Universität des Saarlandes
Naturwissenschaftlich-Technische Fakultät 1
Fachrichtung Informatik
Bachelor-Studiengang Informatik

Bachelor's Thesis

Minimization of Tree Automata

submitted by

Thomas von Bomhard

on 11.09.2008

Supervisor

Prof. Dr. Bernd Finkbeiner

Advisor

Dr. Sven Schewe

Author:	Thomas von Bomhard
1st Reviewer:	Prof. Dr. Bernd Finkbeiner
2nd Reviewer:	Dr. Sven Schewe

Abstract

Many verification and synthesis processes are based on finite automata on infinite inputs. It is therefore important to develop techniques that minimize the considered automata as far as possible.

For Büchi word automata, there are established minimization techniques such as deletion of nonproductive states and simulation-based quotienting. This thesis transfers these techniques to alternating Büchi tree automata.

We adapt three different types of simulation relations to alternating Büchi tree automata and study which of them are useful for minimization. Furthermore, we evaluate the developed techniques on a prototype.

Contents

1 Introduction

A big challenge in computer science is the construction of provably correct systems, this means to ensure that a system behaves as it is intended to do. There are essentially two approaches:

- Verification: Check if the implementation satisfies logical properties.
- Synthesis: Derive correct-by-construction implementations from logical specifications.

Consequently, verification and synthesis problems are logical decision problems.

Temporal logics are widely used languages for specifying programs. There are two variations of temporal logics, linear and branching [18]. In linear-time temporal logics (LTL), a point of time has a unique possible future, whereas in branching-time temporal logics (CTL), a point of time may split in to several possible futures.

In the early sixties, Büchi introduced finite automata on infinite words (Büchi automata) [2, 3], and in the late sixties, Rabin introduced finite automata on infinite trees [23]. They are important tools for program verification, because decision problems of temporal logics can be reduced to automata-theoretic problems. Furthermore, these automata are also important for modeling the non-terminating behavior of reactive systems.

A prominent example that shows the use of Büchi automata, is the approach of Wolper and Vardi [26]: A linear temporal formula ϕ that specifies some undesired property of the system, is translated to a Büchi automaton A_ϕ. The accepted words of A_ϕ represent executions, in which the undesirable property holds. Then the reactive system is interpreted as a Büchi automaton A_M. Finally, the intersection automaton of A_M and A_ϕ determines whether there exists executions that exhibit the undesired property. Many LTL - model checkers, such as SPIN [15], follow this approach.

For branching-time temporal logics, the corresponding automata-theoretic formalism is the theory of tree automata. The automata-theoretic approach to model checking for branching-time logics is described in [17].

From a practical perspective, it is important that the considered automata are as small as possible. Unfortunately, computing for a given Büchi automaton a minimal lanugage equivalent Büchi automaton is PSPACE-hard, hence heuristic methods are applied. A frequently used minimization technique is state-space reduction via simulation relations [20]. The main idea is to identify states where one state can be simulated by another state or in other words all moves from one state can be mimicked by another state.

Simulation relations guarantee language containment. This means all words accepted from one state are also accepted by the state that simulates it. As a consequence, if several states mutually simulate each other, they are language equivalent. Then it may be possible to merge them together in a single state such that the language of the automaton is preserved. This is called minimization by quotienting. It is done for nondeterministic Büchi word automata (NBWA) in [11]. There is previous work on simulation relations for NBWA in [24, 10, 14]. Moreover, minimization by simulation-based quotienting is also done for alternating Büchi word automata (ABWA), see [12].

This thesis presents minimization techniques for alternating Büchi tree automata (ABTA).

Chapter 2 considers conservative but cheap tests for detecting empty and universal states of tree automata. A state is empty or universal, if the automaton starting from the state accepts no or all input trees, respectively. Such states can be eliminated without changing the language of the automaton.

In Chapter 3, we follow the approaches of [12, 11] and adapt the simulation relations to alternating Büchi tree automata. We focus on three types of simulation relations, namely direct [7], delayed [11] and fair simulation [14]. We show that all types of simulation guarantee language containment. Furthermore, we demonstrate that direct and delayed simulation can be used for minimization by quotienting. Also, we show that computing the simulations can be done fast and efficiently.

We have implemented a prototype for benchmarking the minimization techniques. We apply them on randomly generated ABTA and consider the minimization profit of each technique.

2 Preliminaries

In the following we introduce the basic definitions and notations of Büchi word automata, Büchi tree automata, an ABTA to NBTA translation and parity games.

2.1 Words and Trees

For a given nonempty finite alphabet Σ, a finite **word** w is a finite sequence of letters of Σ, formally speaking $w \in \Sigma^*$. An infinite word w' is an element of Σ^ω.

Let N_k be the set $\{0, 1, 2 \ldots k-1\}$. A **k-tree** is a prefix-closed set of finite sequences of elements of N_k. More formally, it is a subset T of $(N_k)^*$ such that

- the empty sequence ϵ is the root node of T
- if a node $x_0 x_1 \ldots x_{n-1}$ is in T
  then the nodes $x_0 x_1 \ldots x_{m-1}$ for all $m < n$, are also in T

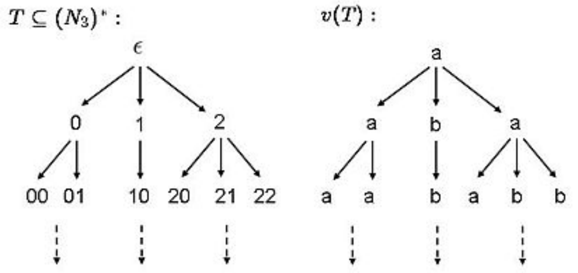
A k-tree where each node has exactly k child nodes is a **full** k-tree. A **subtree** of T with x as root node is $T^x = \{x \cdot y \mid y \in T\}$. The **tree of depth** $l \in \mathbb{N}$ of T is $T_l = \{x \mid x \in T \text{ and } \mathrm{depth}(x) \leq l\}$ where depth(x) is the number of edges from x to the root.

A **path** $\pi = x_0, x_1, \ldots$ of a tree T is a maximal sequence of nodes such that x_0 is the root and x_i is the parent of x_{i+1} for all $i > 0$.

We will use trees where the nodes are labeled with a symbol of an alphabet. A Σ-labeled tree, for a finite alphabet Σ, is a pair (T, v), where T is a tree and

v a mapping $v : T \to \Sigma$. We often refer to v as the labeled tree, leaving its domain implicit. We denote the set of all Σ-labeled trees $\mathcal{T}_\Sigma$.

We use the mapping v also for paths of T, so for a path $\pi = x_0, x_1, \ldots$ we get an Σ-labeled path by $v(\pi) = v(x_0), v(x_1), \ldots$ which is the word along the path π.



2.2 Büchi Automata

In the following, we give the definition and notation for alternating Büchi tree automata (ABTA). Then we define nondeterministic Büchi tree automata (NBTA), alternating Büchi word automata (ABWA) and nondeterministic Büchi word automata (NBWA) as special cases of ABTA.

We follow the formalism of [25] which goes back to [5]. Let X be a set. Let $\mathcal{B}^+(X)$ be the set of all positive boolean formulas over X, which are formulas built from elements of X using the operators $\wedge$ and $\vee$. We also allow the formulas *true* and *false*. Let $Z \subseteq X$. We say that Z satisfies a formula $\theta \in \mathcal{B}^+(X)$ if the truth assignment that assigns true to the members of Z and assigns false to the members of $X \backslash Z$ satisfies θ. Moreover, Z minimal satisfies θ if it satisfies θ and all strict subsets of Z does not satisfy θ. We denote by mss(θ) the set of all minimal satisfying sets of θ.

An **alternating Büchi tree automaton** on infinite k-trees (ABTA) is a tuple.

$$A = (\Sigma, Q, q_I, \delta, F)$$

where Σ denotes a finite alphabet, Q denotes a finite set of states, $q_I \in Q$ denotes a designated initial state, δ denotes the total transition function and $F \subseteq Q$ is the set of accepting states. The transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(N_k \times Q)$ maps a state and an input letter to a positive boolean combination of pairs of directions and states.

A **run** of A on a Σ-labeled full k-tree (T, v) is a $((N_k)^* \times Q)$-labeled k'-tree (T', r). Each node of r corresponds to a node in v. A node (x, q) describes that the automaton is in state q and reads the node x of v. A run tree has to satisfy following properties:

1. $r(\epsilon) = (\epsilon, q_I)$

2. Let $(x, q) = r(y)$ and $\theta = \delta(q, v(x))$. Then there is a set $Z = \{(c_0, q_0), (c_1, q_1), \dots, (c_n, q_n)\} \subseteq N_k \times Q$ such that

    - Z satisfies θ, and
    - for all $0 \le i \le n$, we have $r(y \cdot i) = (x \cdot c_i, q_i)$.

A run is accepting if there exist infinitely many accepting states on each path. Formally, the **infinity set** of a path of a run is the set:

$$\mathit{inf}(r|\pi) = \{q \in Q \mid r(x) = (n_x, q) \text{ for infinitely many } x \in \pi\}$$

A run (T', r) is **accepting** if for all paths π of T', $\mathit{inf}(r|\pi) \cap F \neq \emptyset$ holds true. An automaton A accepts a tree v if there exists an accepting run of A on v. The set of input trees accepted by A is called its **language** $\mathcal{L}(A)$.

A **nondeterministic** automaton is a special alternating automaton, where the image of δ consists only of such formulas that, when rewritten in disjunctive normal form, contain exactly one element of $Q \times \{d\}$ for all $d \in N^k$ in every disjunct. An automaton on infinite **words** is a tree automaton on infinite 1-trees.

For convenience, we denote by A^q the ABTA A with q as initial state. For simplicity of the definition of the games on ABTA, we assume that an ABTA has two distinct states q_T and q_F that simulate *true* and *false*, respectively. Precisely, q_T is an accepting state and $\delta(q_T, a) = (0, q_T) \wedge \dots \wedge (k-1, q_T)$ for all $a \in \Sigma$. Symmetrically, q_F is an non-accepting state and $\delta(q_F, a) := (0, q_F) \wedge \dots \wedge (k-1, q_F)$ for all $a \in \Sigma$. So, a formula $\delta(q, a) = \mathit{true}$ of the ABTA will be rewritten as $\delta(q, a) = (0, q_T) \wedge \dots \wedge (k-1, q_T)$. Analogously for *false*.

