08b] on pure antichain tree automata versions of the algorithms from [WDHR06] which preceded the work on their versions improved with simulation presented here. We compare these algorithms with the classical subset construction-based algorithm and we conclude that similarly as shown in [WDHR06] for FA, the antichain technique fundamentally improves performance of universality and language inclusion checking over tree automata. Moreover, using the proposed pure antichain-based inclusion checking algorithm together with our simulation based reduction methods from Chapter 4, we have implemented a complete ARTMC framework based on nondeterministic tree automata and tested it on verification of several real-life pointer-intensive procedures. The results show a very encouraging improvement in the capabilities of the framework.

We are considering several directions of future work. First, our simulation based improvements of antichain algorithms is based on relatively simple and natural principles and we believe that these techniques can be developed for other classes of automata. We have already done the first attempt in [ACC$^+$10a] where we have successfully combined the Ramsey based approach to universality and inclusion checking for Büchi automata with simulations.

Next, we have already proven first results showing that it is possible to design downward tree automata antichain algorithms. These could be then combined with downward tree automata simulation. We believe that in practice, downward algorithms could outperform the upward ones. The upward algorithms suffer from a need of exploring relatively high nondeterministic choice of an upward tree automata run. One dimension of this nondeterminism could be eliminated by a downward algorithm. Moreover, downward simulation is cheaper and often richer than upward simulation parametrised by identity, which could be another advantage of downward algorithms.

Another interesting idea is to try to combine relations in the spirit of our mediated preorder from Chapter 4 with the antichain methods. Mediated preorders are richer than simulations, but imply different yet still interesting properties of runs of automata.

We would like to perform even more experiments, including, e.g., experiments where our most recent techniques will be incorporated into the entire framework of abstract regular (tree) model checking or into some automata-based decision procedures. A work on a BDD based tree automata library (in the style of MONA tree automata library [KM01]) that could make the recent tree automata techniques widely available even for more practical purposes has already started. We hope that this will yield another significant improvement in the tree automata technology allowing for a new generation of tools using tree automata. Finally, we are working on an ARTMC-based tool for verifying pointer manipulating programs that will also use all the recent tree automata techniques. We also expect that the tools will generate meaningful experimental data that will be helpful for further research on finite automata.

6 Simulation-based Reduction of Alternating Büchi Automata

In this chapter, we present the results from our first attempt to adapt our techniques beyond the scope of finite word/tree automata, which was first published in [ACHV09a]. Namely, we focus on simulation-based reduction of alternating Büchi automata inspired by the technique described in Chapter 4.

Alternating Büchi automata (ABA) are succinct state-machine representations of ω-regular languages (regular sets of infinite sequences). They are widely used in the area of formal specification and verification of non-terminating systems. One of the most prominent examples of the use of ABA is the complementation of nondeterministic Büchi automata [KV01]. It is an essential step of the automata-theoretic approach to model checking when the specification is given as a positive Büchi automaton [Var07] and also learning based model checking for liveness properties [FCC$^+$08]. The other important usage of ABA is as the intermediate data structure for translating a linear temporal logic (LTL) specification to an automaton [GO01].

However, because of the compactness of ABA[1], the algorithms that work on them are usually of high complexity. For example, both the complementation and the LTL translation algorithms transform an intermediate ABA to an equivalent NBA. The transformation is exponential in the size of the input ABA. Hence, one may prefer to reduce the size of the ABA (with some relatively cheaper algorithm) before giving it to the exponential procedure.

In the study of Fritz and Wilke, simulation-based minimisation is proven as a very effective tool for reducing the size of ABA [FW05]. However, they considered only *forward* simulation relations. Inspired by our work on tree automata reduction methods, we introduce also a notion of *backward* simulation (parametrised by forward simulation) that can be used for reducing the size of ABA as well. As will be explained in Section 6.2, similarly as for tree automata upward simulation, quotienting wrt. *backward* simulation (i.e., simplifying the automaton by collapsing backward simulation equivalent states) does not preserve the language, however, backward simulation can be used for quotienting in combination with forward simulation. In fact, we will arrive to an alternating automata equivalent of the tree automata notion of *mediated equivalence* from Chapter 4.

We evaluate the performance of minimising ABA with mediated equivalence is evaluated on a large set of experiments. In the experiments, we apply different simulation-based minimisation approaches to improve the complementation algorithm of nondeterministic Büchi automata. The experimental results show that the minimisation using mediated preorder significantly outperforms the

[1]ABA's are exponentially more succinct than nondeterministic Büchi automata.

minimisation using forward simulation. To be more specific, on average, mediated minimisation results in a 30% better reduction in the number of states and 50% better reduction in the number of transitions than forward minimisation on the intermediate ABA. Moreover, in the complemented nondeterministic Büchi automata, mediated minimisation results in a 100% better reduction in the number of states and 300% better reduction in the number of transitions than forward minimisation.

6.1 Basic Definitions

Given a finite set X, we use X^* to denote the set of all finite words over X and X^ω for the set of all infinite words over X. The empty word is denoted ϵ and $X^+ = X^* \setminus \{\epsilon\}$. The concatenation of a finite word $u \in X^*$ and a finite or infinite word $v \in X^* \cup X^\omega$ is denoted by uv. For a word $w \in X^* \cup X^\omega$, $|w|$ is the length of w ($|w| = \infty$ if $w \in X^\omega$), w_i is the ith letter of w and w^i the ith prefix of w (the word u with $w = uv$ and $|u| = i$). $w^0 = \epsilon$. The concatenation of a finite word u and a set $S \subseteq X^* \cup X^\omega$ is defined as $uS = \{uv \mid v \in S\}$.

An *alternating Büchi automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ where Σ is a finite alphabet, Q is a finite set of states, $\iota \in Q$ is an initial state, $\alpha \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to 2^{2^Q}$ is a total transition function. A *transition* of $\mathcal{A}$ is of the form $p \xrightarrow{a} P$ where $P \in \delta(q, a)$.

A *tree T over Q* is a subset of Q^+ that contains all nonempty prefixes of each one of its elements (i.e., $T \cup \{\epsilon\}$ is prefix-closed). Furthermore, we require that T contains exactly one $r \in Q$, the *root of T*, denoted *root*(T). We call the elements of Q^+ *paths*. For a path πq, we use *leaf*(πq) to denote its last element q. Define the set *branches*$(T) \subseteq Q^+ \cup Q^\omega$ such that $\pi \in$ *branches*(T) iff T contains all prefixes of π and π is not a proper prefix of any path in T. In other words, a *branch of T* is either a maximal path of T, or it is a word from Q^ω such that T contains all its nonempty prefixes. We use $succ_T(\pi) = \{r \mid \pi r \in T\}$ to denote the set of successors of a path π in T, and *height*(T) to denote the length of the longest branch of T. A tree U over Q is a *prefix of T* iff $U \subseteq T$ and for every $\pi \in U$, $succ_U(\pi) = succ_T(\pi)$ or $succ_U(\pi) = \emptyset$. The *suffix of T* defined by a path πq is the tree $T(\pi q) = \{q\psi \mid \pi q\psi \in T\}$.

Given a word $w \in \Sigma^\omega$, a tree T over Q is a *run of $\mathcal{A}$ on w*, if for every $\pi \in T$, *leaf*$(\pi) \xrightarrow{w_{|\pi|}} succ_T(\pi)$ is a transition of $\mathcal{A}$. Finite prefixes of T are called *partial runs on w*. A run T of $\mathcal{A}$ over w is *accepting* iff every infinite branch of T contains infinitely many accepting states. A word w is *accepted* by $\mathcal{A}$ from a state $q \in Q$ iff there exists an accepting run T of $\mathcal{A}$ over w with *root*$(T) = q$. The *language of a state $q \in Q$ in $\mathcal{A}$*, denoted $\mathcal{L}_\mathcal{A}(q)$, is the set of all words accepted by $\mathcal{A}$ from q. Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(\iota)$ is the *language of $\mathcal{A}$*. For simplicity of presentation, we assume in the rest of the paper that δ never allows a transition of the form $p \xrightarrow{a} \emptyset$. This means that no run can contain a finite branch. Any automaton can be easily transformed into one without such transitions by adding a new accepting state q with $\delta(q, a) = \{\{q\}\}$ for every $a \in \Sigma$ and replacing every transition $p \xrightarrow{a} \emptyset$ by $p \xrightarrow{a} \{q\}$.

We note that for technical reasons, we use a simpler definition of a tree and a run of an alternating automaton than the usual one (e.g., [KV01] or Chapter 4). A tree is usually defined as a prefix closed subset of $\mathbb{N}^*$ and a run is then a map r that assigns a state to every element (node) of a tree. This definition allows existence of nodes with more than one immediate successor labelled by the same state and successors of a node are ordered. However, order as well as number of occurrences of a state in the role of a successor of a parent state has no relevance for semantics of an ABA. From this point of view, it is more convenient to define runs simply as unordered trees.

6.2 Simulation Relations

In this section, we give the definitions of forward and backward simulation over ABA and discuss some of their properties. The notion of backward simulation is inspired by a similar tree automata notion studied in Chapter 4—namely, the upward simulation parametrised by a downward simulation (the connection between tree automata and ABA follows from the fact that the runs of ABA are in fact trees).

For the rest of the section, we fix an ABA $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$. We define relations $\preceq_\alpha$ and $\preceq_\iota$ on Q s.t. $q \preceq_\alpha r$ iff $q \in \alpha \implies r \in \alpha$ and $q \preceq_\iota r$ iff $q = \iota \implies r = \iota$. For a binary relation $\preceq$ on a set X, the relation $\preceq^{\forall\exists}$ on subsets of X is defined as $Y \preceq^{\forall\exists} Z$ iff $\forall z \in Z.\ \exists y \in Y.\ y \preceq z$, i.e., iff the upward closure of Z wrt. $\preceq$ is a subset of the upward closure of Y wrt. $\preceq$.

Forward Simulation. A *forward simulation* on $\mathcal{A}$ is a relation $\preceq_F \subseteq Q \times Q$ such that $p \preceq_F r$ implies that (i) $p \preceq_\alpha r$ and (ii) for all $p \xrightarrow{a} P$, there exists a $r \xrightarrow{a} R$ such that $P \preceq_F^{\forall\exists} R$.

For the basic properties of forward simulation, we rely on the work [GKSV03] by Gurumurthy et al. In particular, (i) there exists a unique maximal forward simulation $\preceq_F$ on $\mathcal{A}$ called *forward simulation preorder* which is reflexive and transitive, (ii) for any $q, r \in Q$ such that $q \preceq_F r$, it holds that $\mathcal{L}_\mathcal{A}(q) \subseteq \mathcal{L}_\mathcal{A}(r)$, and (iii) quotienting wrt. $\preceq_F \cap \preceq_F^{-1}$ preserves the language of $\mathcal{A}$.

Backward Simulation. Let $\preceq_F$ be a forward simulation on $\mathcal{A}$. A *backward simulation* on $\mathcal{A}$ parametrised by $\preceq_F$ is a relation $\preceq_B \subseteq Q \times Q$ such that $p \preceq_B r$ implies that (i) $p \preceq_\iota r$, (ii) $p \preceq_\alpha r$, and (iii) for all $q \xrightarrow{a} P \cup \{p\}, p \notin P$, there exists a $s \xrightarrow{a} R \cup \{r\}, r \notin R$ such that $q \preceq_B s$ and $P \preceq_F^{\forall\exists} R$. The lemma below describes basic properties of backward simulation.

Lemma 6.1. *For any reflexive and transitive forward simulation $\preceq_F$ on $\mathcal{A}$, there exists a unique maximal backward simulation $\preceq_B$ on $\mathcal{A}$ parametrised by $\preceq_F$ that is reflexive and transitive.*

Proof. The proof is an analogy of the proof of Lemma 4.2.

Union: Given two backward simulations $\preceq_B^1$ and $\preceq_B^2$ induced by $\preceq_F$, we want to prove that $\preceq_B = \preceq_B^1 \cup \preceq_B^2$ is also a backward simulation induced by $\preceq_F$. Let $p \preceq_B r$ for some $p, r \in Q$, then either $p \preceq_B^1 r$ or $p \preceq_B^2 r$. Assume without loss of generality that $p \preceq_B^1 r$. Then, from the definition of backward simulation, whenever $p' \xrightarrow{a} P \cup \{p\}, p \notin P$, then there is a rule $r' \xrightarrow{a} R \cup \{r\}, r \notin R$, $p' \preceq_B^1 r'$, and $P \preceq_F^{\forall\exists} R$. As $\preceq_B^1 \subseteq \preceq_B$ gives $p' \preceq_B r'$, $\preceq_B$ fulfils the definition of backward simulation induced by $\preceq_F$.

Reflexive closure: It can be seen from the definition that the identity is a backward simulation induced by $\preceq_F$ for any forward simulation $\preceq_F$. Therefore, from the closure under union, the union of the identity and any backward simulation induced by $\preceq_F$ is a backward simulation induced by $\preceq_F$.

Transitive closure: Let $\preceq_B$ be a backward simulation induced by $\preceq_F$ and let $\preceq_B^T$ be its transitive closure. Let $p^1 \preceq_B^T p^m$ and $r^1 \xrightarrow{a} P^1 \cup \{p^1\}, p^1 \notin P^1$. Apparently, $p^1 \preceq_\alpha p^m$ since $\preceq_\alpha$ is a transitive subset of $\preceq_B$. From $p^1 \preceq_B^T p^m$, we have that there are states $p^1, \ldots, p^m$ such that $p^1 \preceq_B p^2 \preceq_B \cdots \preceq_B p^m$. Therefore, there are also rules $r^2 \xrightarrow{a} P^2 \cup \{p^2\}, \ldots, r^m \xrightarrow{a} P^m \cup \{p^m\}$ with $p^2 \notin P^2, \ldots, p^m \notin P^m$, $r^1 \preceq_B \cdots \preceq_B r^m$, and $P^1 \preceq_F^{\forall\exists} P^2 \preceq_F^{\forall\exists} \cdots \preceq_F^{\forall\exists} P^m$. Thus, by definition of $\preceq_B^T$, we have $r^1 \preceq_B^T r^m$, and by transitivity of $\preceq_F^{\forall\exists}$, $P^1 \preceq_F^{\forall\exists} P^m$. Therefore, $\preceq_B^T$ fulfils the definition of a backward simulation induced by $\preceq_F$. $\square$

By Lemma 6.1, for a reflexive and transitive forward simulation $\preceq_F$, there is a unique maximal upward simulation parametrised by $\preceq_F$ and it is a preorder. We call it the *backward simulation preorder* on $\mathcal{A}$ parametrised by $\preceq_F$. Our backward simulation is a close analogy of tree automata upward simulation. Similarly as upward simulation, backward simulation cannot be directly used for quotienting (below we give an example of an automaton where quotienting using backward simulation does not preserve language). However, in Section 6.3.1, we show that backward simulation can be combined with forward simulation into a mediated equivalence (in the same way as tree automata upward simulation can be combined with downward simulation) that can be used for quotienting.