2.3 From ABTA to NBTA

In the following, we briefly describe a construction for converting an alternating Büchi tree automaton to a nondeterministic Büchi tree automaton. The construction is the straightforward extension of the Miyano-Hayashi construction [21], which translates an ABWA to a NBWA, to tree automata.

It is a subset construction modified for de-universalization instead of determinization. The states are pairs (S, R) of subsets of the state set Q. The first component is used as in normal subset constructions that is, if there is a state q in S with $\delta(q, a) = (0, q_0) \wedge (0, q_1) \wedge (1, q_2)$, and $\delta'((S, R), a) = (0, (S_0, R_0)) \wedge (1, (S_1, R_1))$, then $q_0, q_1 \in S_0$ and $q_2 \in S_1$. The second component is for bookmarking the run tree branches with an obligation to reach an accepting state. A state is removed from the second component as soon as its run tree branch reaches an accepting state. A state (S, R) with $R = \emptyset$ is an accepting state and if (S', R') is an successor state of (S, R), then $R' = S' \backslash F$.

For an ABTA $A = (Q, \Sigma, q_I, \delta, F)$ on infinite k-tree, the automaton resulting by the MH-Construction is the NBTA $A' = (Q', \Sigma', q'_I, \delta', F')$ on infinite k-tree with

$$Q' \subseteq 2^Q \times 2^Q$$
$$\Sigma' = \Sigma$$
$$q'_I = (\{q_I\}, \{q_I \mid q_I \notin F\})$$

We define a function Υ_i that maps a satisfying set of a formula to the set of states with direction i. More formally Υ_i maps a set Z of $N^k \times Q$ to the set $\{q \mid (d, q) \in Z \wedge d = i\}$ of Q.

For $S, R \subseteq Q$, $a \in \Sigma$, $\phi_S = \bigwedge_{q \in S} \delta(q, a)$ and $\phi_R = \bigwedge_{q \in R} \delta(q, a)$ we define the transition function δ' as follows:

If $R \neq \emptyset$:

$$\delta'((S, R), a) = \bigvee_{Z \,\in\, \mathrm{mss}(\phi_S),\ Z' \in\, \mathrm{mss}(\phi_R),\ Z' \subseteq Z} \Big(\bigwedge_{0 \leq i < k} (i, (\Upsilon_i(Z), \Upsilon_i(Z') \backslash F)) \Big)$$

If $R = \emptyset$:

$$\delta'((S, R), a) = \bigvee_{Z \in mss(\phi_S)} \Big(\bigwedge_{0 \leq i < k} (i, (\Upsilon_i(Z), \Upsilon_i(Z) \backslash F)) \Big)$$

$$F' = 2^Q \times \{\emptyset\}$$

Note that the NBTA A' has $O(2^n)$ states, where n is the number of states of the ABTA A.

2.4 Parity Games

A **parity game** $G = (V_{odd}, V_{even}, E, c)$ consists of a finite directed game graph $(V_{odd} \cup V_{even}, E)$, whose sets of vertices V_{odd} and V_{even} are the positions of the players *odd* and *even*, respectively. V_{odd} and V_{even} are disjoint sets and we denote as V the union of V_{odd} and V_{even}. $E \subseteq V \times V$ is the edge relation which is also called move relation. The coloring function $c : V \to 0, 1, \ldots k$ maps each game position $v \in V$ to an integer $c(v)$ which we call a color.

Plays: Intuitively, a parity game is played by placing a token on a vertex $v \in V$. If v is a position of *odd*, then *odd* chooses an edge $(v, v') \in E$ and moves the token to position v'. Symmetrically, if v is a position of *even*, then *even* chooses an edge $(v, v') \in E$ and moves the token to position v'. This is repeated either infinitely often or finitely, if some player can't move, which means the position of the player has no outgoing edges. Such a position is called a sink. Let $V_S \subseteq V$ be the set of all sinks and $V_C \subseteq V$ the set of "continuative" positions which means that a position $v \in V_C$ has at least one outgoing edge $(v, v') \in E$.
A play is either:
an infinite path $\Pi = v_0 v_1 v_2 \cdots \in {V_C}^\omega$ with $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$ or
a finite path $\Pi = v_0 v_1 v_2 \ldots v_l \in {V_C}^* V_S$ with $(v_i, v_{i+1}) \in E$ for all $i < l$.
We call these plays correspondingly infinite play and finite play.

Winning Conditions: For a finite play $\Pi = v_0 v_1 \dots v_l$, the player *even* wins the play if the last position v_l, the sink, belongs to *odd*. Symmetrically, the player *odd* wins the play if the sink v_l belongs to *even*.
For a infinite play $\pi = v_0, v_1, v_2, \dots$, the lowest color which occurs infinitely often is deciding. So player *even* (*odd*) wins the play if the lowest color occurring infinitely often in $c(\pi) = c(v_0), c(v_1), c(v_2) \dots$ is even (odd). We call such parity games min-parity games.

A **strategy** for *even* is a partial function $f : V^\star V_{even} \rightarrow V$ which maps the history of a play which ends in a position v of *even* to a successor v' of v. (That means there is an edge $(v, v') \in E$). A play is called f*-conform* if each decision of *even* in the play is in accordance with f. A memoryless strategy for player *even* is a function $g : V_{even} \rightarrow V$ such that $(v, g(v)) \in E$ for all $v \in V_{even}$. Its called memoryless because the choice of the next move depends only on the current position and not on the history of the game which needs memory. The notions for player *odd* are defined analogously.
A strategy f of player *even* (*odd*) is called v-winning if all f-conform plays that start in v are winning for player *even* (*odd*). A position $v \in V$ is v-winning for player *even* (*odd*) if *even* (*odd*) has a v-winning strategy. We call the sets of v-winning positions for player *even* (*odd*) the winning region of *even* (*odd*).

When we want to consider only plays which start at some designated node $v_I \in V$, we define an initialized game (G, v_I). A play of such a game is a play of the uninitialized game which starts in v_I. For initialized games, we say a player has a winning strategy f in (G, v_I), instead of a player has a v_I-winning strategy f in (G, v_I).

Theorem 1. *[19] Parity Games are memoryless determined:*
For every parity game G, the game positions are partitioned into a winning region W_{even} of player even and a winning region W_{odd} of player odd. Moreover, player even and odd have memoryless strategies that are v-winning for all positions in their respective winning region.

Theorem 2. *[16] Parity games with n positions, m edges and two or three colors can be solved in time $O(n\ m)$.*

3 Minimization of Büchi Tree Automata by Removing Trivial States

A simple minimization technique is the elimination of trivial states, namely empty and universal states. Formally, for an ABTA A, a state q is **empty** if $\mathcal{L}(A^q) = \emptyset$ and a state q is **universal** if $\mathcal{L}(A^q) = \mathcal{T}_\Sigma$. Obviously, empty and universal states of an ABTA can be replaced by q_t and q_f, respectively, without changing the language of the automaton.

Unfortunately, checking emptiness or universality of an ABTA is EXPTIME-complete [9, 8, 22], hence we use cheap conservative tests. That means, they are correct and efficient, but incomplete.

Technically, the methods are based on variations of the acceptance game of tree automata [13]. We call these modified games **pseudo emptiness game** and **pseudo universality game**. They are "pseudo" games, because they are incomplete.

3.1 Acceptance Game

At first we give a short description of the game and then we define it in terms of a parity game.

Description: Let $A = (\Sigma, Q, q_I, \delta, F)$ be an ABTA, q_0 be a state and v an input tree. The acceptance game is played by player *accept* and player *reject*, in rounds. Player *accept* will win if A^{q_0} accepts v and player *reject* will win if A^{q_0} does not accept v. At start round 0, the starting position is the state q_0 and letter $v(\epsilon)$. Assume the game is in round i, so in state q_i and letter $v(x)$, then it is played as follows:

1. Player *accept* chooses a set Z that satisfies $\delta(q_i, v(x))$
2. Player *reject* chooses an element (d, q_{i+1}) of Z

which determines the starting position, state q_{i+1} and letter $v(xd)$, for the next round.

Intuitively, player *accept* controls the existential choice of the ABTA and player *reject* controls the universal choice of the ABTA.

By playing the game, the players produce an infinite sequence of states that determines the winner: Player *accept* wins, if the sequence contains infinitely many accepting states, while player *reject* wins if the sequence contains finitely many accepting states.

Lets turn to the formal definition of the described game. The game is actually a Büchi game, but for convenience we define it as an equivalent 2-color parity game.

Definition: An acceptance game is an initialized min-parity game:

$$G_{(A,v)} = ((V_{odd}, V_{even}, E, c), q_I)$$

The notation $G_{(A,v)}$ means: The acceptance game for the input tree v on the ABTA A. Player *odd* takes over the role of player *reject* and player *even* the role of player *accept*. Each game position corresponds to a state of a round. The game graph is a straightforward construction:

Game positions:
A pair of a state and a sequence of directions is a position of player *accept* (A):
$V_A = Q \times \mathbb{N}^*$
For each position of player *accept*, we get positions of player *reject* (R) that consist of a satisfying set and a sequence of directions:
$V_R = \bigcup_{(q,x)\in V_A} \{(Z, x) \mid Z \text{ satisfies } \delta(q, a)\}$

Move relations:
$E_{V_A \times V_R} = \bigcup_{(q,x)\in V_A} \{((q, x), (Z, x)) \mid Z \text{ satisfies } \delta(q, v(x))\}$
$E_{V_R \times V_A} = \bigcup_{(Z,x)\in V_R} \{((Z, x), (q', xd)) \mid (d, q') \in Z\}$

Finally, $G_{A,v} = ((V_{odd}, V_{even}, E, c), q_I)$ is defined by:
$V_{odd} = V_R$
$V_{even} = V_A$
$E = E_{V_A \times V_R} \cup E_{V_R \times V_A}$
The coloring function is defined as:

$$c(v) = \begin{cases} 0 & \text{if } v = (q, n) \text{ and } q \in F \\ 1 & \text{otherwise.} \end{cases}$$

Theorem 3. *[13]: Player accept has a memoryless winning strategy in $G_{(A,v)}$ if and only if A accepts v.*

Proof.
$\rightarrow$: Let g be a memoryless winning strategy of player *accept* in $G_{(A,v)}$. We construct a run (r, T) of A on the input tree v by successively using g:

Let q_0 be the initial state of A, then we set $r(\epsilon) = (\epsilon, q_0)$.