Example 6 (backward simulation cannot be used for quotienting)**.** *Consider the ABA* $\mathcal{A} = (\{a,b\}, \{s_0,s_1,s_2,s_3,s_4,s_5,s_6\}, s_0, \delta, \{s_0,s_1,s_2,s_3,s_4,s_5,s_6\})$ *where*

$$\begin{array}{llll}
s_0 \xrightarrow{a} \{s_4\}, & s_1 \xrightarrow{b} \{s_2,s_5\}, & s_2 \xrightarrow{b} \{s_2,s_3\}, & s_5 \xrightarrow{b} \{s_0\}, \\
s_0 \xrightarrow{a} \{s_1\}, & s_1 \xrightarrow{b} \{s_1,s_3\}, & s_3 \xrightarrow{a} \{s_0\}, & s_6 \xrightarrow{a} \{s_0\} \\
s_0 \xrightarrow{b} \{s_0\}, & & s_4 \xrightarrow{b} \{s_4,s_6\}, &
\end{array}$$

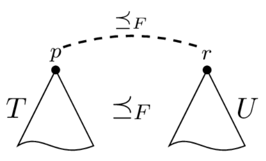
are transitions of $\mathcal{A}$. The maximal forward simulation relation $\preceq_F$ in $\mathcal{A}$ is

$$\{(s_0,s_0),(s_1,s_0),(s_1,s_1),(s_1,s_5),(s_2,s_0),(s_2,s_1),(s_2,s_2),(s_2,s_4),\\
(s_2,s_5),(s_3,s_3),(s_3,s_6),(s_4,s_0),(s_4,s_1),(s_4,s_2),(s_4,s_4),(s_4,s_5),\\
(s_5,s_0),(s_5,s_5),(s_6,s_3),(s_6,s_6)\}.$$

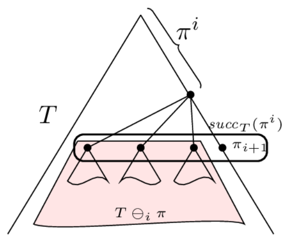
The maximal backward simulation relation $\preceq_B$ parametrised with $\preceq_F$ is

$$\{(s_0,s_0),(s_1,s_1),(s_1,s_4),(s_2,s_2),(s_3,s_3),(s_4,s_1),(s_4,s_4),(s_5,s_2),\\
(s_5,s_3),(s_5,s_5),(s_5,s_6),(s_6,s_2),(s_6,s_3),(s_6,s_5),(s_6,s_6)\}.$$

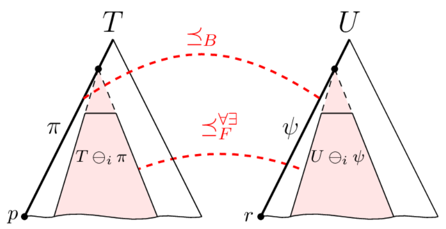
If we collapse states of $\mathcal{A}$ wrt. $\preceq_M$ (i.e., the two sets of states $\{s_1,s_4\},\{s_5,s_6\}$ are collapsed), we will get the following alternating Büchi automaton $\mathcal{A}' =$



(a) Lemma 6.2



(b) The notation $T \ominus_i \pi$



(c) Lemma 6.3

Figure 6.1: Illustration of the lemmas

$(\{a,b\},\{s_0,s_1,s_2,s_3,s_4\},s_0,\delta,\{s_0,s_1,s_2,s_3,s_4\})$ *where*

$$\begin{array}{llll} s_0 \xrightarrow{a} \{s_1\}, & s_1 \xrightarrow{b} \{s_2,s_4\}, & s_2 \xrightarrow{b} \{s_2,s_3\}, & s_4 \xrightarrow{a} \{s_0\}, \\ s_0 \xrightarrow{b} \{s_0\}, & s_1 \xrightarrow{b} \{s_1,s_4\}, & s_3 \xrightarrow{a} \{s_0\}, & s_4 \xrightarrow{b} \{s_0\} \\ & s_1 \xrightarrow{b} \{s_1,s_3\}, & & \end{array}$$

are transitions of $\mathcal{A}'$. Note that $\mathcal{A}'$ accepts the word ab^ω, but $\mathcal{A}$ does not. $\square$

6.2.1 Runs and Simulations

We now formulate connections between simulations and runs of ABA that are fundamental for our further reasoning. Let $\preceq_F$ and $\preceq_B$ be forward and backward simulations on $\mathcal{A}$, which are both reflexive and transitive. For every $x \in \{B,F,\alpha\}$, we extend the relation $\preceq_x$ to $Q^+ \times Q^+$ such that for $\pi,\psi \in Q^+$, $\pi \preceq_x \psi$ iff $|\pi| = |\psi|$ and for all $1 \leq i \leq |\pi|$, $\pi_i \preceq_x \psi_i$. We say that ψ forward simulates π, ψ backward simulates π, or ψ is more accepting than π when $\pi \preceq_F \psi$, $\pi \preceq_B \psi$, or $\pi \preceq_\alpha \psi$, respectively. This notation is further extended to trees. For trees T,U over Q and for $x \in \{\alpha,F\}$, we write, $T \preceq_x U$ if $branches(T) \preceq_x^{\forall\exists} branches(U)$. Similarly, we say that U forward simulates T, or U is more accepting than T when $T \preceq_F U$, or $T \preceq_\alpha U$, respectively. Note that $\preceq_x$ is reflexive and transitive for all the variants of $x \in \{F,B,\alpha\}$ defined over states, paths, or trees (this follows from the assumption that the original relations $\preceq_F$ and $\preceq_B$ on states are reflexive and transitive). Moreover, $\preceq_B \subseteq \preceq_\alpha$, $\preceq_B \subseteq \preceq_\iota$, and $\preceq_F \subseteq \preceq_\alpha$.

Lemma 6.2. *For any $p,r \in Q$ with $p \preceq_F r$ and a partial run T of $\mathcal{A}$ on $w \in \Sigma^\omega$ with the root p, there is a partial run U of $\mathcal{A}$ on w with the root r such that $T \preceq_F U$.*

Proof. We prove the lemma by induction on $\mathit{height}(T)$. In the base case when $T = \{p\}$, it is sufficient to take $U = \{r\}$. Suppose now that the lemma holds for every word u and for every partial run V of $\mathcal{A}$ on u such that $\mathit{height}(V) < \mathit{height}(T)$. From $p \preceq_F r$, there is a transition $r \xrightarrow{w_1} R$ of $\mathcal{A}$ where $\mathit{succ}_T(p) \preceq_F^{\forall\exists} R$. Observe that $T = \{p\} \cup \bigcup_{p' \in \mathit{succ}_T(p)} pT(p')$ where for each $p' \in \mathit{succ}_T(p)$, $T(p')$ is a partial run of $\mathcal{A}$ with the root p' on the word v such that $w = w_1 v$. Notice that $\mathit{height}(T(p')) < \mathit{height}(T)$. The induction hypothesis now can be applied to every triple $p' \in \mathit{succ}_T(p), r' \in R, T(p')$ with $p' \preceq_F r'$. It gives us a partial run $U_{r'}$ of $\mathcal{A}$ on v with $\mathit{root}(U_{r'}) = r'$ such that $T(p') \preceq_F U_{r'}$. The run U with the required properties is then constructed by plugging the runs $U_{r'}, r' \in R$, to r, i.e., $U = \{r\} \cup \bigcup_{r' \in R} rU_{r'}$. □

We will need to inspect the connection between runs and backward simulation in a relatively detailed way. For this, we introduce to following notation. Given a tree T over Q, $\pi \in T$, and $1 \leq i \leq |\pi|$, the set $T \ominus_i \pi$ is the union of branches of suffix trees $T(\pi^i q), q \in \mathit{succ}_T(\pi^i)$, with the branches of the suffix tree $T(\pi^{i+1})$ excluded. Formally, let $Q^i = \mathit{succ}_T(\pi^i) \setminus \{\pi_{i+1}\}$ be the set of all successors of π^i in T without the successor continuing in π. Then $T \ominus_i \pi = \bigcup_{q \in Q^i} \mathit{branches}(T(\pi^i q))$ (notice that if $i = 0$, then $T \ominus_i \pi = \emptyset$).

Lemma 6.3. *For any $p, r \in Q$ with $p \preceq_B r$, a partial run T of $\mathcal{A}$ on $w \in \Sigma^\omega$ and $\pi \in \mathit{branches}(T)$ with $\mathit{leaf}(\pi) = p$, there is a partial run U of $\mathcal{A}$ on w and $\psi \in \mathit{branches}(U)$ with $\mathit{leaf}(\psi) = r$ such that $\pi \preceq_B \psi$, and for all $1 \leq i \leq |\pi|$, $T \ominus_i \pi \preceq_F^{\forall\exists} U \ominus_i \psi$.*

Proof. By induction on the length of π. In the base case, when $\pi = p$ and $T = \{p\}$, it is sufficient to take $U = \{r\}$ and $\psi = r$. Suppose now that $\pi \neq p$ and that the lemma holds for every partial run T' of $\mathcal{A}$ on w, states $p', r' \in Q$ such that $p' \preceq_B r'$, and every $\pi' \in \mathit{branches}(T')$ with $\mathit{leaf}(\pi') = p'$ and $|\pi'| < |\pi|$.

For the induction step, let $\pi = \pi' p$ and let $\mathit{succ}_T(\pi') = P \cup \{p\}, p \notin P$. By the definition of $\preceq_B$, there is a transition $s \xrightarrow{w_{|\pi|}} R \cup \{r\}, r \notin R$ of $\mathcal{A}$ such that $\mathit{leaf}(\pi') \preceq_B s$ and $P \preceq_F^{\forall\exists} R$. Let $T' = T \setminus \{\pi\} \setminus \bigcup_{p' \in P} \pi' T(\pi' p')$. Then T' is a partial run of $\mathcal{A}$ on w and $\pi' \in \mathit{branches}(T')$, $|\pi'| < |\pi|$, and therefore we can apply induction hypothesis to T', $\mathit{leaf}(\pi')$, s, and π'. This gives us a partial run U' of $\mathcal{A}$ on w with $\psi' \in \mathit{branches}(U')$ such that $\mathit{leaf}(\psi') = s$, $\pi' \preceq_B \psi'$ and for each $1 \leq j \leq |\pi'|$, $T' \ominus_j \pi' \preceq_F^{\forall\exists} U' \ominus_j \psi'$. For every $p' \in \mathit{succ}_T(\pi')$, $T(\pi' p')$ is a partial run of $\mathcal{A}$ with the root p' on the suffix v of w such that $w = uv, |u| = |\pi| - 1$. We can apply Lemma 6.2 to the triples $r' \in R, p' \in P, T(\pi' p')$ with $p' \preceq_F r'$. This gives us for each $r' \in R$ a run $U_{r'}$ of $\mathcal{A}$ on v with $\mathit{root}(U_{r'}) = r'$ such that there is some $p' \in P$ with $T(\pi' p') \preceq_F U_{r'}$. Now we construct a run U and a path ψ with the required properties by plugging r and runs $U_{r'}, r' \in R$ to the path ψ' in U', i.e., $\psi = \psi' r$ and $U = U' \cup \{\psi\} \cup \bigcup_{r' \in R} \psi' U_{r'}$. (To see that U really satisfies the required properties, observe the following: (i) As $U \ominus_{|\pi'|} \psi = \bigcup_{r' \in R} \mathit{branches}(U_{r'})$ and $T \ominus_{|\pi'|} \pi = \bigcup_{p' \in P} \mathit{branches}(T(\pi' p'))$, and because for each $r' \in R$, there is $p' \in P$ with $T(\pi' p') \preceq_F U_{r'}$, we have that $T \ominus_{|\pi'|} \pi \preceq_F^{\forall\exists} U \ominus_{|\pi'|} \psi$. (ii) For all $1 \leq j < |\pi'|$, $T \ominus_j \pi = T' \ominus_j \pi' \preceq_F^{\forall\exists} U' \ominus_j \psi' = U \ominus_j \psi$.). □

6.3 Mediated Equivalence and Quotienting

Here, we discuss the possibility of an indirect use of backward simulation for simplifying ABA via quotienting. We will introduce an alternating Büchi automata variant of the mediated preorder from Chapter 4 that is a combination of forward and backward simulation suitable for quotienting.

6.3.1 The Notion and Intuition of Mediated Equivalence

We again use the concept of "jumping runs" based on the observation that quotienting an automaton wrt. some equivalence allows a run that arrives to some state to *jump* to equivalent state and continue from there. Alternatively, this can be viewed as *extending* the source state of the jump by the outgoing transitions of the target state[2]. The equivalence must have the property that the language is not increased even when the jumps (or, alternatively, transition extensions) are allowed. It turns out that forward and backward simulation can be combined into a suitable relation in the same way as downward and upward simulation in Chapter 4. This is, we will define the mediated preorder $\preceq_M$ as a suitable transitive fragment of $\preceq_F \circ \preceq_B^{-1}$ and show that allowing jumping to mediated smaller states does not affect the language. It follows that quotienting wrt. mediated equivalence (the largest symmetric fragment of $\preceq_M$) preserves language too.

The intuition behind allowing a run to jump from a state r to a state q that are related by a mediated preorder is very similar to the one given in Chapter 4. The relation $q \preceq_F \circ \preceq_B^{-1} r$ guarantees the existence of the so called *mediator*, which is a state s such that $q \preceq_F s \preceq_B^{-1} r$ (see Figure 6.2(a)). The state s can be reached in the same way and in the same context[3] as r, and, at the same time, the automaton can continue from s in the same way as from q. Hence, intuitively, the newly allowed run based on the jump from r to q does not add anything to the language because it can anyway be realised through s without jumps.

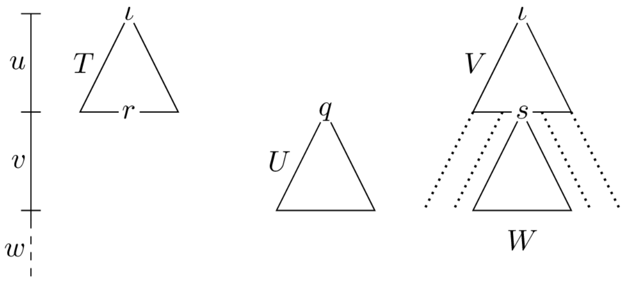
Similarly as in the case of tree automata, jumping cannot be allowed between all pairs of states from $\preceq_F \circ \preceq_B^{-1}$. We will have to restrict ourselves only to its fragments $\preceq_M$ that are *preorders* and are also *forward extensible*, which means that if $q_1 \preceq_M q_2 \preceq_F q_3$, then $q_1 \preceq_M q_3$.

The reason for this is that we were so far taking into account only one isolated jump, however, nothing prevents another jumps from occurring in the context or below the marked occurrence of r. This is problematic since the relations $q \preceq_F s \preceq_B^{-1} r$ are guaranteed only when no further jumps are allowed. The forward extensibility is required to ensure the mechanism to work with arbitrary many jumps. We describe the potential problems when $\preceq_M$ is not forward extensible (see Figure 6.2(b) for the illustration).

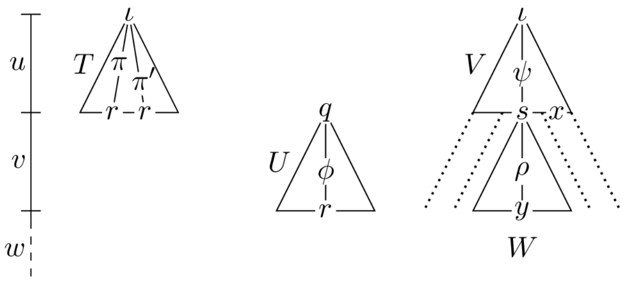
Problem (i): The first problem will arise if there is a branch ϕ of U with

[2]The first view is better when explaining the intuition whereas the other is easier to be used in proofs.

[3]If a state s is a leaf of a partial run, then by a *context* of s we mean all the other leaves of the partial run.



(a) The Mediator



(b) Potential Problems

Figure 6.2: Basic Intuition Behind Mediated Equivalence

leaf$(\phi) = r$. Here, apart from interconnecting T and U, r can use its new transitions also at the end of $\pi\phi$ and connect another copy of U to the end of $\pi\phi$. Suppose that all leaves of T except r accept vvw and that all leaves of U except r accept vw. Then this enables a new accepting run on the word $uvvw$. In this case, the existence of the mediator s is not a guarantee that some accepting run on $uvvw$ was possible before adding transitions to r.

Problem (ii): Another problem may arise if there are two (or more) branches in T ending by r. Here we use the two branches π and π' in Figure 6.2(b) as an example. To construct an accepting run on uvw from T, r has to use the transitions of q at the end of π as well as at the end of π' to connect U to T in the both places. But partial run V "covers" only one of the two occurrences of r. There may be a leaf x of V different from s for which r is the only leaf in T with $r \preceq_F x$. Therefore, x needs not accept vw as there is no guaranteed relation between q and x. In this case V is not a prefix of an accepting run on uvw and uvw need not be in $\mathcal{L}(\mathcal{A})$.

We will show how the two problems can be solved by requiring $\preceq_M$ to be a forward extensible preorder.

In the case of Problem (i), if y uses transitions of q to accept vw, then W becomes a prefix of an accepting run on vvw and thus V becomes a prefix of a new accepting run on $uvvw$. We know that $r \preceq_F y$. Thus, by forwards extensibility, $q \preceq r \preceq_F y$ gives $q \preceq y$, which implies that there is a mediator for q and y. Observe that y used transitions of q just once. Therefore, by an analogical argument by which we derived that $\mathcal{A}$ accepts uvw in the first case when r used the new transitions only once, we can here derive that there is an accepting run of $\mathcal{A}$ on $uvvw$ which does not involve new transitions.

In the case of Problem (ii), if x uses the transitions of q to accept vw, V becomes a prefix of a new accepting run on uvw. We know that $r \preceq_F x$ and thus by forward extensibility $q \preceq r \preceq_F x$ gives $q \preceq x$, which means that there is a mediator for q and x. Similarly as in the previous case, since x used the transitions of q only once, we can derive that there exists an accepting run of $\mathcal{A}$ on uvw that does not involve new transitions.

The argumentation from the two above paragraphs can be used inductively for a run where r uses transitions of q arbitrarily many times.

Mediated Preorder and Equivalence. We formally define mediated preorder for ABA analogically as we have defined it in Chapter 4 for tree automata. Consider a reflexive and transitive forward simulation $\preceq_F$ on $\mathcal{A}$, and a reflexive and transitive backward simulation $\preceq_B$ induced by $\preceq_F$. Recall the relation combination operator $\oplus$ defined in Chapter 4. We call the relation $\preceq_M = \preceq_F \oplus \preceq_B^{-1}$ a *mediated preorder induced by* $\preceq_F$ *and* $\preceq_B$ and $\equiv_M = \preceq_M \cap \preceq_M^{-1}$ a *mediated equivalence induced by* $\preceq_F$ *and* $\preceq_B$. By Lemma 4.7, $\preceq_M$ is a unique maximal preorder satisfying $\preceq_F \subseteq \preceq_F \oplus \preceq_B^{-1} \subseteq \preceq_F \circ \preceq_B^{-1}$.

Ambiguity. To make the mediated equivalence applicable, we must pose one more requirement. Namely, we require that the transitions of the given ABA are not $\preceq_F$*-ambiguous*, meaning that no two states on the right hand side of a transition are forward equivalent. Intuitively, allowing such transitions goes against the spirit of the backward simulation. For a mediator p to backward simulate a state r wrt. rules $\rho_1 : p' \xrightarrow{a} P \cup \{p\}, p \notin P$, and $\rho_2 : r' \xrightarrow{a} R \cup \{r\}, r \notin R$, it must be the case that each state x in the context P of p within ρ_1 is less restrictive (i.e., forward bigger) than some state y in the context R of r within ρ_2. The state r itself is not taken into account when looking for y because we aim at extending its behaviour by collapsing (and it could then become less restrictive than the appropriate x). In the case of $\preceq_F$-ambiguity, the spirit of this restriction is in a sense broken since the forward behaviour of r may still be taken into account when checking that the context of p is less restrictive than that of r. This is because the behaviour of r appears in R as the behaviour of some other forward equivalent state r'' too. Consequently, r and r'' may back up each other in a circular way when checking the restrictiveness of the contexts within the construction of the backward simulation. Both of them can then seem extensible, but once their behaviour gets extended, the restriction of their context based on their own original behaviour is lost, which may then increase the language (an example of such a scenario is given below). However, in Section 6.4, we show that $\preceq_F$-ambiguity can be efficiently removed.

Example 7 (mediated minimization cannot be used on an ambiguous ABA)**.** *Consider the following ABA* $\mathcal{A} = (\{a,b\}, \{s_0,s_1,s_2,s_3,s_4\}, s_0, \delta, \{s_4\})$ *where*

$$\begin{aligned} &s_0 \xrightarrow{a} \{s_1,s_2,s_3\}, && s_3 \xrightarrow{b} \{s_4\},\\ &s_1 \xrightarrow{b} \{s_4\}, && s_3 \xrightarrow{a} \{s_1,s_2,s_3\},\\ &s_2 \xrightarrow{b} \{s_4\}, && s_4 \xrightarrow{a} \{s_4\} \end{aligned}$$

are transitions of $\mathcal{A}$. The maximal forward simulation relation $\preceq_F$ in $\mathcal{A}$ is

$$\{(s_0,s_0),(s_0,s_3),(s_1,s_1),(s_1,s_2),(s_1,s_3),\\ (s_2,s_1),(s_2,s_2),(s_2,s_3),(s_3,s_3),(s_4,s_4)\}.$$

From $s_1 \equiv_F s_2$ and the transition $s_0 \xrightarrow{a} \{s_1,s_2,s_3\}$ we can find that $\mathcal{A}$ is $\preceq_F$-ambiguous. The maximal backward simulation relation $\preceq_B$ parametrised with $\preceq_F$ is

$$\{(s_0,s_0),(s_1,s_1),(s_1,s_2),(s_1,s_3),(s_2,s_1),\\ (s_2,s_2),(s_2,s_3),(s_3,s_1),(s_3,s_2),(s_3,s_3),(s_4,s_4)\}$$

and the mediated preorder $\preceq_M$ is

$$\{(s_0,s_0),(s_0,s_1),(s_0,s_2),(s_0,s_3),(s_1,s_1),(s_1,s_2),(s_1,s_3),\\ (s_2,s_1),(s_2,s_2),(s_2,s_3),(s_3,s_1),(s_3,s_2),(s_3,s_3),(s_4,s_4)\}.$$

If we collapse states wrt. $\preceq_M$ (i.e., merge the three states s_1, s_2, and s_3), we will get the following ABA $\mathcal{A}' = (\{a,b\},\{s_0,s_1,s_2\},s_0,\delta,s_2)$ where

$$s_0 \xrightarrow{a} \{s_1\}, s_1 \xrightarrow{a} \{s_1\}, s_1 \xrightarrow{b} \{s_2\}, s_2 \xrightarrow{a} \{s_2\}$$

are transitions of $\mathcal{A}'$. Note that $\mathcal{A}'$ accepts the word $aaba^\omega$, but $\mathcal{A}$ does not. □

6.3.2 Quotienting Automata According to Mediated Equivalence Preserves Language

In this section, we give a formal proof that under the assumption that $\mathcal{A}$ is $\preceq_F$-unambiguous, quotienting with respect to mediated equivalence preserves the language. The proof roughly follows the pattern of the proof in Chapter 4 that quotienting tree automata according to the mediated equivalence preserves language. However, the fact that we are dealing with infinite tree runs with the Büchi accepting condition and that two accepting runs on the same word need not be isomorphic makes the argument significantly more complicated.

Quotient Automata versus Extended Automata. As already mentioned, quotienting can be seen as a simpler operation of adding transitions and accepting states which simplifies the forthcoming reasoning. Let $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ be an ABA and let $\equiv$ be an equivalence on Q such that $\equiv\, = \,\preceq \cap \preceq^{-1}$ for some preorder $\preceq$. We will use $\mathcal{A}/{\equiv}$ to denote the quotient of $\mathcal{A}$ wrt. $\equiv$ that arises by merging $\equiv$-equivalent states of $\mathcal{A}$, and $\mathcal{A}^+_\preceq$ will stand for the automaton *extended* according to $\preceq$, that is created as follows: for every two states q, r of $\mathcal{A}$ with $q \preceq r$, (i) add all outgoing transitions of q to r, (ii) if $q \equiv r$ and q is final, make r final.

Formally, the automata $\mathcal{A}/{\equiv}$ and $\mathcal{A}^+_\preceq$ are defined as follows. Let $Q/{\equiv}$ denote the partitioning of Q w.r.t. $\equiv$, and let $[q]$ denote the equivalence class of $\equiv$ containing q. Then $\mathcal{A}/{\equiv} = (\Sigma, Q/{\equiv}, [\iota], \delta/{\equiv}, \{[q] \mid q \in \alpha\})$ and $\mathcal{A}^+_\preceq = (\Sigma, Q, \delta^+_\preceq, \iota, \alpha^+_\preceq)$ where $\alpha^+_\preceq = \{p \mid \exists q \in \alpha.\ q \equiv p\}$ and, for each $a \in \Sigma$, $q \in Q$, $\delta/{\equiv}([q], a) = \bigcup_{p \in [q]} \{\{[p'] \mid p' \in P\} \mid P \in \delta(p,a)\}$ and $\delta^+_\preceq(q,a) = \bigcup_{p \in Q \wedge p \preceq q} \delta(p,a)$.

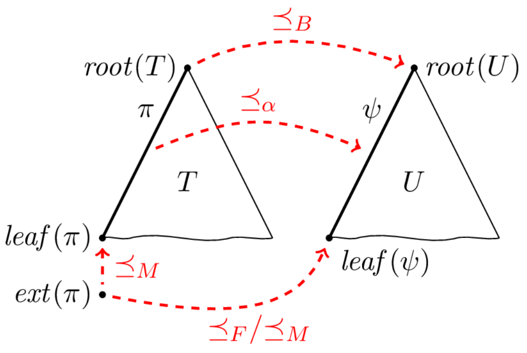


Figure 6.3: U strongly/weakly covers T w.r.t. *ext*

The following lemma implies that if adding transitions and accepting states according to $\preceq$ preserves the language, then quotienting according to $\equiv$ preserves the language too.

Lemma 6.4. $\mathcal{L}(\mathcal{A}/\equiv) \subseteq \mathcal{L}(\mathcal{A}^+_\preceq)$.

Proof. Let $\mathcal{A}^+_\equiv = (\Sigma, Q, \iota, \delta^+_\equiv, \alpha^+_\equiv)$ be the automaton extended according to $\equiv$. Observe that states q and r with $q \equiv r$ are forward simulation equivalent in $\mathcal{A}^+_\equiv$. (q and r are in $\mathcal{A}^+_\equiv$ either both accepting or both nonaccepting, and for all $a \in \Sigma$, $\delta^+_\equiv(q, a) = \delta^+_\equiv(r, a)$). Gurumurthy et al. in [GKSV03] prove that quotienting with respect to forward simulation preserves language. Therefore, $\mathcal{L}(\mathcal{A}/\equiv) = \mathcal{L}(\mathcal{A}^+_\equiv)$. It is also easy to see that $\mathcal{L}(\mathcal{A}^+_\equiv) \subseteq \mathcal{L}(\mathcal{A}^+_\preceq)$, as $\mathcal{A}^+_\preceq$ has a richer transition function than $\mathcal{A}^+_\equiv$ and $\alpha^+_\preceq = \alpha^+_\equiv$. Thus, $\mathcal{L}(\mathcal{A}/\equiv) = \mathcal{L}(\mathcal{A}^+_\equiv) \subseteq \mathcal{L}(\mathcal{A}^+_\preceq)$. □

We now give the proof that extending automata according to the mediated preorder preserves the language. For the rest of the section, we fix an ABA $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$, a reflexive and transitive forward simulation $\preceq_F$ on $\mathcal{A}$ such that $\mathcal{A}$ is $\preceq_F$-unambiguous, and a reflexive and transitive backward simulation $\preceq_B$ on $\mathcal{A}$ parametrised by $\preceq_F$. Let $\preceq_M$ be the mediated preorder induced by $\preceq_F$ and $\preceq_B$, and let $\mathcal{A}^+ = (\Sigma, Q, \iota, \delta^+, \alpha^+)$ be the automaton extended according to $\preceq_M$ (we omit the subscript $\preceq_M$ for the ease of notation). Let $\equiv_M = \preceq_M \cap \preceq_M^{-1}$.
We want to prove that $\mathcal{L}(\mathcal{A}^+) = \mathcal{L}(\mathcal{A})$. The nontrivial part is showing that $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$—the converse is obvious. To prove $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$, we need to show that, for every accepting run of $\mathcal{A}^+$ on a word w, there is an accepting run of $\mathcal{A}$ on w. We first prove Lemma 6.5, which shows how partial runs of $\mathcal{A}$ with an increased power of their leaves (wrt. $\preceq_M$) can be built incrementally from other runs of $\mathcal{A}$, bridging the gap between $\mathcal{A}$ and $\mathcal{A}^+$. Then we prove Lemma 6.8 saying that for every partial run on a word w of $\mathcal{A}^+$, there is a partial run of $\mathcal{A}$ on w that is more accepting (recall that partial runs are finite). By carry this result over to infinite runs we get the proof that extending automata according to $\preceq_M$, and thus also quotienting wrt. $\equiv_M$, preserves language.

Extension Function and Covering. Consider a partial run T of $\mathcal{A}$ on a word w, we choose for each leaf p of T an $\preceq_M$-smaller state p'. Suppose that we allow

p to make one step using the transitions of p' or to become accepting if p' is accepting and $p' \equiv_M p$. (Thus, we give the leaves of T a part of the power they would have in $\mathcal{A}^+$). We will show that there exists a partial run U of $\mathcal{A}$ on w such that (1) it is more accepting than T, and (2) the leaves of U can mimic the next step of the leaves of T even if the leaves of T use their extended power.

The above is formalised in Lemma 6.5 using the following notation. For a partial run T of $\mathcal{A}$ on w, we define *ext* as an *extension function* that assigns to every branch π of T a state $\mathit{ext}(\pi)$ such that $\mathit{ext}(\pi) \preceq_M \mathit{leaf}(\pi)$.

Let U be a partial run of $\mathcal{A}$ on w. For two branches $\pi \in \mathit{branches}(T)$ and $\psi \in \mathit{branches}(U)$, we say that ψ *strongly covers* π wrt. *ext*, denoted $\pi \preceq_{\mathit{ext}} \psi$, iff $\pi \preceq_\alpha \psi$ and $\mathit{ext}(\pi) \preceq_F \mathit{leaf}(\psi)$. Similarly, we say that ψ *weakly covers* π wrt. *ext*, denoted $\pi \preceq_{\mathsf{w}\text{-}\mathit{ext}} \psi$, iff $\pi \preceq_\alpha \psi$ and $\mathit{ext}(\pi) \preceq_M \mathit{leaf}(\psi)$. We extend the concept of covering to partial runs as follows. We write $T \preceq_{\mathit{ext}} U$ (U *strongly covers* T wrt. *ext*) iff $\mathit{branches}(T) \preceq^{\forall\exists}_{\mathit{ext}} \mathit{branches}(U)$ and $\mathit{root}(T) \preceq_B \mathit{root}(U)$. Likewise, we write $T \preceq_{\mathsf{w}\text{-}\mathit{ext}} U$ (U *weakly covers* T wrt. *ext*) iff $\mathit{branches}(T) \preceq^{\forall\exists}_{\mathsf{w}\text{-}\mathit{ext}} \mathit{branches}(U)$ and $\mathit{root}(T) \preceq_B \mathit{root}(U)$. See Figure 6.3 for an illustration. Note that we have $\preceq_{\mathit{ext}} \subseteq \preceq_{\mathsf{w}\text{-}\mathit{ext}}$ for branches as well for partial runs because $\preceq_F \subseteq \preceq_M$—the strong covering implies the weak one.

Lemma 6.5. *For any partial run T of $\mathcal{A}$ on a word w with an extension function ext, there is a partial run U of $\mathcal{A}$ on w with $T \preceq_{\mathit{ext}} U$.*

Proving Lemma 6.5 is the most intricate part of the proof of Theorem 14. We now introduce the concepts used within the proof, prove auxiliary Lemma 6.7, and subsequently present the proof of Lemma 6.5 itself.

Observe that $\mathit{root}(T) \preceq_B \mathit{root}(T)$, and every branch of T weakly covers itself, which means that $T \preceq_{\mathsf{w}\text{-}\mathit{ext}} T$. Within the proof of Lemma 6.5, we will show how to reach U by a chain of partial runs derived from T. The partial runs within the chain will all weakly cover T. Runs further from T will in some sense cover T more strongly than the runs closer to T and the last partial run of the chain will cover T strongly. In the following paragraph, we formulate what it means that a partial run weakly covering T covers T more strongly than another partial run.

The Relation of Covering T More Strongly. To define the relation of covering T more strongly on partial runs that weakly cover T, we concentrate on those branches of partial runs that cause that they do not cover T strongly. Let V be a partial run of $\mathcal{A}$ on w with $T \preceq_{\mathsf{w}\text{-}\mathit{ext}} V$. We call a branch $\psi \in \mathit{branches}(V)$ *strict weakly covering* if there is no $\pi \in \mathit{branches}(T)$ with $\pi \preceq_{\mathit{ext}} \psi$ (there are only some $\pi \in \mathit{branches}(T)$ with $\pi \preceq_{\mathsf{w}\text{-}\mathit{ext}} \psi$). Let $\mathsf{sw}_T(V)$ denote the tree which is the subset of V containing prefixes of strict weakly covering branches of V wrt. T. Note that $T \preceq_{\mathit{ext}} V$ iff V contains no strict weakly covering branches, which is equivalent to $\mathsf{sw}_T(V) = \emptyset$. Given a partial run W of $\mathcal{A}$ on w, we will define which of V and W cover T more strongly by comparing $\mathsf{sw}_T(V)$ and $\mathsf{sw}_T(W)$. For this, we need the following definitions.