Let $l \in \mathbb{N}$. Assume that for all nodes of T_l, the run r is already constructed by using strategy g.

(i) Let x be a leaf of T_l and $(n, q) = r(x)$. By using the strategy g, we get the set $Z = g(q, v(n))$. By definition of the game, Z satisfies $\delta(q, (v(n))$. We extend r by Z according the definition of a run of ABTA.

This construction step (i) can be done for all leaves of T_l. Thus, the run r is constructed for all nodes of T_{l+1}. By induction follows that the infinite construction of r is a run on v.
By construction, each path $r(\pi)$ corresponds to a g-conform play Π of $G_{(A,v)}$. Since the strategy g is winning, the play Π is winning, hence there are infinitely

many accepting states on $r(\pi)$. Finally, r is an accepting run.

$\leftarrow$: Let r be an accepting run of A on input v. Then, player *accept* has a memoryless strategy g for the acceptance game $G_{(A,v)}$ by playing exactly as the run r prescribes. Since the run r is accepting, there exists infinitely many accepting states on each path of r, so each g-conform play of $G_{(A,v)}$ is winning. Thus the strategy g is winning. □

3.2 Pseudo Emptiness Game

The pseudo emptiness game is a modified acceptance game, where, in each round, player *accept* additionally chooses a letter of the alphabet.

We show that this game is correct for ABTA, that means, if player *reject* has a memoryless winning strategy in the pseudo emptiness game on some ABTA A^q, the state q is empty. While the game is incomplete for ABWA, it is complete for NBTA.

Furthermore, we demonstrate that the game is efficiently solvable by a parity game solving algorithm.

Definition: The pseudo emptiness game is an initialized min-parity game $G^{\perp}_{A}(q) = ((V_{odd}, V_{even}, E, c), q)$. The notation $G^{\perp}_{A}(q)$ means: the pseudo emptiness game $G^{\perp}$, where $\perp$ symbolizes "emptiness", on the automaton A for the state q. Player *odd* takes over the role of player *reject* and player *even* the role of player *accept*

Game positions:
Each state of the automaton is a position of player *accept* (A):
$V_A = Q$
Each set that satisfies a formula $\delta(q, a)$ is a position of player *reject* (R):
$V_R = \bigcup_{q \in Q, a \in \Sigma} \{Z \mid Z \text{ satisfies } \delta(q, a)\}$

Move relations:
$E_{V_A \times V_R} = \bigcup_{q \in Q, a \in \Sigma} \{(q, Z) \mid Z \text{ satisfies } \delta(q, a)\}$
$E_{V_R \times V_A} = \bigcup_{Z \in V_R} \{(Z, q) \mid (d, q) \in Z\}$

Finally $G^{\perp}_{A}(q) = ((V_{odd}, V_{even}, E, c), q)$ is defined by:
$V_{odd} = V_R$
$V_{even} = V_A$
$E = E_{V_A \times V_R} \cup E_{V_R \times V_A}$
The coloring function is defined as:

$$c(v) = \begin{cases} 0 & \text{if } v = q \text{ and } q \in F \\ 1 & \text{otherwise.} \end{cases}$$

3.2.1 Correctness of the Pseudo Emptiness Game for ABTA

Theorem 4. *Let A be an ABTA and $q \in Q$. The state q is empty if player reject has a memoryless winning strategy in $G_A^{\perp}(q)$.*

Proof. Let g be a memoryless winning strategy for player *reject* and r a run of A. By construction of the game, it exists a g-conform play of $G_A^{\perp}(q)$ that corresponds to a path $r(\pi)$. Strategy g is winning, so there are only finitely many accepting states on $r(\pi)$. Therefore the run r is non-accepting, so q is an empty state. □

In the following, we demonstrate by an example that this simple emptiness test can be a very profitable minimization technique:
Example 1: Let $A = (\{a\}, \{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_1, q_2\})$ be an NBTA on infinite 2-trees where

$$\begin{aligned}
\delta(q_0, a) &= (0, q_1) \wedge (1, q_2)\\
\delta(q_1, a) &= (0, q_1) \wedge (1, q_2)\\
\delta(q_2, a) &= (0, q_3) \wedge (1, q_0)\\
\delta(q_3, a) &= (0, q_3) \wedge (1, q_1)
\end{aligned}$$

Player *reject* has a winning strategy by choosing $(0, q_3)$ in $G_A^{\perp}(q_3)$. Moreover, player *reject* has also winning strategies in $G_A^{\perp}(q_2)$, $G_A^{\perp}(q_1)$ and $G_A^{\perp}(q_0)$. Thus, by using the simple pseudo emptiness game, the NBTA A can be minimized completely.

3.2.2 Incompleteness of the Pseudo Emptiness Game for ABWA

We show now that there exists an ABWA A and an empty state q, nevertheless player *reject* has no winning strategy in $G_A^{\perp}(q)$.

Intuitively, the problem of the game is that player *accept* is too strong because he is allowed to choose the letters without any restrictions. For that reason he may have a winning strategy that corresponds to an inconsistent input tree.
Example 2: Let $A = (\{a, b\}, \{q_0, q_1, q_2\}, q_0, \delta, \{\})$ be an ABWA with

$$\begin{aligned}
\delta(q_0, a) &= (0, q_1) \wedge (0, q_2)\\
\delta(q_0, b) &= \mathit{false}\\
\delta(q_1, a) &= \mathit{true}\\
\delta(q_1, b) &= \mathit{false}\\
\delta(q_2, a) &= \mathit{false}\\
\delta(q_2, b) &= \mathit{true}
\end{aligned}$$

All input words starting with letter b are not accepted. For an input word starting with letter a, the run tree has one accepting path and one non-accepting path, thus it is non-accepting. Finally, $\mathcal{L}(A)$ is empty. However player *accept* has a winning strategy in $G_A^{\perp}(q_0)$:

At first, he chooses the satisfying set $Z = \{(0, q_1), (0, q_2)\}$. Then player *reject* either chooses q_1 or q_2. In both cases, player *accept* chooses the *true* formula, in the next round. Consequently, he always wins the game.

Note that the corresponding input word is inconsistent, because player *accept* chooses once letter "a" and the other time letter "b" for the same position.

3.2.3 Completeness of the Pseudo Emptiness Game for NBTA

For NBTA, there exists a one to one correspondence between the input tree and the run tree. For that reason, it is not possible that a winning strategy of player *accept* corresponds to an inconsistent input tree.

Theorem 5. *Let A be an NBTA and $q \in Q$. Player reject has a memoryless winning strategy in $G_A^{\perp}(q)$ if the state q is empty.*

Proof. (by Contradiction)
Since q is empty, all runs of A^q are non accepting. Assume that player *reject* does not have a winning strategy in $G_A^{\perp}(q)$. Since parity games are determined (Theorem 1), player *accept* has a winning strategy g in $G_A^{\perp}(q)$. We show that we can construct an accepting run (r', T) of A^q by successively using g on q:

- Set the root node: $r'(\epsilon) = (\epsilon, q)$.
- Let $l \in \mathbb{N}$. Assume r' is already constructed using g for all nodes in T_l. Furthermore the corresponding input tree v is defined for all nodes in T_{l-1}.

  (i) Let x be a leaf of T_l and $(n, q') = r'(x)$. We use the strategy g, so let $Z = g(q')$. By definition of the game, Z satisfies $\delta(q', a)$ for some $a \in \Sigma$. We set $v(n) = a$ and extend r' by Z according the definition of ABTA.

  Since there exist a one to one correspondence between the nodes of the run tree and the input tree, the construction step (i) for r' and v is possible for all leaves of T_l and T_{l-1}, respectively, without inconsistencies on the input tree v. Thus, the run r' on the input tree v is constructed for all nodes in T_{l+1} and T_l, respectively.

By induction follows that the infinite construction of r' is a run of A on input v.

Lastly, we show that the run r' is accepting: By construction of r' each labeled path $r'(\pi)$ corresponds to a g-conform play. Because g is winning, all g-conform plays are winning, thus r' is accepting. That contradicts to the premise, that q is an empty state ! □

3.3 Pseudo Universality Game

In spite of the duality between emptiness and universality, the pseudo games are the same except that the players, who choose letters, are exchanged. Finally, the pseudo universal game is a modified acceptance game, where, in each round player *reject* additionally chooses a letter of the alphabet.

We show that this game is correct for ABTA, that is, if player *accept* has a memoryless winning strategy in the pseudo universality game on an ABTA A^q, the state q is universal. We give an example that demonstrates the incompleteness of the game for NBWA.

Definition: We formalize the universality game as an initialized min-parity game $G_A^\top = ((V_{odd}, V_{even}, E, c), q)$. player *odd* takes over the role of player *reject* and player *even* the role of player *accept*. Each game position corresponds to a state of a round. The game graph is constructed as follows:

Game positions:
Each state of the automaton is a position of player *reject*:
$V_{R_1} = Q$
Each pair of a state and a letter is a position of player *accept*:
$V_{A_1} = \{(q, a) \mid q \in Q, a \in \Sigma\}$
Each set that satisfies a formula $\delta(q, a)$ is a position of player *reject*:
$V_{R_2} = \bigcup_{q \in Q, a \in \Sigma} \{Z \mid Z \text{ satisfies } \delta(q, a)\}$

Move relations:
$E_{V_{R_1} \times V_{A_1}} = \{(q, (q, a)) \mid q \in Q, a \in \Sigma\}$
$E_{V_{A_1} \times V_{R_2}} = \bigcup_{q \in Q, a \in \Sigma} \{((q, a), Z) \mid Z \text{ satisfies } \delta(q, a)\}$
$E_{V_{R_2} \times V_{R_1}} = \bigcup_{Z \in V_{A_1}} \{(Z, q) \mid (d, q) \text{ is element of } Z\ \}$

Finally $G_A^\top(q) = ((V_{odd}, V_{even}, E, c), q)$ is defined by:
$V_{odd} = V_{R_1} \cup V_{R_2}$
$V_{even} = V_{A_1}$
$E = E_{V_{R_1} \times V_{A_1}} \cup E_{V_{A_1} \times V_{R_2}} \cup E_{V_{R_2} \times V_{R_1}}$
The coloring function is defined as:

$$c(v) = \begin{cases} 0 & \text{if } v = q \text{ and } q \in F \\ 1 & \text{otherwise.} \end{cases}$$

3.3.1 Correctness of the Pseudo Universality Game for ABTA

Theorem 6. *Let A be an ABTA and $q \in Q$. The state q is universal if player accept has a winning strategy in $G_A^\top(q)$.*

Proof.