Given a finite tree X over Q and $\tau \in Q^+$, we define the *tree decomposition* of X according to τ as the sequence of (finite) sets of paths $\langle \tau, X \rangle = X \ominus_1$

$\tau, X \ominus_2 \tau, \ldots, X \ominus_{|\tau|} \tau$. We also let $\langle \epsilon, X\rangle = \mathit{branches}(X)$ (it is a sequence of length 1). A substantial property of tree decompositions is that under the condition that $\tau \notin \mathit{branches}(X)$, $\langle \tau, X\rangle = \emptyset \ldots \emptyset$ implies that $X = \emptyset$. Notice that if $\tau \in \mathit{branches}(X)$, $\langle \tau, X\rangle = \emptyset \ldots \emptyset$ does not imply $X = \emptyset$ as τ could be the only branch of X. This is important as for a partial run Y and $\tau' \in Y$, if $\tau' \notin \mathit{branches}(Y)$, the implications $\langle \tau', \mathsf{sw}_T(Y)\rangle = \emptyset \ldots \emptyset \implies \mathsf{sw}_T(Y) = \emptyset \implies T \preceq_{\mathit{ext}} Y$ hold. However, the first implication does not hold if $\tau' \in \mathit{branches}(Y)$.

Let $\tau_V \in V \cup \{\epsilon\}$ and $\tau_W \in W \cup \{\epsilon\}$ be such that $\tau_V \notin \mathit{branches}(\mathsf{sw}_T(V))$ and $\tau_W \notin \mathit{branches}(\mathsf{sw}_T(W))$. We say that W *covers* T *more strongly* than V wrt. τ_V and τ_W, denoted $V \prec^T_{\tau_V,\tau_W} W$, iff $\mathit{root}(V) \preceq_B \mathit{root}(W)$ and $\langle \tau_V, \mathsf{sw}_T(V)\rangle \sqsubset \langle \tau_W, \mathsf{sw}_T(W)\rangle$ where $\sqsubset$ is a binary relation on finite sequences of sets of paths defined as follows:

For two sets of paths P and P', we use $P \prec^{\forall\exists}_F P'$ to denote that $P \preceq^{\forall\exists}_F P'$ but not $P' \preceq^{\forall\exists}_F P$. In other words, the upward closure of P' wrt. $\preceq_F$ is a proper subset of the upward closure of P wrt. $\preceq_F$. Then, for two finite sequences $S, S' \in (2^{Q^+})^+$ of sets of paths, $S \sqsubset S'$ iff there is some $k \in \mathbb{N}, k \leq \min\{|S|, |S'|\}$, such that $S_k \prec^{\forall\exists}_F S'_k$ and for all $1 \leq j < k$, $S_j \preceq^{\forall\exists}_F S'_j$.

Given $c \in \mathbb{N}$, we say that a sequence S of sets of paths is *c-bounded* if $|S| \leq c$ and also the length of every path in every $S_i, 1 \leq i \leq |S|$ is at most c. Lemma 6.6 below shows that every maximal increasing chain of c-bounded sequences related by $\sqsubset$ eventually arrives to $\emptyset \ldots \emptyset$. This will allow us to show that every maximal sequence of partial runs that cover T more and more strongly must terminate by a partial run that covers T strongly.

Lemma 6.6. *Given a constant c, every maximal increasing chain of c-bounded sequences related by* $\sqsubset$ *eventually terminates by* $\emptyset \ldots \emptyset$.

Proof. First, observe that for every sequence S of sets of paths with $S \neq \emptyset \ldots \emptyset$, it holds that $S \sqsubset \emptyset \ldots \emptyset$. This is easy to see since $\emptyset \preceq^{\forall\exists}_F \emptyset$ and $\emptyset \prec^{\forall\exists}_F X$ for any nonempty $X \in 2^{Q^+}$. Therefore, to prove the lemma, it is sufficient to show that $\sqsubset$ does not allow infinite increasing chains of c-bounded sequences.

Let $S = S(1) \sqsubset S(2) \sqsubset S(3) \sqsubset \cdots$ be such a chain of c-bounded sequences. We will show that S must be finite. Observe that the domain of possible c-bounded $S(i)$s is finite since there is only finitely many of paths with the length bounded by c (Q is finite). Therefore, if S is an infinite chain, there has to be i and j with $i < j$ such that $S(i) = S(j)$. We will argue that this is not possible by showing that $\sqsubset$ is irreflexive and transitive, which means that it does not allow loops (if there was a loop $X \sqsubset \cdots \sqsubset X$, then by transitivity, $X \sqsubset X$ which contradicts irreflexifity).

Irreflexivity of $\sqsubset$ may be shown as follows. Let $S \sqsubset S$ for some c-bounded sequence S. By the definition of $\sqsubset$, there is $k \in \mathbb{N}$ such that $S_i \preceq^{\forall\exists}_F S_i$ for all $i \in \mathbb{N}$ smaller than k, and $S_k \prec^{\forall\exists}_F S_k$. However, this is clearly not possible since since the upward closure of S_k wrt. $\preceq_F$ would have to be a proper subset of itself.

Transitivity of $\sqsubset$ can be shown as follows. Let S, S', S'' be three c-bounded sequences with $S \sqsubset S' \sqsubset S''$. By the definition of $\sqsubset$, there is $k \in \mathbb{N}$ such that $S_i \preceq^{\forall\exists}_F S'_i$ for all $i \in \mathbb{N}$ smaller than k, and $S_k \prec^{\forall\exists}_F S'_k$; and there is $k' \in \mathbb{N}$ such that $S'_i \preceq^{\forall\exists}_F S''_i$ for all $i \in \mathbb{N}$ smaller than k', and $S'_{k'} \prec^{\forall\exists}_F S''_{k'}$. Let $l = \min\{k, k'\}$.

By transitivity of $\preceq_F^{\forall\exists}$, we have that $S_i \preceq_F^{\forall\exists} S''_i$ for all $i \in \mathbb{N}$ smaller than l. Then, for the lth position, we have that $S_l \prec_F^{\forall\exists} S'_l \prec_F^{\forall\exists} S''_l$ or $S_l \preceq_F^{\forall\exists} S'_l \prec_F^{\forall\exists} S''_l$ or $S_l \prec_F^{\forall\exists} S'_l \preceq_F^{\forall\exists} S''_l$. All these three possibilities give $S_l \prec_F^{\forall\exists} S''_l$, and thus $S \sqsubset S''$. □

The last ingredient we need for the proof of Lemma 6.5 is to show that for every maximal sequence of partial runs that cover T more and more strongly, the underlying $\sqsubset$-related sequence is also maximal. Particularly, we need to show that for any partial run weakly (but not strongly) covering T, we are always able to construct a partial run covering T more strongly. This is stated by the following lemma.

Lemma 6.7. *Given a partial run V of $\mathcal{A}$ on w s.t. $T \preceq_{\mathsf{w}\text{-}ext} V$, $T \not\preceq_{ext} V$, and $\tau_V \in V \cup \{\epsilon\}$ with $\tau_V \notin branches(\mathsf{sw}_T(V))$, we can construct a partial run W of $\mathcal{A}$ on w with $T \preceq_{\mathsf{w}\text{-}ext} W$ and a path $\tau_W \in W$ with $\tau_W \notin branches(\mathsf{sw}_T(W))$ such that $V \prec^T_{\tau_V,\tau_W} W$.*

Proof. The proof relies on Lemma 6.3 and the definition of $\preceq_M$. We first choose a suitable branch π of $\mathsf{sw}_T(V)$ as follows. Let $1 \le k \le |\tau_V|$ be some index such that $\mathsf{sw}_T(V) \ominus_k \tau_V$ is nonempty. If $\tau_V = \epsilon$, then $k = 1$. We choose some $\pi' \in \mathsf{sw}_T(V) \ominus_k \tau_V$ which is minimal wrt. $\preceq_F$, meaning that there is no $\pi'' \in \mathsf{sw}_T(V) \ominus_k \tau_V$ different from π' such that $\pi'' \preceq_F \pi'$. We put $\pi = \tau_V^k \pi'$. We note that this is the place where we use the $\preceq_F$-unambiguity assumption. If $\mathcal{A}$ was $\preceq_F$-ambiguous, there need not be a k such that $\mathsf{sw}_T(V) \ominus_k \tau_V$ contains a minimal element wrt. $\preceq_F$.

As $T \preceq_{\mathsf{w}\text{-}ext} V$, there is $\sigma \in branches(T)$ with $\sigma \preceq_{\mathsf{w}\text{-}ext} \pi$. From $ext(\sigma) \preceq_M leaf(\pi)$, there is a mediator s with $ext(\sigma) \preceq_F s \succeq_B leaf(\pi)$. We can apply Lemma 6.3 to V, π, $leaf(\pi)$ and s, which give us a partial run W and $\psi \in branches(W)$ with $leaf(\psi) = s$ such that $\pi \preceq_B \psi$, and for all $1 \le i \le |\pi|$, $V \ominus_i \pi \preceq_F^{\forall\exists} W \ominus_i \psi$. Let $\tau_W = \psi$. The proof will be concluded by showing that (i) $T \preceq_{\mathsf{w}\text{-}ext} W$, (ii) $\tau_W \notin branches(\mathsf{sw}_T(W))$, and (iii) $\langle \tau_V, \mathsf{sw}_T(V)\rangle \sqsubset \langle \tau_W, \mathsf{sw}_T(W)\rangle$, which implies $V \prec^T_{\tau_V,\tau_W} W$.

(i) To show that $T \preceq_{\mathsf{w}\text{-}ext} W$, we proceed as follows. Observe that for every $\phi \in branches(W) \setminus \{\psi\}$ there is a branch $\phi' \in branches(V) \setminus \{\pi\}$ such that $leaf(\phi') \preceq_F leaf(\phi)$ and $\phi' \preceq_\alpha \phi$. This holds because for all $1 \le i \le |\pi|$, $V \ominus_i \pi \preceq_F^{\forall\exists} W \ominus_i \psi$ and because $\pi \preceq_B \psi$ (To be more detailed, for every $\phi \in branches(W) \setminus \{\psi\}$, $\phi = \psi^i \rho$ for some i and $\rho \in W \ominus_i \psi$. There must be $\rho' \in V \ominus_i \pi$ with $\rho' \preceq_F \rho$. As $\pi \preceq_B \phi$, $\pi^i \preceq_B \phi^i$ which implies $\pi^i \preceq_\alpha \phi^i$. Similarly, $\rho' \preceq_F \rho$ implies $\rho' \preceq_\alpha \rho$ and also $leaf(\rho') \preceq_F leaf(\rho)$. Therefore, we can construct the branch $\phi' = \pi^i \rho' \in branches(V) \setminus \{\pi\}$ with $\pi^i \rho' \preceq_\alpha \psi^i \rho = \phi$ and $leaf(\pi^i \rho') \preceq_F leaf(\psi^i \rho)$). We also know that since $T \preceq_{\mathsf{w}\text{-}ext} V$, $branches(T) \preceq^{\forall\exists}_{\mathsf{w}\text{-}ext} branches(V)$. Thus, by the definition of $\preceq_{\mathsf{w}\text{-}ext}$, we have that for every $\phi \in branches(W) \setminus \{\psi\}$, there are $\phi' \in branches(V)$ and $\phi'' \in branches(T)$ with $\phi'' \preceq_\alpha \phi' \preceq_\alpha \phi$ and $ext(\phi'') \preceq_M leaf(\phi') \preceq_F leaf(\phi)$. This by transitivity of α and the definition of $\preceq_M$ gives $\phi'' \preceq_\alpha \phi$ and $ext(\phi'') \preceq_M leaf(\phi)$, which means $\phi'' \preceq_{\mathsf{w}\text{-}ext} \phi$. To see that also ψ is weakly covering, observe that since $\sigma \preceq_{\mathsf{w}\text{-}ext} \pi$, we have $\sigma \preceq_\alpha \pi \preceq_B \psi$ and $ext(\sigma) \preceq_F s = leaf(\psi)$, which by $\preceq_B \subseteq \preceq_\alpha$ and transitivity of $\preceq_\alpha$ gives even $\sigma \preceq_{ext} \psi$ (immediately implying

$\sigma \preceq_{\mathsf{w}\text{-}ext} \psi$). Finally, from $root(T) \preceq_B root(V)$ (implied by $T \preceq_{\mathsf{w}\text{-}ext} V$), $\pi \preceq_B \psi$, and transitivity of $\preceq_B$, $root(T) \preceq_B root(W)$. We have shown that $T \preceq_{\mathsf{w}\text{-}ext} W$.

(ii) Showing that $\psi \notin branches(\mathsf{sw}_T(W))$ is easy. In the above paragraph we have just shown that $\sigma \preceq_{ext} \psi$, thus ψ is not a strict weakly covering branch.

(iii) To show that $\langle \tau_V, \mathsf{sw}_T(V)\rangle \sqsubset \langle \psi, \mathsf{sw}_T(W)\rangle$, we will argue that (a) for all $1 \leq i < k$, it holds that $\mathsf{sw}_T(V) \ominus_i \tau_V \preceq_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_i \psi$ and that (b) $\mathsf{sw}_T(V) \ominus_k \tau_V \prec_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_k \psi$. Notice first that for any partial run X of $\mathcal{A}$ and $\tau \in X$ with $\tau \notin branches(\mathsf{sw}_T(X))$, for all $1 \leq j \leq |\tau|$, $\mathsf{sw}_T(X) \ominus_j \tau \subseteq X \ominus_j \tau$. Recall that $\tau_V^k = \pi^k$, that $\mathsf{sw}_T(V) \ominus_k \tau_V$ is nonempty, and that for all $1 \leq i < |\pi|$, $V \ominus_i \pi \preceq_F^{\forall\exists} W \ominus_i \psi$.

We first show that for all $1 \leq i < |\pi|$, $\mathsf{sw}_T(V) \ominus_i \pi \preceq_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_i \psi$. For every $\phi \in \mathsf{sw}_T(W) \ominus_i \psi$, there is at least one $\phi' \in V \ominus_i \pi$ with $\phi' \preceq_F \phi$ (because $V \ominus_i \pi \preceq_F^{\forall\exists} W \ominus_i \psi$ and $\mathsf{sw}_T(W) \ominus_i \psi \subseteq W \ominus_i \psi$). We will show by contradiction that $\phi' \in \mathsf{sw}_T(V) \ominus_i \pi$ which will imply $\mathsf{sw}_T(V) \ominus_i \pi \preceq_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_i \psi$. Suppose that $\phi' \notin \mathsf{sw}_T(V) \ominus_i \pi$. Then the branch $\pi^i\phi'$ of V is not strict weakly covering, and as $T \preceq_{\mathsf{w}\text{-}ext} V$, we have that there is some $\phi'' \in branches(T)$ with $\phi'' \preceq_{ext} \pi^i\phi'$. As $\pi \preceq_B \psi$, we have that $\pi^i \preceq_\alpha \psi^i$. As $\phi' \preceq_F \phi$, we have that $\phi' \preceq_\alpha \phi$ and $leaf(\phi') \preceq_F leaf(\phi)$. This together with $\phi'' \preceq_{ext} \pi^i\phi'$ gives that $\phi'' \preceq_\alpha \pi^i\phi' \preceq_\alpha \psi^i\phi$ and $ext(\phi'') \preceq_F leaf(\pi^i\phi') \preceq_F leaf(\psi^i\phi)$. By transitivity of $\preceq_\alpha$ and $\preceq_F$ and by the definition of $\preceq_{ext}$, we obtain $\phi'' \preceq_{ext} \psi^i\phi$. This contradicts with the fact that $\psi^i\phi$ is strict weakly covering (as $\phi \in \mathsf{sw}_T(W) \ominus_i \psi$) and therefore it must be the case that $\phi' \in \mathsf{sw}_T(V) \ominus_i \pi$.

(a) The fact that for all $1 \leq i < k$, $\mathsf{sw}_T(V) \ominus_i \tau_V \preceq_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_i \psi$ is implied by the result of the previous paragraph, because $\tau_V^k = \pi^k$ (thus $\mathsf{sw}_T(V) \ominus_i \tau_V = \mathsf{sw}_T(V) \ominus_i \pi$).