Let g be a winning strategy of player *accept* and v an input tree. We construct an accepting run (r, T) of A^q on input v by successively using g:

Set $r(\epsilon) = (\epsilon, q)$.

Let $l \in \mathbb{N}$. Assume that for all nodes of T_l, the run r is constructed using g.

(i) Then let x be a leaf of T_l and $(n, q') = r(x)$. By using strategy g, we get a satisfying set $Z = g(\delta(q', v(n)))$. By construction of the game, Z satisfies

$\delta(q', v(n))$, thus we can extend r by Z according the definition of ABTA.

The construction step (i) can be done for all leaves of T_l. By induction, r is a run on input v.

By construction, a path $r(\pi)$ corresponds to a g-conform play Π of $G_A^\top(q)$. Since g is a winning strategy, the play Π is winning, thus there are infinitely many accepting states on $r(\pi)$. Hence, the run r is accepting. $\square$

3.3.2 Incompleteness of the Pseudo Universality Game for NBWA

We show now that there exists an NBWA A and an universal state q, nevertheless player *accept* has no memoryless winning strategy in $G_A^\top(q)$.

Intuitively, the problem of the game is that player *accept* is too weak because he is restricted to one strategy for all input trees. Hence, if at some game position of player *accept*, the acceptance of an input tree v_1 requires the choice of the satisfying set Z and the acceptance of an input tree v_2 requires the choice of transition Z', player *accept* cannot have a winning strategy although the state could be universal. In more detail, consider the following automaton, which is the dual of the automaton shown in Example 2:

Example 3: Let $U = (\{a,b\}, \{q_0, q_1, q_2\}, q_0, \delta, \{\})$ be an NBWA with

$$\begin{aligned}\delta(q_0, a) &= (0, q_1) \vee (0, q_2)\\ \delta(q_0, b) &= \mathit{true}\\ \delta(q_1, a) &= \mathit{false}\\ \delta(q_1, b) &= \mathit{true}\\ \delta(q_2, a) &= \mathit{true}\\ \delta(q_2, b) &= \mathit{false}\end{aligned}$$

All input words starting with letter b are accepted. For an input word starting with letter a, there exists either an accepting run starting with q_1 or an accepting run starting with q_2. Hence q_0 is universal. However player *reject* has a winning strategy for $G_A^\top(q_0)$:

First, he chooses the formula $\delta(q_0, a)$. Then player *accept* either chooses $Z = \{(0, q_1), (1, q_1)\}$ or $Z' = \{(0, q_2), (1, q_2)\}$. In both cases, player *reject* chooses the *false* formula in the next round. Hence he always wins the game.

3.4 Solving the Pseudo Emptiness and Pseudo Universality Game

In the following, we estimate the size of the game graphs depending on the size of the ABTA and then we give time complexities for solving the games.

Let $A = (\Sigma, Q, q_I, \delta, F)$ be an ABTA on infinite k-trees, n the number of states, l the number of letters and m the total sum of minimal satisfying sets of all the formulas $\delta(q, a)$. Formally speaking: $m = \Sigma_{q \in Q, a \in \Sigma} |mss(\delta(q, a))|$.

Size of the pseudo emptiness game graph:
Positions:

- $|V_A|$ is in $O(n)$.
- $|V_R|$ is in $O(m)$.

Since each state has at least one formula, it follows that $n \leq m$, hence $|V|$ is in $O(m)$.

Edges:

- A position of V_A has $|\bigcup_{a\in\Sigma} mss(\delta(q,a))|$ edges to V_R. Hence, the edge set $E_{V_A\times V_R}$ has $\Sigma_{q\in Q}|\bigcup_{a\in\Sigma} mss(\delta(q,a))|$ edges that is in $O(m)$.
- A position of V_R has at most n edges to V_A. So, $|E_{V_R\times V_A}|$ is in $O(nm)$.

Totally, $|E|$ is in $O(nm)$.

The pseudo emptiness game is a 2-color parity game. Thus, theorem 2 entails that it can be solved in $O(nm^2)$.

Size of the pseudo universality game graph:
Positions:

- $|V_{R_1}|$ is in $O(n)$.
- $|V_{A_1}|$ is in $O(nl)$.
- $|V_{R_2}|$ is in $O(m)$

Since each state has for each letter of the alphabet a formula, it follows that $nl \leq m$, hence $|V|$ is in $O(m)$.

Edges:

- $|E_{V_{R_1}\times V_{A_1}}|$ is in $O(nl)$
- A position of V_{A_1} has $|mss(\delta(q,a))|$ edges to V_{R_2}. Consequently, $|E_{V_{A_1}\times V_{R_2}}|$ is in $O(m)$.
- A position of V_{R_2} has at most n edges to V_{A_1}. Hence $|E_{V_{R_2}\times V_{R_1}}|$ is in $O(nm)$.

As already mentioned above, $nl \leq m$, hence $|E|$ is in $O(nm)$

The pseudo universality game is a 2-color parity game. Thus, theorem 2 entails that it can be solved in $O(nm^2)$.

4 Minimization of Büchi Tree Automata by Simulation-Based Quotienting

In the following, we transfer what has be done for NBWA [11] and ABWA [12] to Büchi tree automata. We introduce and study simulation relations for ABTA (which includes the NBTA). Technically, we define the simulation relations on simulation games that we formalize in terms of min-parity games as in [14, 1, 11]. We concentrate on the three types of simulation relations, namely direct, delayed and fair simulation. The approach for all types is as follows:

- We introduce a simulation game for ABTA such that a winning position corresponds to the fact that some state is simulated by another state.
- Based on the game, we define a simulation relation, which formalizes that a state is simulated by another state.
- We show that the simulation relation guarantees language containment.
- Based on the simulation relation, we define a simulation equivalence relation which formalizes that two states simulate each other.
- We define a quotient automaton with respect to the simulation equivalence relation.
- We show whether the quotient automaton preserves the language of the original automaton
- We demonstrate that computing the simulation relation can be done fast and efficiently.

4.1 Direct, Delayed, and Fair Simulation Game

In the following, we give an intuitive description of the three simulation games and then a formal definition as parity games.

4.1.1 Description of the Simulation Games

In principle, the three simulation games rely on the same game, only the winning conditions varies.

Basic Game: Let A be an ABTA and q_0, q_0' be states. A simulation game is played by the two players Spoiler and Duplicator, in rounds.

The basic principle of the game is as follows: Spoiler dictates letters and visits of accepting states from q_0, which Duplicator must match from q_0' in order to win the simulation game on (q_0, q_0').

The game starts, at round 0, in position (q_0, q_0') but assume the game is in round i, in position (q_i, q_i'). A round is played as follows:

1. Spoiler chooses a letter $a \in \Sigma$, which determines the formula $\delta(q_i, a)$.

2. Spoiler chooses a set Z_S that satisfies $\delta(q_i, a)$.

3. Duplicator, responding, chooses a set Z_D that satisfies $\delta(q'_i, a)$. Note that he has to use the same letter as Spoiler.

4. Spoiler chooses an element (d, q'_{i+1}) of Z_D.

5. Duplicator, responding, chooses an element (d, q_{i+1}) of Z_S. Note that he has to use the same direction d as Spoiler.

which determines the starting pair for the next round: (q_{i+1}, q'_{i+1}).

Intuitively, Spoiler controls the choice of letters and the existential choice of A^{q_0} and Duplicator controls the existential choice of $A^{q'_0}$. For the universal choices the controls are switched. So Spoiler controls the universal choice of $A^{q'_0}$ and Duplicator controls the universal choice of A^{q_0}.

Winning condition: If some player cannot proceed, he looses immediately. If both players can always proceed, they construct an infinite sequence of pairs of states: $(q_0, q'_0)(q_1, q'_1)(q_2, q'_2)\dots$
This sequence determines the winner, depending on which simulation we are interested in:

- Direct (di): Duplicator wins if for every i with $q_i \in F$ we have $q'_i \in F$.
- Delayed (de): Duplicator wins if for every i with $q_i \in F$ there exists $j \geq i$ such that $q'_j \in F$.
- Fair (f): Duplicator wins if it holds: if there are infinitely many i with $q_i \in F$, then there are also infinitely many j with $q'_j \in F$.

In all other cases Spoiler wins.

The winning conditions in the order fair, delayed, direct get stricter for Duplicator. That means, if Duplicator wins a fair simulation game on (q, q'), he might loose the delayed simulation game on (q, q').

4.1.2 Definition of the Simulation Games

We formalize the three simulation games as initialized min parity games

$$G^\star_A(q_0, q'_0) = ((V^\star_{odd}, V^\star_{even}, E^\star, c^\star), (q_0, q'_0)) \text{ with } \star \in \{\text{di,de,f}\}$$

Player *odd* takes over the role of Spoiler and player *even* takes over the role of Duplicator. We present the game graph for G^f_A at first, then for G^{di}_A and lastly for G^{de}_A.

Fair Simulation Game
Each state of a round corresponds to a game position. The state 1. and 2. of a round can be joined to one game position, since Spoiler chooses consecutively.