(b) It remains to show that $\mathsf{sw}_T(V) \ominus_k \tau_V \prec_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_k \psi$. By the definitions of $\ominus_k$, π and τ_V, it holds that $\mathsf{sw}_T(V) \ominus_k \tau_V \supset \mathsf{sw}_T(V) \ominus_k \pi$. (To see this, recall that π is strict weakly covering, but τ_V is not. Therefore, $\mathsf{sw}_T(V) \ominus_k \pi = \mathsf{sw}_T(V) \ominus_k \tau_V \setminus branches(\mathsf{sw}_T(V)(\pi^{k+1})))$. Since $\supset$ implies $\preceq_F^{\forall\exists}$, we have that $\mathsf{sw}_T(V) \ominus_k \tau_V \preceq_F^{\forall\exists} \mathsf{sw}_T(V) \ominus_k \pi$. Moreover, since $\pi' \notin \mathsf{sw}_T(V) \ominus_k \pi$ and π' is a minimal element of $\mathsf{sw}_T(V) \ominus_k \tau_V$, $\mathsf{sw}_T(V) \ominus_k \pi \preceq_F^{\forall\exists} \mathsf{sw}_T(V) \ominus_k \tau_V$ cannot hold (there is no $\pi'' \in \mathsf{sw}_T(V) \ominus_k \pi$ with $\pi'' \preceq_F \pi'$), and therefore we have $\mathsf{sw}_T(V) \ominus_k \tau_V \prec_F^{\forall\exists} \mathsf{sw}_T(V) \ominus_k \pi$. Finally, $\mathsf{sw}_T(V) \ominus_k \tau_V \prec_F^{\forall\exists} \mathsf{sw}_T(V) \ominus_k \pi \preceq_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_k \psi$ gives $\mathsf{sw}_T(V) \ominus_k \tau_V \prec_F^{\forall\exists} \mathsf{sw}_T(W) \ominus_k \psi$. This completes the part (iii) of the proof and we can conclude that $V \prec_{\tau_V,\psi}^T W$. □

With Lemma 6.7 in hand, we are finally ready to prove Lemma 6.5.

Proof of Lemma 6.5. If $T \preceq_{ext} T$, we are done as in the statement of the lemma, we can take T to be U. So, suppose that $T \npreceq_{ext} T$. Observe that $root(T) \preceq_B root(T)$, and every branch of T weakly covers itself, which means that $T \preceq_{\mathsf{w}\text{-}ext} T$. We construct a run U strongly covering T as follows. Starting from T and ϵ, we can construct a chain $T \prec_{\epsilon,\tau_1}^T T_1 \prec_{\tau_1,\tau_2}^T T_2 \prec_{\tau_2,\tau_3}^T T_3 \ldots$ of partial runs that more and more strongly cover T by successively applying Lemma 6.7 for each i, $\tau_i \in T_i$, $\tau_i \notin branches(\mathsf{sw}_T(T_i))$, and $T \preceq_{\mathsf{w}\text{-}ext} T_i$. Observe that by the definition of stronger covering, we have that $\langle \epsilon, \mathsf{sw}_T(T)\rangle \sqsubset \langle \tau_1, \mathsf{sw}_T(T_1)\rangle \sqsubset \langle \tau_2, \mathsf{sw}_T(T_2)\rangle \sqsubset \langle \tau_3, \mathsf{sw}_T(T_3)\rangle \ldots$.

Notice now that for each i, since $T \preceq_{\mathsf{w}\text{-}ext} T_i$, $height(T_i) \leq height(T)$. Therefore, since length of τ_i is bounded by $height(T)$, the length of $\langle \tau_i, \mathsf{sw}_T(T_i)\rangle$ is bounded by $height(T)$ too. Since lengths of all paths in the sets within $\langle \tau_i, \mathsf{sw}_T(T_i)\rangle$ are obviously bounded by $height(T)$ as well, $\langle \tau_i, \mathsf{sw}_T(T_i)\rangle$ is a $height(T)$-bounded sequence. Therefore, by Lemma 6.6, the chain must eventually arrive to its last T_k and τ_k with $\langle \tau_k, \mathsf{sw}_T(T_k)\rangle = \emptyset \ldots \emptyset$. As $\langle \tau_k, \mathsf{sw}_T(T_k)\rangle = \emptyset \ldots \emptyset$, $\mathsf{sw}_T(T_k)$ has to be empty, which implies that $T \preceq_{ext} T_k$. We can put $U = T_k$ and Lemma 6.5 is proven. □

We use Lemma 6.5 to prove Lemma 6.8. Informally, it says that even despite the poorer transition relation and smaller set of accepting states, $\mathcal{A}$ can answer to any partial run of $\mathcal{A}^+$ by a more accepting partial run. To express this formally, we need to define the following weaker version $\preceq_{\alpha^+\Rightarrow\alpha}$ of the relation of being more accepting that takes into account α^+ on the left and α on the right. This is, for states q and r, $q \preceq_{\alpha^+\Rightarrow\alpha} r$ iff $q \in \alpha^+ \implies r \in \alpha$. For two paths $\pi, \psi \in Q^+$, $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi$ iff $|\pi| = |\psi|$ and for all $1 \leq i \leq |\pi|$, $\pi_i \in \alpha^+ \implies \psi_i \in \alpha$. Last, for finite trees T and U over Q, we use $T \preceq_{\alpha^+\Rightarrow\alpha} U$ to denote that $branches(T) \preceq^{\forall\exists}_{\alpha^+\Rightarrow\alpha} branches(U)$.

Lemma 6.8. *For any partial run T of $\mathcal{A}^+$ on $w \in \Sigma^\omega$, there exists a partial run U of $\mathcal{A}$ on w such that $root(T) \preceq_B root(U)$ and $T \preceq_{\alpha^+\Rightarrow\alpha} U$.*

Proof. By induction to the structure of T, using Lemma 6.5 within the induction step. To make the induction argument pass, we will prove a stronger variant of the lemma. Particularly, we will replace the relation $\preceq_{\alpha^+\Rightarrow\alpha}$ within the statement of the lemma by its stronger variant $\preceq^M_{\alpha^+\Rightarrow\alpha}$ which is defined as follows. Given paths π and ψ, $\pi \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$ iff $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi$ and $leaf(\pi) \preceq_M leaf(\psi)$. For two partial runs V and W, we use $V \preceq^M_{\alpha^+\Rightarrow\alpha} W$ to denote that $branches(V)\ (\preceq^M_{\alpha^+\Rightarrow\alpha})^{\forall\exists}\ branches(W)$. Apparently, $\preceq^M_{\alpha^+\Rightarrow\alpha} \subseteq \preceq_{\alpha^+\Rightarrow\alpha}$ for paths as well as for partial runs.

A stronger variant of the lemma: For any partial run T of $\mathcal{A}^+$ on $w \in \Sigma^\omega$, there exists a partial run U of $\mathcal{A}$ on w such that $root(T) \preceq_B root(U)$ and $T \preceq^M_{\alpha^+\Rightarrow\alpha} U$.

It is obvious that the above statement implies the statement of the lemma. We will prove it by induction to the structure of T. In the base case, $T = \{q\}$ for some $q \in Q$. If $q \notin \alpha^+$, we can put $U = \{q\}$ ($\preceq_M$ and $\preceq_B$ are reflexive). If $q \in \alpha^+$, then by the definition of α^+, there is $p \in \alpha$ such that $p \equiv_M q$. This means that $q \preceq_M p$ and $p \preceq_M q$. By the definition of $\preceq_M$, there exists a mediator s with $p \preceq_F s \succeq_B q$. As $\preceq_F \subseteq \preceq_\alpha$, $s \in \alpha$. Again by the definition of $\preceq_M$, $q \preceq_M p \preceq_F s \succeq_B q$ gives us $q \preceq_M s \succeq_B q$ and we can put $U = \{s\}$.

Suppose now that T is not only a root and that the stronger variant of the lemma holds for every partial run of $\mathcal{A}^+$ on w that is a proper subset of T. We choose some $\pi \in T$ such that $succ_T(\pi) \neq \emptyset$ and for every $p \in succ_T(\pi)$, $succ_T(\pi p) = \emptyset$. Notice that since T is a finite tree, such π always exists. Denote $P = succ_T(\pi)$ and $q = leaf(\pi)$. Let $T' = T \setminus \{\pi p \mid p \in P\}$. T' is a partial run of $\mathcal{A}^+$ on w which is a proper subset of T, therefore we can apply the induction hypothesis on it. This gives us a partial run V of $\mathcal{A}$ on w such that $root(T') \preceq_B root(V)$ and $T' \preceq^M_{\alpha^+\Rightarrow\alpha} V$.

Let $Bad_V \subseteq branches(V)$ be the set such that $\psi \in Bad_V$ iff there is no $\phi \in branches(T)$ such that $\phi \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$, and let $Good_V = branches(V) \setminus Bad_V$. Intuitively, Bad_V contains the problematic branches because of which $T \preceq^M_{\alpha^+\Rightarrow\alpha} V$ does not hold. If Bad_V it is empty, then the relation holds and we can conclude the proof. We continue assuming that $Bad_V \neq \emptyset$.

By the definition of δ^+ and because $q \xrightarrow{w_{|\pi|}} P$ is a transition of $\mathcal{A}^+$, there must be some $s \in Q, s \preceq_M q$ where $s \xrightarrow{w_{|\pi|}} P$ is a transition of δ. We define an extension function ext_V such that $ext_V(\phi) = s$ for every $\phi \in Bad_V$ and $ext_V(\psi) = leaf(\psi)$ for every $\psi \in Good_V$. To see that ext_V conforms the definition of extension function, one has to show that for every branch $\phi \in Bad_V$, $s \preceq_M leaf(\phi)$. We know that $T' \preceq^M_{\alpha^+\Rightarrow\alpha} V$ but not $T \preceq^M_{\alpha^+\Rightarrow\alpha} V$. Therefore, there is some branch $\phi' \in T'$ with $\phi' \preceq^M_{\alpha^+\Rightarrow\alpha} \phi$ such that $\phi' \notin branches(T)$ (if ϕ' was a branch of T, ϕ would not be in Bad_V). Notice that π is the only branch of T' which is not a branch of T, which means that it must be the case that $\phi' = \pi$. Therefore, since $s \preceq_M q \preceq_M leaf(\phi)$, $s \preceq_M leaf(\phi)$ holds.

By applying Lemma 6.5 to V and ext_V, we get a partial run W of $\mathcal{A}$ on w with $V \preceq_{ext_V} W$. Now, for each $\psi \in branches(W)$, there is $\phi \in branches(V)$ with $\phi \preceq_{ext_V} \psi$. As $T' \preceq^M_{\alpha^+\Rightarrow\alpha} V$, $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \phi$ for some $\rho \in branches(T')$. There are two cases of how ρ and ψ may be related, depending on ϕ:

1. If $\phi \in Good_V$, then $ext(\phi) = leaf(\phi)$. In this case, by the definitions of $\preceq^M_{\alpha^+\Rightarrow\alpha}$ and $\preceq_{ext_V}$, we have $\rho \preceq_{\alpha^+\Rightarrow\alpha} \phi \preceq_\alpha \psi$ and $leaf(\rho) \preceq_M leaf(\phi) \preceq_F leaf(\psi)$, which gives $\rho \preceq_{\alpha^+\Rightarrow\alpha} \psi$ and $leaf(\rho) \preceq_M leaf(\psi)$ (since $\preceq_M$ is forward extensible), meaning that $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$.

2. To analyse the case when $\phi \in Bad_V$, recall that π is the only branch of T' which is not a branch of T, and therefore π is also the only branch of T' with $\pi \preceq^M_{\alpha^+\Rightarrow\alpha} \phi$. Therefore, $\rho = \pi$. According to the definition of ext_V, $ext_V(\phi) = s$. Since $\phi \preceq_{ext_V} \psi$, we have $\pi \preceq^M_{\alpha^+\Rightarrow\alpha} \phi \preceq_\alpha \psi$ which gives $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi$. However, since (contrary to the previous case 1.) $ext_v(\phi) \neq leaf(\phi)$, we cannot guarantee any further relation between $leaf(\phi)$ and $leaf(\psi)$, and we cannot derive that $leaf(\pi) \preceq_M leaf(\psi)$ and $\pi \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$ need not hold.

We define the set $Bad_W \subseteq branches(W)$ such as $\psi \in Bad_W$ iff there is no $\rho \in T$ with $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$ and we let $Good_W = branches(W) \setminus Bad_V$. This is, Bad_W contains the branches because of which $T \preceq^M_{\alpha^+\Rightarrow\alpha} W$ does not hold. Note that if $\psi \in Bad_V$, then all the $\phi \in branches(V)$ with $\phi \preceq_{ext_V} \psi$ are as in the case (2) above, i.e., π is the only branch of T' with $\pi \preceq^M_{\alpha^+\Rightarrow\alpha} \phi$. By the definition of $\preceq_{ext_V}$, $s = ext_V(\phi) \preceq_F leaf(\psi)$. Therefore, by the definition of $\preceq_F$ and since $s \xrightarrow{w_{|\pi|}} P$, there must be some transition $leaf(\psi) \xrightarrow{w_{|\pi|}} R_\psi$ of $\mathcal{A}$ where $P \preceq^{\forall\exists}_F R_\psi$. We extend W by firing these transitions for every $\psi \in Bad_W$, in which way we obtain a run $X = W \cup \{\psi R_\psi \mid \psi \in Bad_W\}$ of $\mathcal{A}$ on w.

Let us use $New_X = \{\psi R_\psi \mid \psi \in Bad_W\}$ to denote the branches of X that arose by firing the transitions. Observe that $branches(X) = Good_W \cup New_X$.

Recall that for all $\psi \in \mathit{Bad}_W$, $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi$ and that for every $\psi \in \mathit{New}_X$, there is some $p \in P$ such that $p \preceq_F \mathit{leaf}(\psi)$. We will define an extension function ext_X of X as follows:

1. If $\psi \in \mathit{Good}_W$, $\mathit{ext}_X(\psi) = \mathit{leaf}(\psi)$.

2. If $\psi \in \mathit{New}_X$ and there is $p \in P$ with $p \preceq_F \mathit{leaf}(\psi)$ and $p \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\psi)$, we let $\mathit{ext}_X(\psi) = \mathit{leaf}(\psi)$.

3. If $\psi \in \mathit{New}_X$ and there is no $p \in P$ with $p \preceq_F \mathit{leaf}(\psi)$ and $p \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\psi)$, we proceed as follows. By the definition of New_X, there is some $p' \in P$ such that $p' \preceq_F \mathit{leaf}(\psi)$. Since $\preceq_F \subseteq \preceq_\alpha$, $p' \preceq_F \mathit{leaf}(\psi)$, and not $p' \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\psi)$, it must be the case that $p' \notin \alpha$, $\mathit{leaf}(\psi) \notin \alpha$, and $p' \in \alpha^+$. This by the definition of α^+ means that there is some $v \in \alpha$ with $p' \equiv_M v$. We put $\mathit{ext}_X(\psi) = v$.

We apply Lemma 6.5 to X and ext_X, which gives us a partial run U of $\mathcal{A}$ on w with $X \preceq_{\mathit{ext}_X} U$. We will check that U satisfies the statement of the stronger variant of the lemma. We will first prove that $T \preceq^M_{\alpha^+\Rightarrow\alpha} U$. For each $\tau \in \mathit{branches}(U)$, there is $\psi \in \mathit{branches}(X)$ with $\psi \preceq_{\mathit{ext}_X} \tau$. We will derive that there is some $\rho \in \mathit{branches}(T)$ with $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \tau$. The argument depends on properties of ψ. Particularly, we have the following three cases.

1. If $\psi \in \mathit{Good}_W$, then there is some $\rho \in T$ with $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \psi$. Recall that $\mathit{ext}_X(\psi) = \mathit{leaf}(\psi)$ in this case. Thus, by the definitions of $\preceq^M_{\alpha^+\Rightarrow\alpha}$ and $\preceq_{\mathit{ext}_X}$, we have $\rho \preceq_{\alpha^+\Rightarrow\alpha} \psi \preceq_\alpha \tau$ and $\mathit{leaf}(\rho) \preceq_M \mathit{leaf}(\psi) \preceq_F \mathit{leaf}(\tau)$, which gives $\rho \preceq_{\alpha^+\Rightarrow\alpha} \tau$ and $\mathit{leaf}(\rho) \preceq_M \mathit{leaf}(\tau)$, i.e., $\rho \preceq^M_{\alpha^+\Rightarrow\alpha} \tau$.