Therefore we have positions (q, q') for the starting and end point of a round. Secondly, we have positions of the form (Z, q', a) which represent the fact that Spoiler has chosen a satisfying set of $\delta(q, a)$. Next turn, Duplicator chooses a satisfying set of $\delta(q', a)$. Hence, we get positions of the form (Z_S, Z_D), which means that both players have chosen their satisfying sets. Now its Spoilers turn for choosing an element of Z_D. Lastly, we have positions of the form $(Z_S, (d, q'))$ which represent the fact that Spoiler has chosen an element. Duplicator has to choose an element from Z_S with the same direction d and the round is finished. In detail, the game graph consists of the following game positions and move relations:

game positions:
$S_1 = \{(q, q') \mid q, q' \in Q\}$

$D_1 = \bigcup_{(q,q') \in S_1} \{(Z, q', a) \mid a \in \Sigma,\ Z \in \delta(q, a)\}$

$S_2 = \bigcup_{(Z,q',a) \in D_1} \{(Z, Z') \mid a \in \Sigma,\ Z' \in \delta(q', a)\}$

$D_2 = \bigcup_{(Z,Z') \in S_2} \{(Z, (d, q')) \mid (d, q') \in Z'\}$

move relations:
$E_{S_1 \times D_1} = \{((q, q'), (Z, q', a)) \mid (q, q') \in S_1, Z' \in \delta(q, a)\}$

$E_{D_1 \times S_2} = \{((Z, q', a), (Z, Z')) \mid (Z, q', a) \in D_1, Z' \in \delta(q', a)\}$

$E_{S_2 \times D_2} = \{((Z, Z'), (Z, (d, q'))) \mid (Z, Z') \in S_2, (d, q') \in Z'\}$

$E_{D_2 \times S_1} = \{((Z, (d, q')), (q, q')) \mid (Z, (d, q')) \in D_2, (d, q) \in Z\}$

Finally, the game $G_A^f = (V_{even}^f, V_{odd}^f, E^f, c^f)$ can be defined by:
$V_{odd}^f = S_1 \cup S_2$
$V_{even}^f = D_1 \cup D_2$
$E^f = E_{S_1 \times D_1} \cup E_{D_1 \times S_2} \cup E_{S_2 \times D_2} \cup E_{D_2 \times S_1}$

$$c^f(v) = \begin{cases} 0 & \text{if } v = (q, q') \text{ and } q' \in F \\ 1 & \text{if } v = (q, q'), q \in F, \text{ and } q' \notin F \\ 2 & \text{otherwise.} \end{cases}$$

If the smallest occurring number is 0, then Duplicator sees infinitely many accepting states, thus he wins. If the smallest occurring number is 1, then Spoiler sees infinitely many accepting states, but Duplicator only finitely many, hence Spoiler wins. Lastly if the smallest occurring number is 2, then Spoiler has only seen finitely many accepting states, so Duplicator wins. In that way the fair winning condition is defined in terms of a parity game.

Direct Simulation Game
We now describe how G_A^f can be modified to get G_A^{di}. The game graph of G_A^{di} is constructed as the game graph of G_A^f, except that we leave out the edges from positions of D_2 to positions (q, q') of S_1 where $q \in F$ and $q' \notin F$ in order to take care of the direct winning condition.
The game $G_A^{di} = (V_{even}^{di}, V_{odd}^{di}, E^{di}, c^{di})$ can be defined by

$V_{odd}^{di} = V_{odd}^f$

$V_{even}^{di} = V_{odd}^f$

$E^{di} = E^f \setminus \{(v, (q, q')) \mid v \in D_2, (q, q') \in S_1, q \in F, q' \notin F\}$

and the coloring function is 0 for all positions: $c^{di}(v) = 0$

Delayed Simulation Game
For the delayed game G_A^{de} we have to modify the game graph of G_A^f somewhat more: For each vertex of the game graph there will be two corresponding vertices in G_A^{de}. The extra bit b encodes whether Spoiler has seen an accept state or not, for which Duplicator has not seen one so far. The modified positions and move relations are as follows:
game positions:
$S_1' = \{(b, q, q') \mid q, q' \in Q, b \in \{0, 1\}, (q' \in F \to b = 0), (q \in F, q' \notin F \to b = 1)\}$

The other game positions D_1', S_2', D_2' are constructed as for the fair game graph except that each position contains the bit b:
$D_1' = \bigcup_{(b,q,q') \in S_1'} \{(b, Z, q', a) \mid a \in \Sigma,\ Z \in \delta(q, a)\}$

$S_2' = \bigcup_{(b,Z,q',a) \in D_1'} \{(b, Z, Z') \mid Z' \in \delta(q', a)\}$

$D_2' = \bigcup_{(b,Z,Z') \in S_2'} \{(b, Z, (d, q')) \mid (d, q') \in Z'\}$

move relations:
The edges from S_1' to D_1', D_1' to S_2' and S_2' to D_2' are exactly as for the fair game graph except that the bit b is passed from position to position:
$E_{S_1' \times D_1'} = \{((b, q, q'), (b, Z, q', a)) \mid (b, q, q') \in S_1', Z' \in \delta(q, a)\}$

$E_{D_1' \times S_2'} = \{((b, Z, q', a), (b, Z, Z')) \mid (b, Z, q', a) \in D_1', Z' \in \delta(q', a)\}$

$E_{S_2' \times D_2'} = \{((b, Z, Z'), (b, Z, (d, q'))) \mid (b, Z, Z') \in S_2', (d, q') \in Z'\}$

The edges from D_2' to S_1' update the bit b such that the delayed winning condition is transfered:

$$\begin{aligned}
E_{D_2' \times S_1'} = &\{((0, Z, (d, q')), (1, q, q')) \mid (Z, (d, q')) \in D_2', (d, q) \in Z, q \in F \wedge q' \notin F\} \\
&\cup \{((0, Z, (d, q')), (0, q, q')) \mid (Z, (d, q')) \in D_2', (d, q) \in Z, q \notin F \vee q \in F\} \\
&\cup \{((1, Z, (d, q')), (1, q, q')) \mid (Z, (d, q')) \in D_2', (d, q) \in Z, q' \notin F\} \\
&\cup \{((1, Z, (d, q')), (0, q, q')) \mid (Z, (d, q')) \in D_2', (d, q) \in Z, q' \in F\}
\end{aligned}$$

So the game $G_A^{de} = (V_{even}^{de}, V_{odd}^{de}, E^{de}, c^{de})$ can be defined:

$V_{odd}^{de} = S_1' \cup S_2'$

$V_{even}^{de} = D_1' \cup D_2'$

$E^{de} = E_{S_1' \times D_1'} \cup E_{D_1' \times S_2'} \cup E_{S_2' \times D_2'} \cup E_{D_2' \times S_1'}$

$$c^{de}(v) = \begin{cases} b & \text{if } v = (b, q, q') \\ 1 & \text{otherwise} \end{cases}$$

We assign priority 0 or 1 to the vertices $(b, q, q') \in V_{odd}^{de}$ which signifies if the delayed winning condition is fulfilled or not. In other words, if there is an "‘unmatched"’ final state by Duplicator, (for example Spoiler has seen an final state, but Duplicator hasn't seen one) then the bit is 1. If from some point onwards the bit 1 occurs infinitely often, then the smallest number in the game is 1, so Spoiler wins the game. Otherwise, if Duplicator can match each occurrence of an final state then the smallest occurring number will be a 0, so Duplicator wins the game. In this way the delayed winning condition is defined in terms of a parity game.

4.2 Simulation Relations

We introduce the direct, delayed and fair simulation relations via the simulation games:

Definition: Let A be an ABTA. A state q is **$\star$-simulated** by a state q' if there is a memoryless winning strategy g for Duplicator in $G_A^\star(q, q')$ where $\star \in$ {di,de, or f}. We denote such a relationship by $q \preceq_\star q'$.

Example 4: Let $A = (\{a\}, \{q_0, q_1, q_2\}, q_0, \delta, \{q_2\})$ be an ABTA on infinite 2-tree with

$$\begin{aligned}
\delta(q_0, a) &= (0, q_1) \wedge (0, q_2) \wedge (1, q_2) \vee (0, q_0) \wedge (1, q_0) \\
\delta(q_1, a) &= (0, q_2) \wedge (1, q_2) \\
\delta(q_2, a) &= (0, q_2) \wedge (1, q_2) \\
\delta(q_3, a) &= (0, q_2) \wedge (1, q_1) \wedge (1, q_3)
\end{aligned}$$

Clearly, q_2 $\star$-simulates all other states. Moreover, it holds true that $q_0 \preceq_{de} q_1$: In a play starting in (q_0, q_1), Spoiler chooses a satisfying set Z_S for $\delta(q_0, a)$ and then Duplicator must choose $Z_D = \{(0, q_2), (1, q_2)\}$. Next, Spoiler must choose $(0, q_2)$ or $(1, q_2)$ of Z_D. So, in the next round, Duplicator always starts from state q_2 and since q_2 delayed simulates all states, the described strategy is winning for Duplicator in $G_A^{de}(q_0, q_1)$

The converse $q_1 \preceq_{de} q_0$ also holds true: Spoiler chooses $Z_S = \{(0, q_2), (1, q_2)\}$ for $\delta(q_1, a)$. Duplicator chooses $Z_D = \{(0, q_1), (0, q_2), (1, q_2)\}$ for $\delta(q_0, a)$, because otherwise he will stay in q_0 forever and loose. Next, if Spoiler chooses $(0, q_2)$ or $(1, q_2)$, then in the next round Duplicator starts in q_2 and therefore will win. Otherwise, if Spoiler chooses $(0, q_1)$, then in the next round Duplicator starts in q_1. Nevertheless, clearly $q_2 \preceq_{de} q_1$ holds true, so Duplicator also wins from q_1. Thus the described strategy is winning for Duplicator in $G_A^{de}(q_1, q_0)$.

It holds true that $q_2 \not\preceq_{de} q_3$: Spoiler chooses $Z_S = \{(0, q_2), (1, q_2)\}$ for $\delta(q_2, a)$. Duplicator chooses $Z_D = \{(0, q_2), (1, q_1), (1, q_3)\}$ for $\delta(q_3, a)$. Next, Spoiler chooses $(1, q_3)$ of Z_D, Duplicator must choose $(1, q_2)$ of Z_S. Since q_3 is non-accepting and q_2 is accepting, the described strategy is winning for Spoiler in $G_A^{de}(q_2, q_3)$.

4.2.1 Basic Properties

Lemma 1. *For $\star \in \{di, de, f\}$, the simulation relation $\preceq_\star$ is a preorder on the state set Q of an ABTA A.*

Proof. Reflexivity of $\preceq_\star$ with $\star \in \{di, de, f\}$: Let $q \in Q$. Duplicator has a winning strategy in the game $G_A^\star(q, q)$ by choosing the same satisfying sets and directions as Spoiler.

Transitivity of $\preceq_{di}$: Suppose that $q_0 \preceq_{di} q_0'$ and $q_0' \preceq_{di} q_0''$. By definition Duplicator has winning strategies in $G_A^{di}(q_0, q_0')$ and $G_A^{di}(q_0', q_0'')$, say g and g', respectively. We construct a strategy g'' for Duplicator in $G_A^{di}(q_0, q_0'')$ by g and g':

If $g(Z, q', a) = (Z, Z'')$ and $g'(Z', q'', a) = (Z', Z'')$
    then $g''(Z, q'', a) = (Z, Z'')$
And if $g(Z, (d, p')) = (d, p)$ and $g'(Z', (d, p'')) = (d, p')$
    then $g''(Z, (d, p'')) = (d, p)$

The strategy g'' is winning:
Let $\Pi'' = (q_0, q_0'')w_0''v_0''x_0''(q_1, q_1'')w_1''v_1''x_1'' \ldots$ be a g''-conform play. There exists a g-conform play $\Pi = (q_0, q_0')w_0v_0x_0(q_1, q_1')w_1v_1x_1 \ldots$ and a g'-conform play $\Pi' = (q_0', q_0'')w_0'v_0'x_0'(q_1', q_1'')w_1'v_1'x_1' \ldots$.
Since g is a winning strategy, it holds for all i that if q_i is accepting then q_i' is accepting. Similarly, since g' is also a winning strategy, it holds for all i that if q_i' is accepting then q_i'' is also accepting. Consequently, the play Π'' is a winning play and hence g'' is winning.

Transitivity of $\preceq_{de}$: The proof is also analogous to the direct case except the last step. So g and g' are winning strategies, then Π and Π' are winning plays. This implies that if $q_i \in F$ then there exists a $j \geq i$ such that $q_j' \in F$ and then there exists a $k \geq j$ such that $q''_k \in F$. So Π'' is a winning play, hence the claim holds.

Transitivity of $\preceq_f$: The proof is analogous to the direct case except the last step. So g and g' are winning strategies, then Π and Π' are winning plays. This implies that if there are infinitely many i such that $q_i \in F$ then there are also infinitely many j such that $q'_j \in F$ and then there are also infinitely many k such that $q''_k \in F$. So Π'' is a winning play, hence the claim holds.

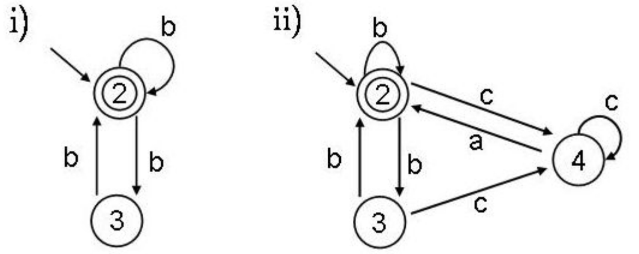
Finally, we have shown that $\preceq_{di}, \preceq_{de}, \preceq_f$ are preorder relations. □

Intuitively, the winning condition from direct to delayed to fair relaxes which leads to a strengthening of Duplicator. Hence if he has a winning strategy in a direct simulation game, then the strategy is also winning in the delayed or fair simulation game.

Lemma 2. *[11] Simulation relations are ordered by containment: $\preceq_{di} \subseteq \preceq_{de} \subseteq \preceq_f$.*

Proof. Let A be an ABTA and let g be a winning strategy of Duplicator in $G^{di}_A(q, q')$. For a g-conform play $\Pi = (q_0, q'_0)w_0v_0x_0(q_1, q'_1)w_1v_1x_1 \ldots$ holds true that for all i: If q_i is accepting, then also q'_i is accepting. The delayed winning condition, (which is for all i holds: If q_i is accepting, then there exists a $j \geq i$ such that q_j is accepting), includes the direct case. So, strategy g is also winning in $G^{de}_A(q, q')$. Similarly, the fair winning condition, (which is, if there are infinitely many i with q_i accepting, then there are also infinitely many j with q'_j accepting), includes the delayed case. Consequently, if g is a winning strategy of Duplicator in $G^{de}_A(q, q')$, then it is also a winning strategy of Duplicator in $G^f_A(q, q')$. □

These inclusions are strict for certain automata:



i): State 2 is not direct simulated by state 3, but state 2 is delayed simulated by 3.
ii): State 2 is not delayed simulated by state 3, since Spoiler has a winning strategy by always choosing c's, but state 2 is fair simulated by state 3.

4.2.2 Simulation Relations guarantee language containment

In the following we prove that all three types of simulation guarantee language containment. At first we show a lemma and then the theorem.

Lemma 3. *For $\star \in \{di, de, f\}$: Let A be an ABTA, $q \preceq_\star q'$ and (r, T) be a run of A^q on input tree v. Then there exist*

- *a run (r', T') of $A^{q'}$ on v and*
- *a mapping $\Psi : T' \to T$ that maps every node $r'(x) = (n, q')$ to a node $r(y) = (n, q)$ such that $q \preceq_\star q'$*

Proof.

Let the domain T be subset of $(N^m)^*$ and the domain T' be subset of $(N^{m'})^*$. Since $q \preceq_\star q'$, Duplicator has a winning strategy g in $G_A^\star(q, q')$. We construct by successively using g a run tree r' of $A^{q'}$ on input v:

Set $r'(\epsilon) = (\epsilon, q')$. Since $q \preceq_\star q'$, set $\Psi(\epsilon) = \epsilon$.

Let $l \in \mathbb{N}$. Assume we have already constructed r' using strategy g for all nodes of T'_l. Moreover, the mapping Ψ exists for all nodes of T'_l.

Let x be a leaf of T'_l and $r'(x) = (n, p')$. By assumption, we get $y = \Psi(x)$ with $(n, p) = r(y)$. Then let Z be the set of the children of the node $r(y)$, formally speaking $Z := \{(d_i, s_i) \mid (n \cdot d_i, s_i) = r(y \cdot i), i \in N^m\}$. By using the strategy g, we get the game position $(Z, Z') := g(Z, p', v(n))$. By definition of the game, Z' is a satisfying set of $\delta(p', v(n))$. We extend $r'(x)$ by Z' according to the definition of a run of an ABTA.

(i) Now let $(n \cdot d, s')$ be a child of $r'(x)$. By using g, we get a position $(s, s') = g(Z, (d, s'))$. By definition of the game, the element (d, s) is in Z. Let $j \in N^m$ such that $r(y \cdot j) = (d, s)$. Since g is winning, $s \preceq s'$ holds true, so we set $\Psi(x \cdot i) = y \cdot j$.

This construction step is done for all leaves of T'_l. So, for all $x \in T'_{l+1}$ r' is constructed and the mapping Ψ exists .

By induction, the infinite construction of r' is a run of $A^{q'}$ on input v and the mapping Ψ exists for all nodes of $T^{'}$. □

Theorem 7. *For $\star \in \{di, de, f\}$: Let A be an ABTA and $q \preceq_\star q'$, then $L(A^q) \subseteq L(A^{q'})$*

Proof. We have to prove that for each input tree with an accepting run on A^q, exists an accepting run on $A^{q'}$. Let $v \in L(A^q)$ and r be an accepting run tree of A^q on input v. Since $q \preceq_\star q'$, Duplicator has a winning strategy g in $G_A^\star(q, q')$. By lemma 3, there exists a run r' of $A^{q'}$ on v and a mapping Ψ. We must show that the run r' is also accepting:

Let $r'(\pi') = (\epsilon, q')(n_1, q'_1)(n_2, q'_2)\ldots$ be a path. By construction of r', there exists a g-conform play $\Pi = (q, q')w_0v_0u_0(q_1, q'_1)\ldots$ of $G_A^\star(q, q')$ consisting of the path $r'(\pi')$ and a path $r(\pi) = (\epsilon, q_0)(n_1, q_1)(n_2, q_2)\ldots$.

The run tree r is accepting, so $r(\pi)$ is accepting. Since the strategy g is winning, Π is winning, thus $r'(\pi')$ is also accepting. Finally, r' is an accepting run.

$\square$

4.3 State Space Minimization by Quotienting

In this section, we introduce the equivalence relation $\approx$ which formalizes that two nodes simulate each other. Based on the equivalence relation, we define a quotient automaton. Then we show that for direct and delayed simulation, language equivalence between the automaton and quotient automaton holds true. For fair simulation there is no reasonable definition of a quotient automaton as shown in Proposition 16 of [11]

Definition: For each simulation relation we define an **equivalence relation** $\approx_{di}, \approx_{de}$, and $\approx_f$, where

$q \approx_\star q'$ if and only if $q \preceq_\star q'$ and $q' \preceq_\star q$.

Definition: For ABTA $A = (\Sigma, Q, q_I, \delta, F)$ and equivalence relation $\approx$ on the states Q, let $[q]$ denote the equivalence class of $q \in Q$ with respect to $\approx$. The **quotient automaton** of A with respect to $\approx$ is the automaton

$$A/\approx \; = (\Sigma, Q/\approx, [q_I], \delta_\approx, F/\approx)$$

where

$$\delta_\approx([q], a) = \bigvee_{q' \in [q]} \Phi(\delta(q', a))$$

and

$$F/\approx \; = \{[q] \mid q \in [q] \text{ and } q \in F\}$$

with function $\Phi : \mathcal{B}^+(D \times Q) \to \mathcal{B}^+(D \times Q/\approx)$, which maps a formula θ consisting of concrete states to the corresponding formula consisting of their quotient states. That means, each (d, q') of θ is mapped to $(d, [q])$ if $q' \in [q]$.

The quotient automaton is an overestimation because it accepts at least all input trees of $\mathcal{L}(A)$. The reasons are that each quotient state $[q]$ has the transitions of all concrete states $q \in [q]$ and a quotient state $[q]$ is accepting if there exist some concrete accepting state $q \in [q]$. Thus the accepting run of A must be also an accepting run of the quotient automaton. The converse direction also holds true, but needs much more work. First, we prove two lemmas and then the main theorem.