2. If $\psi \in \mathit{New}_X$ and there is some $p \in P$ with $p \preceq_F \mathit{leaf}(\psi)$ and $p \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\psi)$, then by the definition of ext_X, $\mathit{ext}_X(\psi) = \mathit{leaf}(\psi)$. Recall that as $\psi^{|\psi|-1} \in \mathit{Bad}_W$, $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi^{|\psi|-1}$. Therefore, also $\pi p \preceq_{\alpha^+\Rightarrow\alpha} \psi$. By the definition of $\preceq_{\mathit{ext}_X}$, we have that $\psi \preceq_\alpha \tau$ and $\mathit{leaf}(\psi) \preceq_F \mathit{leaf}(\tau)$. Finally, $\pi p \preceq_{\alpha^+\Rightarrow\alpha} \psi \preceq_\alpha \tau$ and $p \preceq_F \mathit{leaf}(\psi) \preceq_F \mathit{leaf}(\tau)$ together imply that $\pi p \preceq^M_{\alpha^+\Rightarrow\alpha} \tau$.

3. If $\psi \in \mathit{New}_X$ and there is no $p \in P$ with $p \preceq_F \mathit{leaf}(\psi)$ and $p \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\psi)$, then by the definition of ext_X, there are $p' \in P$ with $p' \preceq_F \mathit{leaf}(\psi)$ and $v \in \alpha$ with $v \equiv_M p'$ such that $\mathit{ext}_X(\psi) = v$. By $\psi \preceq_{\mathit{ext}_X} \tau$, we have $\psi \preceq_\alpha \tau$ and $v \preceq_F \mathit{leaf}(\tau)$. Thus, since $\preceq_M$ is forward extensible, $p' \equiv_M v \preceq_F \mathit{leaf}(\tau)$ gives $p' \preceq_M \mathit{leaf}(\psi)$. As $\preceq_F \subseteq \preceq_\alpha$, we have that $\mathit{leaf}(\tau) \in \alpha$ and thus $p' \preceq_{\alpha^+\Rightarrow\alpha} \mathit{leaf}(\tau)$. As $\psi^{|\psi|-1} \in \mathit{Bad}_W$, we have that $\pi \preceq_{\alpha^+\Rightarrow\alpha} \psi^{|\psi|-1}$. Together with $\psi \preceq_\alpha \tau$, this gives $\pi p' \preceq_{\alpha^+\Rightarrow\alpha} \tau$. Therefore, $\pi p' \preceq^M_{\alpha^+\Rightarrow\alpha} \tau$.

Since the above three cases cover all possible variants of ψ and thus all branches of U, we have proven that $T \preceq^M_{\alpha^+\Rightarrow\alpha} U$. Finally, it is easy to show that $\mathit{root}(T) \preceq_B \mathit{root}(U)$ since $\preceq_B$ is transitive and we know that $\mathit{root}(T) = \mathit{root}(T') \preceq_B \mathit{root}(V) \preceq_B \mathit{root}(W) = \mathit{root}(X) \preceq_B \mathit{root}(U)$. We have verified that the constructed partial run U satisfies the statement of the stronger variant of the lemma, which concludes the proof. $\square$

With Lemma 6.8 in hand, we can prove that for each accepting run of $\mathcal{A}^+$ on a word w, there is an accepting run of $\mathcal{A}$ on w. This requires to carry Lemma 6.8 from finite partial runs to full infinite runs.

Lemma 6.9. *A run T of $\mathcal{A}$ with $root(T) = \iota$ is accepting if and only if for every $\pi \in T$, there exists a constant $k_\pi \in \mathbb{N}$ such that every ψ with $\pi\psi \in T$ and $|\psi| \geq k$ contains an accepting state.*

Proof. (if) For every $\pi \in branches(T)$, there is an infinite sequence of $k_0, k_1 \ldots$ such that:

- $k_0 = 0$ and
- for all $i \in \mathbb{N}$, $k_i = k_{i-1} + k_{\pi^n}$ where $n = k_{i-1} + 1$.

For all $i \in \mathbb{N}$, every segment of π between $k_{i-1}+1$ and k_i contains an accepting state, therefore π contains infinitely many accepting states.

(only if) By contradiction. Suppose that there is $\pi \in T$ for which there is no k_π. We will show that in this case, there must be $\psi \in Q^\omega$ such that $\pi\psi \in branches(T)$ and ψ does not contain an accepting state (which contradicts the assumption that T is accepting).

We will give a procedure which returns ψ^i for each $i \in \mathbb{N}$ (based on the knowledge of ψ^{i-1}). For each $i \in \mathbb{N}^0$, we will keep the invariant that for $\pi\psi^i$, $k_{\pi\psi^i}$ does not exists and that ψ^i does not contain an accepting state. Since $\psi^0 = \epsilon$, the invariant holds for $i = 0$.

Let the invariant hold for $i-1, i \in \mathbb{N}$, and suppose that we have already constructed ψ^{i-1}. Denote P the subset of $succ_T(\pi\psi^{i-1})$ containing nonaccepting states. P must be nonempty, because if all the states from $succ_T(\pi\psi^{i-1})$ were accepting, $k_{\pi\psi^{i-1}}$ would equal 1, violating the invariant for $i-1$. Then, there must be a state $q \in P$ such that $k_{\pi\psi^{i-1}q}$ does not exist, since otherwise we could put $k_{\pi\psi^{i-1}} = \max\{k_{\pi\psi^{i-1}p} \mid p \in P\} + 1$, which would also violate the invariant for $i-1$. We choose q as the continuation and put $\psi^i = \psi^{i-1}q$. Observe that this choice satisfied the invariant for i.

We have shown that for every $i \in \mathbb{N}$, we can construct the ith prefix ψ^i of ψ that does not contain an accepting state. Therefore, the whole infinite path ψ does not contain an accepting state, and the branch $\pi\psi$ of T does not contain infinitely many accepting states. This contradicts the assumption that T is accepting. □

Lemma 6.10. *For every accepting run T of $\mathcal{A}^+$ a word $w \in \Sigma^\omega$, there exists an accepting run U of $\mathcal{A}$ on w.*

Proof. For a tree X over Q, let $X(i) = \{\pi \in X \mid |\pi| \leq i\}$ be the ith prefix of X ($X(0) = \emptyset$). From Lemma 6.8, for each $i \in \mathbb{N}$, there is a partial run U_i of $\mathcal{A}$ on w such that $T(i) \preceq_{\alpha^+ \Rightarrow \alpha} U_i$ and $root(T(i)) \preceq_B root(U_i)$. As $\preceq_B \subseteq \preceq_\iota$, $root(U_i) = \iota$. Note that for all $\pi \in branches(U_i)$, $|\pi|$ equals i, because only paths of the same length can be related by $\preceq_{\alpha^+ \Rightarrow \alpha}$. Denote $\mathbb{U}^\infty = \{U_1, U_2, \ldots\}$. $\mathbb{U}^\infty$ is an infinite set that for each $k \in \mathbb{N}$ contains a partial run U_k of $\mathcal{A}$ with all the branches of the length k. We will use $\mathbb{U}^\infty$ to construct the infinite accepting run U.

Observe that for any infinite set $\mathbb{V}^\infty$ of partial runs of $\mathcal{A}$ and for any $i \in \mathbb{N}$, there has to be at least one partial run W of $\mathcal{A}$ such that for infinitely many $V \in \mathbb{V}^\infty$, $W = V(i)$. The reason is that for any $i \in \mathbb{N}$, there is obviously only finitely many of possible partial runs of the height i that $\mathcal{A}$ can generate.

We prove the existence of U by giving a procedure, which for every $k \in \mathbb{N}$ gives the kth prefix $U(k)$ of U.

- Let $\mathbb{U}_0^\infty = \mathbb{U}^\infty$ and let $U(0) = \emptyset$.
- For every $k \in \mathbb{N}$, $U(k)$ is derived from $U(k-1)$ as follows. Let $\mathbb{U}_k^\infty \subseteq \mathbb{U}^\infty$ be defined as the set such that for all $i \in \mathbb{N}$, $U_i \in \mathbb{U}_k^\infty$ iff $U(k-1) = U_i(k-1)$. In other words, $\mathbb{U}_k^\infty$ is the subset of $\mathbb{U}^\infty$ of the partial runs with the ith prefix equal to $U(k-1)$. Then, $U(k) = U_n(k)$ for some $n \geq k$ such that $U_n \in \mathbb{U}_k^\infty$ and there is infinitely many $m \in \mathbb{N}$ such that $U_m \in \mathbb{U}_k^\infty$ and $U_n(k) = U_m(k)$. I other words, $U(k)$ is a tree that appears as the kth prefix of infinitely many partial runs in $\mathbb{U}_k^\infty$.

To see that this construction is well defined, observe that:

- $\mathbb{U}_0^\infty$ is infinite, and
- for all $k \in \mathbb{N}$, if $\mathbb{U}_{k-1}^\infty$ is infinite, then $U(k-1)$ is defined and $\mathbb{U}_k^\infty$ is infinite.

Thus, $U(k)$ is well defined for every $k \in \mathbb{N}$ and U is a run of $\mathcal{A}$.

It remains prove that U is accepting. We will show that for every $\pi \in U$, there is $k_\pi \in \mathbb{N}$ such that every ψ with $\pi\psi \in T$ and $|\psi| \geq k$ contains an accepting state. By Lemma 6.9, it will follow that U is accepting.

Let us choose arbitrary $\pi \in U$. Let $n = |\pi|$. By Lemma 6.9, for every $\pi' \in \mathit{branches}(T_n)$, there is $k_{\pi'} \in \mathbb{N}$ such that every ψ' with $\pi'\psi' \in T$ and $|\psi'| \geq k_{\pi'}$ contains an accepting state. Let $k = \max\{k_{\pi'} \mid \pi' \in \mathit{branches}(T(n))\}$. By the construction of U, $T(n+k) \preceq_{\alpha^+ \Rightarrow \alpha} U(n+k)$. This implies that for every $\pi'' \in \mathit{branches}(U(n))$, every ψ'' with $\pi''\psi'' \in T$ and $|\psi''| \geq k$ contains an accepting state. As π in $\mathit{branches}(U(n))$, we can put $k_\pi = k$ and we are done. □

Theorem 14. $\mathcal{L}(\mathcal{A}^+) = \mathcal{L}(\mathcal{A})$.

Proof. The inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$ is obvious as $\mathcal{L}(\mathcal{A}^+)$ has riches both transition function and the set of accepting states. The inclusion $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$ follows immediately from Lemma 6.10. □

Corollary 2. *Quotienting with mediated equivalence preserves the language.*

6.4 Computing the Relations

In this section, we describe algorithms for computing ABA forward and backward simulation, and mediated preorder. For forward simulation, we use an algorithm from [FW02], for backward simulation, we present an algorithm based on a translation to an LTS simulation problem similar to the one from Chapter 4 for computing upward TA simulation. Mediated preorder is then computed by the algorithm presented in Chapter 4. For the mediated preorder to be useful for quotienting, we also need to remove ambiguity before we start computing the backward simulation. This can be done by a simple procedure presented in this section too. For the rest of the section, we fix an ABA $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$.

Forward Simulation. The algorithm for computing maximal forward simulation $\preceq_F$ on $\mathcal{A}$ can be found in Fritz and Wilke's work [FW02] (it is called direct simulation in their paper). They reduce the problem of computing maximal forward simulation to a simulation game. Although Fritz and Wilke use a slightly different definition of ABA, it is easy to translate $\mathcal{A}$ to an ABA under their definition with $\mathcal{O}(n+m)$ states and $\mathcal{O}(nm)$ transitions and then use their algorithm to compute $\preceq_F$. The time complexity of the above procedure is $\mathcal{O}(nm^2)$.

Removing Ambiguity. As we have argued in Section 6.3.1, $\mathcal{A}$ needs to be $\preceq_F$-unambiguous for mediated minimisation. Here, we describe how to modify $\mathcal{A}$ to make it $\preceq_F$-unambiguous. The modification does not change the language of $\mathcal{A}$ and also the forward simulation relation $\preceq_F$, therefore we do not need to recompute forward simulation again for the modified automaton.

The procedure for removing ambiguity is simple. For every transition $p \xrightarrow{a} P$ with $P = \{p_1, \dots, p_k\}$ and for each $i \in \{1, \dots, k\}$, we check if there exists some $i < j \leq k$ such that $p_j \preceq_F p_i$. If there is one, remove p_i from P. The time complexity of this procedure is obviously in $\mathcal{O}(n^2 m)$.

We note that an alternative way is quotienting the automaton w.r.t. forward simulation equivalence.

6.4.1 Computing Backward Simulation

Our algorithm for computing backward simulation is inspired by the algorithms for computing tree automata simulations—we translate the problem of computing maximal backward simulation on $\mathcal{A}$ to a problem of computing maximal simulation on a labelled transition system.

The reduction is very similar to the reduction of the problem of computing tree automata backward simulation from Chapter 4. We first define the notion of an *environment*, which is a tuple of the form $(p, a, P \setminus \{p'\})$ obtained by removing a state $p' \in P$ from the transition $p \xrightarrow{a} P$ of $\mathcal{A}$. Intuitively, an environment records the neighbours of the removed state p' in the transition $p \xrightarrow{a} P$. We denote the set of all environments of $\mathcal{A}$ by $Env(\mathcal{A})$. Formally, we define the LTS $\mathcal{A}^{\odot} = (\Sigma, Q^{\odot}, \Delta^{\odot})$ as follows:

- $Q^{\odot} = \{q^{\odot} \mid q \in Q\} \cup \{(p, a, P)^{\odot} \mid (p, a, P) \in Env(\mathcal{A})\}$.
- $\Delta^{\odot} = \{(p, a, P \setminus \{p'\})^{\odot} \xrightarrow{a} p^{\odot}, p'^{\odot} \xrightarrow{a} (p, a, P \setminus \{p'\})^{\odot} \mid P \in \delta(p, a), p' \in P\}$.

An example of the reduction is given in Figure 6.4. The goal of this reduction is to obtain a simulation relation on $\mathcal{A}^{\odot}$ with the following property: $p^{\odot}$ is simulated by $q^{\odot}$ in $\mathcal{A}^{\odot}$ iff $p \preceq_B q$ in $\mathcal{A}$. However, the maximal simulation on $\mathcal{A}^{\odot}$ is not sufficient to achieve this goal. Some essential conditions for backward

A transition in $\mathcal{A}$ Transitions in $\mathcal{A}^\odot$

$$\begin{tikzcd} & p_1^\odot \arrow[r, "a"] & (p,a,\{p_2,p_3\})^\odot \arrow[dr, "a"] & \\ p \xrightarrow{a} \{p_1,p_2,p_3\} \quad \Rightarrow & p_2^\odot \arrow[r, "a"] & (p,a,\{p_1,p_3\})^\odot \arrow[r, "a"] & p^\odot \\ & p_3^\odot \arrow[r, "a"] & (p,a,\{p_1,p_2\})^\odot \arrow[ur, "a"'] & \end{tikzcd}$$

Figure 6.4: An example of the reduction from an ABA transition to LTS transitions

simulation (e.g., $p \preceq_B q \implies p \preceq_\alpha q$) are missing in $A^\odot$. This can be fixed by defining a proper initial preorder I.

Formally, we let $I = \{(q_1^\odot, q_2^\odot) \mid q_1 \preceq_\iota q_2 \wedge q_1 \preceq_\alpha q_2\} \cup \{((p,a,P)^\odot, (r,a,R)^\odot) \mid P \preceq_F^{\forall\exists} R\}$. Observe that I is a preorder. Recall that according to the definition of the backward simulation, $p \preceq_B r$ implies that (1) $p \preceq_\iota r$, (2) $p \preceq_\alpha r$, and (3) for all transitions $q \xrightarrow{a} P \cup \{p\}, p \notin P$, there exists a transition $s \xrightarrow{a} R \cup \{r\}, r \notin R$ such that $q \preceq_B s$ and $P \preceq_F^{\forall\exists} R$. The set $\{(q_1^\odot, q_2^\odot) \mid q_1 \preceq_\iota q_2 \wedge q_1 \preceq_\alpha q_2\}$ encodes the conditions (1) and (2) required by the backward simulation, while the set $\{((p,a,P)^\odot, (r,a,R)^\odot) \mid P \preceq_F^{\forall\exists} R\}$ encodes the condition (3). A simulation relation $\preceq^I$ can be computed using the aforementioned procedure with LTS $\mathcal{A}^\odot$ and the *initial* preorder I. The following theorem shows the correctness of our approach to computing backward simulation.

Theorem 15. *For all $q, r \in Q$, we have $q \preceq_B r$ iff $q^\odot \preceq^I r^\odot$.*

Proof. (if) We define $\preceq$ to be a binary relation on Q such that $p \preceq r$ iff $p^\odot \preceq^I r^\odot$. We show that $\preceq$ is a backward simulation on Q which immediately implies the result.

Suppose that $p \preceq r$ and $p' \xrightarrow{a} \{p\} \cup P$ where $p \notin P$ is a transition of $\mathcal{A}$. Since $p \preceq r$, we know that $p^\odot \preceq^I r^\odot$; and since $p' \xrightarrow{a} \{p\} \cup P$ is a transition of $\mathcal{A}$, we know by definition of $\mathcal{A}^\odot$ that $p^\odot \xrightarrow{a} (p',a,P)^\odot$ and $(p',a,P)^\odot \xrightarrow{a} p'^\odot$ are transitions in $\mathcal{A}^\odot$. Since $\preceq^I$ is a simulation, we can find two transitions $r^\odot \xrightarrow{a} (r',a,R)^\odot$ and $(r',a,R)^\odot \xrightarrow{a} r'^\odot$ in $\mathcal{A}^\odot$ with $(p',a,P)^\odot \preceq^I (r',a,R)^\odot$ and $p'^\odot \preceq^I r'^\odot$. From $p'^\odot \preceq^I r'^\odot$, $(p',a,P)^\odot \preceq^I (r',a,R)^\odot$, and the definition of the initial preorder I, we have $p' \preceq r'$ and $P \preceq_F^{\forall\exists} R$. It follows that $\preceq$ is in fact a backward simulation parametrised by $\preceq_F$.

(only if) Define $\preceq_\odot$ as a binary relation on $Q^\odot$ such that $p^\odot \preceq_\odot r^\odot$ iff $p \preceq_B r$ and $(p,a,P)^\odot \preceq_\odot (r,a,R)^\odot$ iff $P \preceq_F^{\forall\exists} R$ and $p \preceq_B r$. By definition, $\preceq_\odot \subseteq I$. We show that $\preceq_\odot$ is a simulation on $Q^\odot$ which immediately implies the result. In the proof, we consider two sorts of states in $\mathcal{A}^\odot$; namely those corresponding to states and those corresponding to "environments".

Suppose that $p^\odot \preceq_\odot r^\odot$ and the transition $p^\odot \xrightarrow{a} (p',a,P)^\odot$ is in $\mathcal{A}^\odot$. Since $p^\odot \preceq_\odot r^\odot$, we know that $p \preceq_B r$. From the transition $p^\odot \xrightarrow{a} (p',a,P)^\odot$ and by definition of $\mathcal{A}^\odot$, $p' \xrightarrow{a} P \cup \{p\}$ is a transition in $\mathcal{A}$. Since $p \preceq_B r$, there exists a transition $r' \xrightarrow{a} R \cup \{r\}$ in $\mathcal{A}$ such that $p' \preceq_B r'$ and $P \preceq_F^{\forall\exists} R$. It follows that there exists a transition $r^\odot \xrightarrow{a} (r',a,R)^\odot$ in $\mathcal{A}^\odot$ such that $(p',a,P)^\odot \preceq_\odot (r',a,R)^\odot$.

Suppose that $(p,a,P)^\odot \preceq_\odot (r,a,R)^\odot$ and the transition $(p,a,P)^\odot \xrightarrow{a} p^\odot$ is in $\mathcal{A}^\odot$. Since $(p,a,P)^\odot \preceq_\odot (r,a,R)^\odot$, we know that $P \preceq_F^{\forall\exists} R$ and $p \preceq_B r$. By definition of $\mathcal{A}^{\odot}$, the transition $(r, a, R)^{\odot} \xrightarrow{a} r^{\odot}$ is in $\mathcal{A}^{\odot}$. Since $p \preceq_B r$, we have $p^{\odot} \preceq_{\odot} r^{\odot}$. Together we have there exists a transition $(r, a, R)^{\odot} \xrightarrow{a} r^{\odot}$ in $\mathcal{A}^{\odot}$ such that $p^{\odot} \preceq_{\odot} r^{\odot}$. It follows that $\preceq_{\odot}$ is a simulation on $Q^{\odot}$. □

6.4.2 Complexity of Computing Backward Simulation

The complexity comes from three parts of the procedure: (1) compiling $\mathcal{A}$ into its corresponding LTS $\mathcal{A}^{\odot}$, (2) computing the initial preorder I, and (3) running Algorithm 1 from Chapter 3 for computing the LTS simulation relation. Let n and m be the number of states and transitions in $\mathcal{A}$, respectively. The LTS $\mathcal{A}^{\odot}$ has at most $nm+n$ states and $2nm$ transitions. It follows that Part (3) has both time complexity and space complexity $\mathcal{O}(|\Sigma|n^2m^2)$. As we will show, among the three parts, Part (3) has the highest time and space complexity and therefore computing backward simulation also has time and space complexity $\mathcal{O}(|\Sigma|n^2m^2)$. Under our definition of ABA, every state has at least one outgoing transition for each symbol in Σ. It follows that $m \geq |\Sigma|n$. Therefore, we can also say that the procedure for computing maximal backward simulation has time and space complexity $\mathcal{O}(nm^3)$.

Initial Preorder for Computing Backward Simulation. Recall that the preorder I is the union of two components: $\{(q_1^{\odot}, q_2^{\odot}) \mid q_1 \preceq_{\iota} q_2 \wedge q_1 \preceq_{\alpha} q_2\}$ and $\{((p, a, P)^{\odot}, (r, a, R)^{\odot}) \mid \forall r_j \in R \exists p_i \in P : p_i \preceq_F r_j\}$. It is trivial that the first set can be computed by an algorithm with time complexity $\mathcal{O}(n^2)$. However, a naïve algorithm (pairwise comparison of all different environments in $\mathit{Env}(\mathcal{A})$) for computing the second set has time complexity $\mathcal{O}(n^4m^2)$. Here, we will describe a more efficient algorithm, which allows the computation of I in time $\mathcal{O}(n^2m^2)$ and space $\mathcal{O}(n)$.

The main idea of the algorithm is the following. For each pair of transitions of $\mathcal{A}$, it computes all the pairs of environments that arise from them (by deleting a right-hand side state) and are to be added to I at once, reusing a lot of information that a naïve algorithm would compute repeatedly for each pair of environments. For a fixed pair of transitions, this procedure has time complexity $\mathcal{O}(n^2)$ and space complexity $\mathcal{O}(n)$. Because $\mathcal{A}$ has at most m^2 different pairs of transitions and the $\mathcal{O}(n)$ memory needed for the data structures for one pair of transitions can then be reused for the other pairs, the second component of I can be this way computed in time $\mathcal{O}(n^2m^2)$ and space $\mathcal{O}(n)$.

We now explain how to efficiently compute all pairs of environments that arise from a given pair of transitions and that are related by I. Let us fix transitions $p \xrightarrow{a} P$ and $r \xrightarrow{a} R$. We will maintain a function $\beta : R \to \{T, F\} \cup P$ such that:

$$\beta(r') = \begin{cases} T & \text{if at least two states in } P \text{ are forward smaller than } r'. \\ F & \text{if no state in } P \text{ is forward smaller than } r'. \\ p' & \text{if } p' \text{ is the only state in } P \text{ such that } p' \preceq_F r'. \end{cases}$$

The function β can be computed by lines 1-4 of Algorithm 7 in time $\mathcal{O}(n^2)$ and space $\mathcal{O}(n)$.Let us consider a pair of states $((p, a, P \setminus \{p'\})^{\odot}, (r, a, R \setminus \{r'\})^{\odot})$ in $\mathcal{A}^{\odot}$. This pair can be added to I if and only if the following two conditions hold:

1. $\forall \hat{r} \in (R \setminus \{r'\}).\beta(\hat{r}) \neq F$.
2. $\forall \hat{r} \in (R \setminus \{r'\}).\beta(\hat{r}) \neq p'$.

The algorithm first pre-processes $p \xrightarrow{a} P$ and $r \xrightarrow{a} R$, computing certain information that will allow us to check the two conditions in constant time for every pair of environments arising from the two transitions.

The pre-processing needed for efficient checking of Condition (1) is the following. We define $\hat{r} \in R$ as the `KeyState` if $\hat{r}$ is the only one state in R such that $\beta(\hat{r}) = F$. Given a function β, the `KeyState` can be found efficiently (with time complexity $\mathcal{O}(n)$ and space complexity $\mathcal{O}(1)$) by scanning through R and

- if there exist two states $r_1, r_2 \in R$ such that $\beta(r_1) = \beta(r_2) = F$, the algorithm terminates immediately because it follows that none of the pairs of environments generated from the given pair of transitions satisfies the requirement of I;
- if there exists only one state such that β maps it to F, let it be the `KeyState`.

Then we have Condition (1) is satisfied if (1) there is no `KeyState` or (2) r' is the `KeyState`.

For efficient checking of Condition (2), we maintain a function $\gamma : P \to \{T, F\} \cup R$ such that

$$\gamma(p') = \begin{cases} F & \text{if } \beta^{-1}(p') = \emptyset \\ r' & \text{if } \beta^{-1}(p') = \{r'\} \\ T & \text{otherwise.} \end{cases}$$

The function γ can be found in time $\mathcal{O}(n^2)$ and space $\mathcal{O}(n)$ by scanning once through β for each element of P. With the function γ, Condition (2) can easily be verified by checking if $\gamma(p') \in \{F, r'\}$, which means that for all the states $\hat{r}$ in $R \setminus \{r'\}$, there is some state $\hat{p}$ different from p' such that $\hat{p} \preceq_F \hat{r}$. In Algorithm 7, we first find out the `KeyState` if there is one and compute the function γ from β. Then in the main loop, for each pair of states $((p, a, P \setminus \{p'\})^{\odot}, (r, a, R \setminus \{r'\})^{\odot})$, we check if it belongs to I by verifying the Conditions (1) and (2). Since it is easy to see that Algorithm 7 has time complexity $\mathcal{O}(n^2)$ and space complexity $\mathcal{O}(n)$ (not taking into account the space needed for I itself), we can conclude that the initial preorder I can be computed in time $\mathcal{O}(n^2 m^2)$ and space $\mathcal{O}(m^2)$ (encoding of I). This leads to the following theorem that summarises complexity of computing backward simulation.

Theorem 16. *Maximal backward simulation parametrised by a given transitive and reflexive forward simulation can be computed with both time and space complexity* $\mathcal{O}(|\Sigma| n^2 m^2) \subseteq \mathcal{O}(nm^3)$.

6.5 Experimental Results

In this section, we evaluate the performance of ABA mediated minimisation by applying it to accelerate the algorithm proposed by Vardi and Kupferman [KV01] for complementing nondeterministic Büchi automata (NBA). In

Algorithm 7: *Add Pairs of States to I*

```
Input: Two transitions $p \xrightarrow{a} P$ and $r \xrightarrow{a} R$ in $\mathcal{A}$.
/* Computing function $\beta$ */
1  forall $r' \in R$ do $\beta(r') := F$;
2  forall $p' \in P, r' \in R$ do
3      if $p' \preceq_F r'$ then
4          if $\beta(r') = F$ then $\beta(r') := p'$;
5          else $\beta(r') := T$;
/* Preprocessing for Condition (1) (computing KeyState) */
6  forall $r' \in R$ do if $\beta(r') = F$ then
7      if there is no KeyState then Let $r'$ be the KeyState;
8      else Terminate the algorithm;
/* Preprocessing for Condition (2) (computing function $\gamma$) */
9  forall $p' \in P$ do $\gamma(p') := F$;
10 forall $r' \in R$ do if $\beta(r') \notin \{T, F\}$ then
11     if $\gamma(\beta(r')) = F$ then $\gamma(\beta(r')) := r'$;
12     else $\gamma(\beta(r')) := T$;
/* main loop */
13 forall $p' \in P, r' \in R$ do
14     if there is no KeyState or $r'$ is the KeyState then
15         if $\gamma(p') \in \{F, r'\}$ then add $((p, a, P \setminus \{p'\})^{\odot}, (r, a, R \setminus \{r'\})^{\odot})$ to $I$
```

this algorithm, ABA's are used as an intermediate notion for the complementation. To be more specific, the complementation algorithm has two steps: (1) it translates an NBA to an ABA that recognises its complement language, and (2) it translates the ABA back to an equivalent NBA. The second step is an exponential procedure (exponential in the size of the ABA), hence reducing the size of the ABA before the second step usually pays off.

The experimentation is carried out as follows. Three sets of 100 random NBA's (of $|\Sigma| = 2$,4, and 8, respectively) are generated by the GOAL [TCT$^+$07] tool and then used as inputs of the complementation experiments. We compare results of experiments performed according to the following different options: (1) **Original:** keep the ABA as it is, (2) **Mediated:** minimising the ABA with mediated equivalence, and (3) **Forward:** minimising the ABA with forward equivalence.

For each input NBA, we first translate it to an ABA that recognises its complement language. The ABA is (1) processed according to one of the options described above and then (2) translated back to an equivalent NBA using an exponential procedure [4]. The results are given in Table 6.1 and Table 6.2. Table 6.1 is an overall comparison between the three different options and Table 6.2 is a more detailed comparison between **Mediated** and **Forward** minimisation.

[4]For the option "Original", we also use the optimisation suggested in [KV01] that only takes a consistent subset.

Table 6.1: Combining minimisation with complementation.

	$\lvert\Sigma\rvert$	NBA		Complemented-NBA		Time (ms)	Timeout (10 min)
		St.	Tr.	St.	Tr.		
Original	2	2.5	3.3	13.9	52.75	5500.9	0
Mediated				6.68	34.02	524.7	0
Forward				9.45	55.25	5443.7	1
Original	4	3.3	6.0	46.4	348.5	9298.6	6
Mediated				20.42	235.5	1985.4	6
Forward				26.88	325.6	1900.6	7
Original	8	4.7	11.9	127.1.3	1723.4	33429.4	24
Mediated				57.63	1738.3	12930.6	21
Forward				81.23	2349.2	22734.2	24

Table 6.2: Comparison: *Mediated* vs. *Forward*

	$\lvert\Sigma\rvert$	Minimised-ABA		Complemented-NBA	
		St.	Tr.	St.	Tr.
Average Difference	2	33.54%	51.62%	63.3%	235.56%
	4	36.24%	51.44%	89.9%	298.99%
	8	27.94%	40.88%	152.3%	412.7%

In Table 6.1, the columns “NBA” and “Complemented-NBA” are the average statistical data of the input NBA and the complemented NBA. The column “Time(ms)” is the average execution time in milliseconds. “Timeout” is the number of cases that cannot finish within the timeout period (10 min). Note that in the table, the cases that cannot finish within the timeout period are excluded from the average number. From this table, we can see that minimisation by mediated equivalence can effectively speed up the complementation and also reduce the size of the complemented NBA’s.

In Table 6.2, we compare the performance between **Mediated** and **Forward** minimisation in detail. The columns “Minimised-ABA” and “Complemented-NBA” are the average difference in the sizes of the ABA after minimisation and the complemented BA. From the table, we observe that mediated minimisation results in a much better reduction than forward minimisation.

6.6 Conclusion and Future Work

We have introduced a novel notion of alternating automata backward simulation. Inspired by our previous work on tree automata simulation reduction, we combined forward and backward simulation to form a coarser relation called mediated preorder and showed that quotienting wrt. mediated equivalence preserves the language of ABA. Moreover, we developed an efficient algorithm for computing backward simulation and mediated equivalence. Experimental results show that the mediated reduction of ABA significantly outperforms the reduction based on forward simulation.

In the future, we would like to extend our experiments to other applications such as LTL to NBA translation. Furthermore, we would like to extend the mediated equivalence by building it on top of even coarser forward simulation relations, e.g., *delayed* or *fair* forward simulation relations [FW05]. Also, we would like to study the possibility of using mediated preorder to remove redundant transitions (similar to the approaches described in [SB00]). We believe that the extensions described above can significantly improve the performance of mediated reduction.

7 Conclusions and Future Directions

Each of the main chapters contains detailed conclusions concernign the specific topic. Here, we summarise once more the main points and discuss possible further research directions.