Lemma 4. *For $\star \in \{di, de\}$: Let A be an ABTA, q be an accepting state, $q \preceq_\star q'$ and (r,T) be a run of A^q on input tree v. Then there exist*

- *a run (r', T') of $A^{q'}$ on v such that on each path of r' at least one accepting state exists*
- *a mapping $\Psi : T' \to T$ that maps every node $r'(x) = (n, q')$ to a node $r(y) = (n, q)$ such that $q \preceq_\star q'$*

Proof. By lemma 3 we know that there exists a run r' of $A^{q'}$ on v and a mapping Ψ (as defined above). Since $q \preceq_\star q'$, Duplicator has a winning strategy g in $G^\star_A(q,q')$. Let $r'(\pi') = (\epsilon, q')(n_1, q'_1)(n_2, q'_2)\dots$ be a path. By construction of r', there exists a g-conform play $\Pi = (q,q')w_0v_0u_0(q_1,q'_1)\dots$ of $G^\star_A(q,q')$ consisting of the path $r'(\pi')$ and some path $r(\pi) = (\epsilon, q)(n_1, q_1)(n_2, q_2)\dots$. Since g is winning, the play Π is winning. Hence by the definition of the $\star$-simulation game follows that there exists a $i \in \mathbb{N}$ such that q'_i is accepting. □

Lemma 5. *Let $A = (\Sigma, Q, q', \delta, F)$ be an ABTA on k-trees and $A/\approx_{de} = (\Sigma, Q/\approx, [q], \delta_\approx, F/\approx)$ the quotient ABTA. Let $v \in \mathcal{L}(A/\approx_{de})$, $q \preceq_{de} q'$ and (r,T) a run of $A/\approx_{de}$ on v. Then it exist*

- *a run (r', T') of A on input v and*
- *a mapping $\Psi : T' \to T$ that maps every node $r'(x) = (n, q')$ to a node $r(y) = (n, [q])$ such that $q \preceq_\star q'$*

Proof.
Let the domain $T \subseteq (N^m)^*$ and the domain $T' \subseteq (N^{m'})^*$. Let g be Duplicator's winning strategy in $G^{de}_A(q,q')$. We construct r' by successively using g:

We set $r'(\epsilon) = (\epsilon, q')$. Since $q \preceq q'$ holds, we set $\Psi(\epsilon) = \epsilon$.

Let $l \in \mathbb{N}$. Assume the run r' is constructed for all nodes of T'_l (using g). Moreover the mapping Ψ exists for all nodes of T'_l

Let x be a leaf of T'_l with $r'(x) = (n, p')$. By assumption, we get $y = \Psi(x)$ with $(n, [p]) = r(y)$. By definition of Ψ: $p \preceq p'$ holds. Then let Z be the set of children of $r(y)$, formally speaking $Z = \{(d_i, [s_i]) \mid (n \cdot d_i, [s_i]) = r(y \cdot i), i \in N^m\}$.
By definition of ABTA, Z is a satisfying set of $\delta_\approx([p], v(n))$.
By definition of the quotient automaton $A/\approx$, it exists
(i) a state $\hat{p}$ such that $\hat{p} \approx p$
(ii) a satisfying set $\hat{Z}$ of $\delta(\hat{p}, v(n))$ such that
for each $(d, \hat{s}) \in \hat{Z}$, exists a $(d, [s]) \in Z$
such that $\hat{s} \approx s$
By (i) and $p \preceq p'$ and transitivity follows: $\hat{p} \preceq p'$
Let g' be the memoryless winning strategy for Duplicator in $G^{de}_A(\hat{p}, p')$. By using g' we get the game position $(\hat{Z}, Z') = g'(\hat{Z}, p', v(n))$. By definition of the simulation game, Z' satisfies $\delta(p', v(n))$. We extend $r'(x)$ by Z' according the definition of a run of an ABTA.

Moreover, by using g' we get for a $(d, s') \in Z'$ a $(d, \hat{s}) \in \hat{Z}$ such that $\hat{s} \preceq s'$. By (ii) and transitivity follows $s \preceq s'$. Let $i \in N^{m'}$ and $j \in N^m$ such that $r'(xi) = (n, s')$ and $r(yj) = (n, [s])$. Then we set $\Psi(xi) = yi$.

This construction step can be done for all leaves of T_l. Hence for all nodes of T_{l+1}, r' is constructed and the mapping Ψ exists.

By induction, the infinite construction of r' is a run of A on input v and the mapping Ψ exists for all nodes of r'. $\square$

4.3.1 Direct and Delayed Quotienting are language preserving

Theorem 8. *For $\star = \{di, de\}$: Let A be an ABTA and $\approx_\star$ a simulation equivalence relation. Then it holds true that $\mathcal{L}(A) = \mathcal{L}(A/\approx_\star)$*

We prove the claim for delayed simulation, because the claim for direct simulation follows from that.

Proof.
$\mathcal{L}(A) \subseteq \mathcal{L}(A/\approx_{de})$:
Let $v \in \mathcal{L}(A)$ and (r, T) an accepting run of A on v. As already mentioned, the quotient automaton $A/\approx_{de}$ is an overestimation, thus it follows directly that (r', T) with $r'(x) := (n, [q])$ if $r(x) = (n, q)$, is an accepting run of $A/\approx_{de}$ on v.

$\mathcal{L}(A/\approx_{de}) \subseteq \mathcal{L}(A)$:
Let $v \in \mathcal{L}(A/\approx_{de})$. Let r be an accepting run of $A/\approx_{de}$ on v. We cannot guarantee that r' with $r'(x) = (n, q)$ if $r(x) = (n, [q])$, is a run of A on v. However we show that there exists an accepting run $\hat{r}$ of A on input v:

Let $\hat{q_0}$ be the initial state of A. By definition of the quotient automaton $\hat{q_0} \approx_{de} q_0$ and in particular $q_0 \preceq_{de} \hat{q_0}$. By lemma 5 follows that there exists a run $\hat{r}$ of A on input v and a mapping Ψ. Let T be the domain of r, $\hat{T}$ be the domain of $\hat{r}$ and W be the domain of v.

We will successively modify the run $\hat{r}$ such that it guarantees:
(i) If a state $[q]$ of a node $r(x) = (n, [q])$ is accepting,
then $\forall\ y \in \Psi^{-1}(x)\ \forall$ paths $\pi \subseteq \hat{T}^y\ \exists\ z \in \pi$ with $(n, \hat{q}) = \hat{r}(z) : \hat{q} \in F$.
(where $\Psi^{-1}(x)$ is the pre image of x)

Let $l \in \mathbb{N}$. Assume the run $\hat{r}$ is modified such that (i) holds for all nodes of T_l. Let x be a leaf of T_{l+1} and $(n, [q]) = r(x)$.
If $[q] \notin F/\approx_{de}$ then $\hat{r}$ is not modified.
If $[q] \in F/\approx_{de}$, then we consider the states of $\hat{r}$ that simulate q:
Let $S = \Psi^{-1}(x)$.
If S is empty, then we do not modify $\hat{r}$.
If S is non empty, let $(n, \hat{q}) = \hat{r}(y)$ for $y \in S$

If y is affected by a modification of a parent node of y, then we do not modify $\hat{r}$ for y. (This will be clear later on.)
If $\hat{q} \in F$, then no modifications of $\hat{r}$ for y.
If $\hat{q} \notin F$ then we change $\hat{r}$ for the subtree $\hat{T^y}$ as follows:

By definition of the quotient automaton there exists a state $p' \in [q]$ that is an accepting state. Since $q \preceq_{de} \hat{q}$ and $q \approx_{de} p'$, it follows $p' \preceq_{de} \hat{q}$ by transitivity.
Then by lemma 5 follows that there exist a run t of $A^{p'}$ on the input subtree (v, W^n) and a mapping Ψ_t.

So we have a run t with the root node (ϵ, p'), $p' \in F$ and $p' \preceq_{de} \hat{q}$. By Lemma 4 follows that there exists a run t' of $A^{\hat{q}}$ on (v, W^n) such that on each path of t' at least one accepting state exists. Furthermore, there exists a mapping $\Psi_{t'}$.

Finally, we modify $\hat{r}$ for the subtree $\hat{T^y}$:
We set $\hat{r}(y \cdot z) := t'(z)$ for all nodes z of the domain of t'.
Moreover we redefine the mapping Ψ:
$\Psi(x \cdot z) := \Psi_t(\Psi_{t'}(z))$ for all nodes z of the domain of t'. We exclude all nodes of a path that starts at $\hat{r}(y)$ and ends at the first accepting state, for further modifications.

The subtree substitution is done for all nodes of S. The modification is done for all leaves of T_{l+1}. Consequently, the property (i) holds for all nodes of T_{l+1}.

By induction, the infinite modified construction of $\hat{r}$ is an accepting run of A on input v. □

4.4 Solving Simulation Games

In the following, we compute the size of the different game graphs in order to give time complexities for solving the simulation games.

Let $A = (\Sigma, Q, q_i, \delta, F)$ be an ABTA on infinite k-trees, n the number of states, l the number of letters and m the total number of minimal satisfying sets of all the formulas $\delta(q, a)$. That means precisely: $m = \Sigma_{q \in Q, a \in \Sigma} |mss(\delta(q, a))|$.

For $\star \in \{di, de, f\}$ we show that the game graph of the simulation game $G_A^\star$ has $O(m(m + k \cdot n))$ vertices and $O(k \cdot n \cdot m^2)$ edges. At first we estimate the number of vertices of the game graph:

- The set S_1 contains at most n^2 positions.
- The set D_1 has a position for each pair of a state and a satisfying set, so $n \cdot m$ positions.

- The set S_2 contains a position for every pair of satisfying sets with the same letter, hence $|S_2| \leq m^2$.
- Lastly, the set D_2 has for each pair of satisfying set and an element of $N_k \times Q$ a position, so at most $m \cdot k \cdot n$ positions.

By definition of an ABTA, each state has at least one formula, hence $n < m$ holds true, hence $|V^*|$ is in $O(m(m+kn))$. Remark, for the delayed case, because of the bit b, the number of vertices is larger by at most a factor of 2.

Secondly, we estimate the number of edges of the game graph:

- S_1 has n^2 positions. A position of S_1 has less than m edges, thus $|E_{S_1 \times D_1}|$ is in $O(n^2 \cdot m)$.
- D_1 has $n \cdot m$ positions. A position of D_1 has less than m edges, thus $|E_{D_1 \times S_2}|$ is in $O(n \cdot m^2)$.
- S_2 has at most m^2 positions. A position of S_2 can have at most $k \cdot n$ edges, hence $|E_{S_2 \times D_2}|$ is in $O(m^2 \cdot k \cdot n)$
- D_2 has at most $m \cdot k \cdot n$ positions. A position of D_2 can have $k \cdot n$ edges, hence $|E_{D_2 \times S_1}|$ is in $O(m \cdot k^2 \cdot n^2)$

Totally $|E^f|$ is in $O(mkn(m + kn))$ and for the direct case, it's no difference because E^{di} is a subset of E^f. As already mentioned above, because of the bit b, also $|E^{de}|$ is larger by at most a factor of 2.

We consider the time complexities for computing the different simulation relations.

The direct simulation game is an 1-color parity game that can be directly translated to a simple reachability game. A reachability game is solvable in linear time in the number of vertices and edges, thus the direct simulation relation is computed in time $O(mkn(m + kn))$

The delayed simulation game is a 2-color parity game that is solvable in time $O(|V||E|)$ (Theorem 2). Consequently, the delayed simulation relation is computed in time $O(m^2kn(m+kn)^2)$.