7.1 A Summary of the Contributions

The main focus of this thesis was on developing efficient methods for handling nondeterministic tree automata. We have studied simulation based methods for size reduction of tree automata and methods for universality and language inclusion testing. We have found efficient algorithms for computing tree automata simulations that are based on translating problems of computing tree automata simulations to problems of computing common simulation over LTS. For this, we developed an efficient LTS simulation algorithm which is an extension of the fastest Kripke structure simulation algorithm. The same TA to LTS translations as for the TA simulations can be used also for computing tree automata bisimulations. Thus, all tree automata (bi)simulations can be computed in a uniform and elegant way, with possibility of using the most efficient LTS simulation and bisimulation algorithms. We have discovered a new type of relations that we call mediated equivalences that can be used for quotienting tree automata as well as for word automata. Mediated equivalence arises from a combination of upward and downward simulation, it includes downward simulation and thus gives a better reduction, as we confirm also experimentally. Since the combination principle allows also combining simulations with bisimulations, we have obtained a scale of TA mediated equivalences that offer a fine choice between reduction power and computational cost.

To solve language inclusion problem for tree automata, we have adapted the so called antichain universality and inclusion checking method for FA [DR10]. According to our experiments, this optimisation of the classical subset construction method leads to a major speed-up of the TA language inclusion and universality tests. We then improve the antichain method for both FA and TA by interconnecting it with the simulation based methods. This again significantly improves efficiency of the algorithms.

We have shown practical applicability of the above TA reduction and inclusion testing methods by applying them in the framework of abstract regular tree model checking. These algorithms allowed us to build a version of ARTMC method purely on nondeterministic tree automata, avoiding determinisation completely. According to our experiments, this greatly improved efficiency and scalability of the ARTMC method.

Since our tree automata reduction methods are based on quite simple and general principles, applying them for other types of automata comes into consideration. We have done this for alternating Büchi automata, for which we have introduced a notion of backward simulation and defined the mediated equivalence analogically as in the case of tree automata. As shown by our experiments, mediated equivalence gives very good reduction even in the case of ABA.

7.2 Further Directions

There is a number of interesting directions of further work. We have already started to work on an algorithm for computing simulation on Kripke structures and LTS that would match the best time complexity of the algorithm [RT07] and also the best space complexity of the algorithm [GPP03]. We are considering extensions of our simulation reduction methods to other types of automata, such as hedge automata, weighted tree automata, or nested word automata. Also the mediation principle itself can be further elaborated. We already have some preliminary results suggesting that it is possible to define a hierarchy of coarser and coarser relations similar to the mediated equivalence (and suitable for quotienting automata), where a mediated relation of level i is used to induce a mediated relation of level $i + 1$. The finite automata minimisation/reduction is an interesting problem itself and we are thinking about reduction techniques based on other principles than simulation quotienting. For instance, an efficient reduction heuristic based on the theory of universal automaton [ADN92, Pol05, KW70, Car70] could possibly be designed.

Further, we are still working on the tree automata language inclusion problem. We are developing a universality and language inclusion checking algorithm for tree automata that proceeds downwards (wrt. tree automata transition relation) and makes use of downward simulation, in contrary to the upward algorithm from Chapter 5 that exploits only upward simulation. Similarly as our reduction techniques, our language inclusion and universality antichain/simulation techniques can be extended for other types of automata. We have shown this in [ACC$^+$10a] for the Büchi automata language inclusion problem and we are continuing the work on this topic. Further, we do not restrict ourselves to simulation based techniques. One could think for instance about using some abstraction techniques as in [GMR09], and it may also be interesting to look for inspiration at the areas of decision procedures of logics or solving other hard problems such as QBF.

Our work on alternating Büchi automata simulation reduction can be continued in the way of looking at more advanced handling of Büchi acceptance condition. More specifically, we would like to study possibilities of constructing a mediated equivalence from delayed or fair simulation [FW05], which could lead to even better reductions.

Last, we are working towards applying our methods in practice. We are developing an efficient BDD based library that would provide procedures for handling nondeterministic tree automata (in the style of [KM01]). This work includes also a development of BDD versions of our algorithms, which is itself an interesting problem. We are also working on an ARTMC based method for verification of pointer manipulating programs that will make use of our TA reduction and language inclusion checking techniques.

7.3 Publications Related to this Thesis

The algorithm for computing simulations over labelled transition systems appeared in [ABH+08c]. The tree automata reduction methods and algorithms for computing simulations and bisimulations were published in [ABH+08c, ABH+09, AHKV09]. The generalisation of the antichain universality and language inclusion method for TA appeared in [BHH+08b, ACH+10a]. The combination of the antichain and simulation methods was published in [ACHV09a]. Finally, the results on ABA simulation reduction are from [ACC+10a].

The following publications are also to a large degree outcomes of work on this thesis. The work [HŠ09a] presents optimizations of the algorithm for computing simulations on LTS from Chapter 3. In [HR07], we fix some problems in counterexample guided refinement loop for complex systems that were discovered within the work on the ARTMC tool presented in Section 5.3.3. The work [ACC+10a] presents an application of our simulation based subsumption principle in Büchi automata inclusion testing.

Full versions of the above mentioned papers were published as the technical reports [ABH+07, BHH+08a, AHKV08a, ACH+10b, ABH+08a, ACC+10b, ACHV09b, HŠ09b]. The works [ABH+09] and [AHKV09] first appeared as [ABH+08b] and [AHKV08b].

Bibliography

[ABH$^+$07] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. Computing Simulations over Tree Automata: Efficient Techniques for Reducing Tree Automata. Technical Report FIT-TR-2007-01, FIT BUT, Brno, Czech Republic, 2007.

[ABH$^+$08a] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. Composed Bisimulation for Tree Automata. Technical Report FIT-TR-2008-04, FIT BUT, Brno, Czech Republic, 2008.

[ABH$^+$08b] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. Composed Bisimulation for Tree Automata. In *CIAA'08*, volume 5148 of *LNCS*. Springer, 2008.

[ABH$^+$08c] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. Computing Simulations over Tree Automata: Efficient Techniques for Reducing Tree Automata. In *TACAS'08*, volume 4963 of *LNCS*, pages 93–108. Springer, 2008.

[ABH$^+$09] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. Composed Bisimulation for Tree Automata. *Int. J. Found. Comput. Sci.*, 20(4):685–700, 2009.

[ACC$^+$10a] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *CAV'10*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.

[ACC$^+$10b] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong Hong, Richard Mayr, and Tomáš Vojnar. Simulation Subsumption in Ramsey-based Büchi Automata Universality and Inclusion Testing. Technical Report FIT-TR-2010-02, FIT BUT, Brno, Czech Republic, 2010.

[ACH$^+$10a] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. When Simulation Meets Antichains (on Checking Language Inclusion of NFAs). In *TACAS'10*, volume 6015 of *LNCS*, pages 158–174. Springer, 2010.

[ACH$^+$10b] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. When Simulation Meets Antichains (on Checking Language Inclusion of NFAs). Technical Report FIT-TR-2010-01, FIT BUT, Brno, Czech Republic, 2010.

[ACHV09a] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, and Tomáš Vojnar. Mediating for Reduction (on Minimizing Alternating Büchi Automata). In *FSTTCS'09*, volume 4 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[ACHV09b] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, and Tomáš Vojnar. Mediating for Reduction (On Minimizing Alternating Büchi Automata). Technical Report FIT-TR-2009-02, FIT BUT, Brno, Czech Republic, 2009.

[ADN92] André Arnold, Anne Dicky, and Maurice Nivat. A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169, 1992.

[AHK07] Parosh Aziz Abdulla, Johanna Högberg, and Lisa Kaati. Bisimulation Minimization of Tree Automata. *Int. J. Found. Comput. Sci.*, 18(4):699–713, 2007.

[AHKV08a] Parosh Aziz Abdulla, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. A Uniform (Bi-)Simulation-Based Framework for Reducing Tree Automata. Technical Report FIT-TR-2008-05, FIT BUT, Brno, Czech Republic, 2008.

[AHKV08b] Parosh Aziz Abdulla, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. A Uniform (Bi-)Simulation-Based Framework for Reducing Tree Automata. In *MEMICS'08*, 2008.

[AHKV09] Parosh Aziz Abdulla, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. A Uniform (Bi-)Simulation-Based Framework for Reducing Tree Automata. *Electr. Notes Theor. Comput. Sci.*, 251:27–48, 2009.

[AJMd02] Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso. Regular Tree Model Checking. In *CAV'02*, volume 2404 of *LNCS*, pages 555–568. Springer, 2002.

[ALdR05] Parosh Aziz Abdulla, Axel Legay, Julien d'Orso, and Ahmed Rezine. Simulation-Based Iteration of Tree Transducers. In *TACAS*, volume 3440 of *LNCS*, pages 30–44. Springer, 2005.

[ALdR06] Parosh Aziz Abdulla, Axel Legay, Julien d'Orso, and Ahmed Rezine. Tree Regular Model Checking: A Simulation-Based Approach. *J. Log. Algebr. Program.*, 69(1-2):93–121, 2006.

[BHH$^+$08a] Ahmed Bouajjani, Peter Habermehl, Lukáš Holík, Tayisir Touili, and Tomáš Vojnar. Antichain-based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. Technical Report FIT-TR-2008-01, FIT BUT, Brno, Czech Republic, 2008.

[BHH$^+$08b] Ahmed Bouajjani, Peter Habermehl, Lukáš Holík, Tayssir Touili, and Tomáš Vojnar. Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. In *CIAA'08*, volume 5148 of *LNCS*, pages 57–67. Springer, 2008.

[BHMV05] Ahmed Bouajjani, Peter Habermehl, Pierre Moro, and Tomáš Vojnar. Verifying Programs with Dynamic 1-Selector-Linked Structures in Regular Model Checking. In *TACAS'05*, volume 3440 of *LNCS*, pages 13–29. Springer, 2005.

[BHRV06a] Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomáš Vojnar. Abstract Regular Tree Model Checking. *Electr. Notes Theor. Comput. Sci.*, 149(1):37–48, 2006.

[BHRV06b] Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomáš Vojnar. Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *SAS'06*, pages 52–70, 2006.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomáš Vojnar. Abstract Regular Model Checking. In *CAV'04*, volume 3114 of *LNCS*, pages 372–386. Springer, 2004.

[Brz62] Janusz A. Brzozowski. Canonical Regular Expressions and Minimal State Graphs for Definite Events. In *Mathematical Theory of Automata*, volume 12 of *MRI Symposia Series*, pages 529–561, Polytechnic Institute of Brooklyn, NY, 1962. Polytechnic Press.

[BT02] Ahmed Bouajjani and Tayssir Touili. Extrapolating Tree Transformations. In *CAV'02*, volume 2404 of *LNCS*, pages 539–554. Springer, 2002.

[Buc08] Peter Buchholz. Bisimulation relations for weighted automata. *Theor. Comput. Sci.*, 393(1-3):109–123, 2008.

[Car70] Christian Carrez. On the minimalization of non-deterministic automaton. *Laboratoire de Calcul de la Faculté des Sciences de l'Université de Lille*, 1970.

[CDG$^+$07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1989.

[CRT09] Silvia Crafa, Francesco Ranzato, and Francesco Tapparo. Saving Space in a Time Efficient Simulation Algorithm. In *ACSD'09*, pages 60–69. IEEE, 2009.

[DGG93] Dennis Dams, Orna Grumberg, and Rob Gerth. Generation of Reduced Models for Checking Fragments of CTL. In *CAV'93*, volume 697 of *LNCS*, pages 479–490. Springer, 1993.

[DHWT91] David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for Language Inclusion Using Simulation Preorders. In *CAV'91*, volume 575 of *LNCS*, pages 255–265. Springer, 1991.

[DR10] Laurent Doyen and Jean-François Raskin. Antichain Algorithms for Finite Automata. In *TACAS'10*, volume 6015 of *LNCS*, pages 2–22. Springer, 2010.

[FCC$^+$08] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In *TACAS'08*, volume 4963 of *LNCS*, pages 2–17. Springer, 2008.

[FV09] Seth Fogarty and Moshe Y. Vardi. Büchi Complementation and Size-Change Termination. In *TACAS'09*, volume 5505 of *LNCS*, pages 16–30. Springer, 2009.

[FW02] Carsten Fritz and Thomas Wilke. State Space Reductions for Alternating Büchi Automata. In *FSTTCS'02*, pages 157–168, London, UK, 2002. Springer.

[FW05] Carsten Fritz and Thomas Wilke. Simulation relations for alternating Büchi automata. *Theor. Comput. Sci.*, 338(1-3):275–314, 2005.

[GKSV03] Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On Complementing Nondeterministic Büchi Automata. In *CHARME'03*, volume 2860 of *LNCS*, pages 96–110. Springer, 2003.

[GL94] Orna Grumberg and David E. Long. Model Checking and Modular Verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[GMR09] Pierre Ganty, Nicolas Maquet, and Jean-François Raskin. Fixpoint Guided Abstraction Refinement for Alternating Automata. In *CIAA'09*, volume 5642 of *LNCS*, pages 155–164. Springer, 2009.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi Automata Translation. In *CAV'01*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

[GPP03] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From Bisimulation to Simulation: Coarsest Partition Problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[GVT03] Thomas Genet, Valérie Viet, and Triem Tong. Timbuk: A Tree Automata Library. `http://www.irisa.fr/lande/genet/timbuk`, 2003.

[HHK95] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *FOCS'95*, pages 453–462, Washington, DC, USA, 1995. IEEE.

[HMM07a] Johanna Högberg, Andreas Maletti, and Jonathan May. Backward and Forward Bisimulation Minimisation of Tree Automata. In *CIAA'07*, volume 4783 of *LNCS*, pages 109–121. Springer, 2007.

[HMM07b] Johanna Högberg, Andreas Maletti, and Jonathan May. Bisimulation Minimisation for Weighted Tree Automata. In *DLT'08*, volume 4588 of *LNCS*, pages 229–241. Springer, 2007.

[Hop71] John E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.

[HR07] Lukáš Holík and Adam Rogalewicz. Counterexample Analysis in Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *MEMICS'07*, pages 59–66, 2007.

[HŠ09a] Lukáš Holík and Jiří Šimáček. Optimizing an LTS-Simulation Algorithm. In *MEMICS'09*, pages 93–101. Faculty of Informatics MU, 2009. An extended version accepted at Computing and Informatics.

[HŠ09b] Lukáš Holík and Jiří Šimáček. Optimizing an LTS-Simulation Algorithm. Technical Report FIT-TR-2009-03, FIT BUT, Brno, Czech Republic, 2009.

[KM01] Nils Klarlund and Anders Møller. MONA Version 1.4 User Manual, 2001. BRICS, Department of Computer Science, University of Aarhus, Denmark.

[KV01] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

[KW70] T. Kameda and P. Weiner. On the State Minimization of Nondeterministic Finite Automata. *IEEE Trans. Comput.*, 19(7):617–627, 1970.

[Møl04] Anders Møller. `http://www.brics.dk/automaton`, 2004.

[MS72] Albert R. Meyer and Larry J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *FOCS'72*, pages 125–129. IEEE, 1972.

[Pol05] Libor Polák. Minimalizations of NFA Using the Universal Automaton. *Int. J. Found. Comput. Sci*, 16(5):999–1010, 2005.

[PT87] Robert Paige and Robert Endre Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[RT07] Francesco Ranzato and Francesco Tapparo. A new efficient simulation equivalence algorithm. In *LICS'07*, pages 171–180. IEEE, 2007.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. In *CAV'00*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.

[Sha01] E. Shahar. *Tools and Techniques for Verifying Parameterized Systems.* PhD thesis, Faculty of Mathematics and Computer Science, The Weizmann Inst. of Science, Rehovot, Israel, 2001.

[SJ05] Zdenek Sawa and Petr Jancar. Behavioural Equivalences on Finite-State Systems are PTIME-hard. *Computers and Artificial Intelligence*, 24(5), 2005.

[SVW85] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic (Extended Abstract). In *ICALP'85*, volume 194 of *LNCS*, pages 465–474. Springer, 1985.

[TCT$^+$07] Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Kang-Nien Wu, and Wen-Chin Chan. GOAL: A Graphical Tool for Manipulating Büchi Automata and Temporal Formulae. In *TACAS'07*, volume 4424 of *LNCS*, pages 466–471. Springer, 2007.

[TV05] Deian Tabakov and Moshe Y. Vardi. Experimental Evaluation of Classical Automata Constructions. In *LPAR'05*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.

[Val09] Antti Valmari. Bisimilarity Minimization in $\mathcal{O}(m \log n)$ Time. In *Petri Nets*, volume 5606 of *LNCS*, pages 123–142. Springer, 2009.

[Var07] Moshe Y. Vardi. Automata-Theoretic Model Checking Revisited. In *VMCAI'07*, volume 4349 of *LNCS*, pages 137–150. Springer, 2007.

[WDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A New Algorithm for Checking Universality of Finite Automata. In *CAV'06*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.