The fair simulation game is a 3-color parity game that is also solvable in time $O(|V||E|)$ (Theorem 2). Thus the fair simulation relation is also computed in time $O(m^2kn(m+kn)^2)$.

5 Evaluation of the Minimization Techniques

This chapter describes our prototypical implementation, experiments and the experimental results.

5.1 Prototypical Implementation

We briefly describe the main components of the prototype:

ABTA. We implemented ABTA on infinite 2-trees where all formulas of the transition function δ are in disjunctive normal form. Furthermore an ABTA has two distinct states q_t and q_f that are equivalent to *true* and *false*, respectively.

Generator of CTL-Formulas. It generates CTL-formulas with a certain number of operators. The operators are chosen according to a probability distribution and the atomic propositions are chosen with equal probability. More precisely a CTL formula is generated as follows::

```
Formula createFormula(numOp) {
if(numOp == 0) {
   choose atomic proposition AP
   return new AP;
}
else {
     choose Operator op w.r.t. to distribution
     numOp--
     if(op unary)
         return new Op(createFormula(numOp))
     if(op is binary) {
         choose a number n between 0 and numop
         return new Op(createFormula(n),  createFormula(numOp-n))
         }
     }
}
```

CTL to ABTA Translator. A CTL-formula is translated to an ABTA (on 2-trees) as described in [4, 6]. Additionally, the formulas of the transition function of the ABTA are put in disjunctive normal form.

Game Graph Construction. The constructions of the game graphs are implemented as described in section 4.1.2.

Game Solver. First, we implemented the Jurdziński algorithm for solving parity games (with at most 2 colors) [16]. For our experiments the algorithm performs badly. Thus, we also implemented the classical fixpoint algorithm for Büchi games [19], which performs much better.

ABTA Reducer. For a given set of states that are empty, the ABTA Reducer removes the states and all transitions to these states. For a given set of states that are universal, the ABTA Reducer removes the states and replaces all transitions to these states by a transition to the state q_t.

ABTA Quotient. For a given simulation relation, it constructs the quotient automaton as defined in section 4.3.

MH-Construction. We implemented the MH-construction for tree automata (see section 2.3). Our construction algorithm firstly creates the initial state of the NBTA. Then, starting at the initial state, it successively adds new reachable states and transitions to the NBTA. Thus, the constructed NBTA contains only necessary states.

5.2 Experiments

Since there exists neither previous experiments on minimization of tree automata nor any datasets of tree automata that we could use for benchmarks, we decided to generate random CTL-formulas and translate them to ABTA as described in the previous section.

We consider two types of experiments for ABTA:

The first type of experiment tests how much the minimization techniques reduce ABTA. We apply the techniques in the following order on an ABTA A_0:

- Elimination of empty states (A_1)
- Elimination of universal states (A_2)
- Direct Simulation-Based Quotienting (A_3)
- Delayed Simulation-Based Quotienting (A_4)

The idea of the described order is that we try to minimize the automaton as much as possible using the cheap techniques until we apply the costly techniques.

The second type of experiment tests how much the minimization techniques reduce blowups of ATBA caused by the MH-construction. The experiment is as follows:

- An ABTA A_0 is minimized as explained in the previous experiment (A_4).
- We translate the minimized ABTA A_4 to a NBTA N_0 by MH-construction.
- The NBTA N_0 is minimized as explained in the previous experiment (N_4).
- We also translate the original ABTA A_0 to a NBTA N_0' by the MH-construction and compare the sizes of N_0 and N_0' as well as N_4 and N_0'.

The reduction of the ABTA is relatively cheap compared to the reduction of the NBTA, because the MH-construction can incur an exponential blowup.

5.3 Results

For the first type of experiment, we consider the average difference of the state space size of the original ABTA and the full minimized ABTA. Moreover, we compute the average difference of state space size of ABTA after a single minimization technique. In detail, we compute for each automaton:

Reduction by total ABTA Minimization:	$\frac{\lvert A_0\rvert - \lvert A_4\rvert}{\lvert A_0\rvert}$
Reduction by ABTA Empty States Elim.:	$\frac{\lvert A_0\rvert - \lvert A_1\rvert}{\lvert A_0\rvert}$
Reduction by ABTA Universal States Elim.:	$\frac{\lvert A_1\rvert - \lvert A_2\rvert}{\lvert A_1\rvert}$
Reduction by ABTA Direct Quotienting:	$\frac{\lvert A_2\rvert - \lvert A_3\rvert}{\lvert A_2\rvert}$
Reduction by ABTA Delayed Quotienting:	$\frac{\lvert A_3\rvert - \lvert A_4\rvert}{\lvert A_3\rvert}$

For each category, we compute an average value. If an ABTA is reduced to the empty automaton by the Empty States Elimination, we do not consider them for the computation of the average values of the further minimization techniques. Similarly, if an automaton is reduced to the universal automaton by the Universal States Elimination.

For the second type of experiment, we additionally compute the state space differences of the minimized MH-NBTA and the non-minimized MH-NBTA:

Red. by total NBTA Min.:	$\frac{\lvert N_0\rvert - \lvert N_4\rvert}{\lvert N_0\rvert}$
Red. by total ABTA Min. + MH.:	$\frac{\lvert N_0'\rvert - \lvert N_0\rvert}{N_0'}$
Red. by total ABTA Min. + MH + Red. by total NBTA Min.:	$\frac{\lvert N_0'\rvert - \lvert N_4\rvert}{\lvert N_0'\rvert}$

5.3.1 Experiments of Type 1

We run the experiment of type 1 on six types of ABTA or rather CTL-Formulas. The number of different atomic propositions in a generated CTL-formula is always 2. The number of operators in a CTL-formula is either 10, 20 or 30. We consider two probability distributions of the operators in the CTL-Formulas. The first is "with all operators", whereas the second is "without AG, EG, AF and EF".

The first distribution P_1 is:

Neg: 20 %,	AND: 20%,	OR: 20%,	
EX: 5 %,	AX: 5 %,	EU: 5 %,	AU: 5 %
EG: 5 %,	AG: 5 %,	EF: 5 %,	AF: 5 %

	Exp. 1 (10 Op.)		Exp. 2 (20 Op.)		Exp. 3 (30 Op.)	
ABTA Min.:	50 %	(1000)	60 %	(1000)	70 %	(1000)
Empty S. E.:	36 %	(1000)	42 %	(1000)	52 %	(1000)
Universal S. E.:	15 %	(736)	24 %	(735)	29 %	(658)
Direct Quot.:	7 %	(600)	7 %	(554)	8 %	(486)
Delayed Quot.:	2 %	(600)	2 %	(554)	2 %	(486)

(The numbers in brackets are the numbers of ABTA on which the technique has been applied)

Experiment 1 shows a total reduction of an ABTA of 50 % in average and by increasing the number of operators for a formula, the total reductions increase as well. (see Experiment 2 and 3)

Most of the reduction is achieved by the elimination of trivial states. For example, in experiment 3, the elimination of empty states reduces 1000 ABTA by 52 % in average whereas 342 automata are minimized to the empty automaton. Furthermore, the elimination of universal states reduces 658 ABTA by about 29 % in average whereas 172 automata are minimized to an universal automaton.

Now, we consider a second distribution P_2:

Neg: 20 %,	AND: 20 %,	OR: 20 %,	
EX: 10 %,	AX: 10 %,	EU: 10 %,	AU: 10 %
EG: 0 %,	AG: 0 %,	EF: 0 %,	AF: 0 %

	Exp. 4 (10 Op.)		Exp. 5 (20 Op.)		Exp. 6 (30 Op.)	
ABTA Min.:	8 %	(1000)	13 %	(1000)	16 %	(1000)
Empty S. E.:	3 %	(1000)	5 %	(1000)	6 %	(1000)
Universal S. E.:	2 %	(979)	3 %	(967)	4 %	(964)
Direct Quot.:	3 %	(937)	5 %	(912)	6 %	(915)
Delayed Quot.:	1 %	(937)	1 %	(912)	1 %	(915)

Without the operators AG, EG, AF, EF, the total reductions of the ABTA is at most 16 %. The reason for these relatively sparse reductions is that the probability of unsatisfiable subformulas and tautologic subformulas in a CTL-formula with AG, EG, AF, EF is much higher than without these operators. As a result, the ABTA of experiment 1-3 have much more empty and universal states than the ABTA of experiment 4-6.

5.3.2 Experiments of Type 2

For these experiments, we reduce the number of operators in a CTL-formula to 15 in order to avoid a time blowup. The number of different atomic propositions in a CTL-formula is again 2. We use the same distributions of operators, namely P_1 and P_2, as in the experiments of type 1.

	Exp. 7 (15 Op., P_1)		Exp. 8 (15 Op., P_2)	
ABTA Min.:	55 %	(1000)	10 %	(1000)
NBTA Min.:	23 %	(551)	33 %	(917)
ABTA Min. + MH.:	60 %	(1000)	10 %	(1000)
ABTA Min. + MH. + NBTA Min.:	70 %	(1000)	39 %	(1000)
ABTA Empty States Elim.:	39 %	(1000)	4 %	(1000)
ABTA Universal States Elim.:	21 %	(715)	3 %	(972)
ABTA Direct Quot.:	6 %	(551)	4 %	(917)
ABTA Delayed Quot.:	2 %	(551)	1 %	(917)
NBTA Empty States Elim.:	4 %	(551)	6 %	(917)
NBTA Universal States Elim.:	0 %	(537)	0 %	(875)
NBTA Direct Quot.:	18 %	(537)	26%	(875)
NBTA Delayed Quot.:	4 %	(537)	7 %	(875)

As in experiment 1, the ABTA of experiment 7 are reduced by about 55 % whereas 449 are reduced to the empty or universal automaton. Hence there are also high average size difference between the MH-NBTA N_0 and the MH-NBTA N_0' of 60 %. Furthermore, experiment 7 shows that the minimization of the 551 MH-NBTA leads to a reduction of 23 % in average whereas most of the reduction is achieved by direct simulation-based quotienting.

All in all, the blowup of the MH-construction is reduced to 70 % due to minimization. The simple elimination of trivial states on ABTA and the direct simulation-based quotienting on the MH-NBTA contribute most of the reduction gain.

Experiment 8 shows almost no reductions due to the minimization techniques for ABTA, but the 917 MH-NBTA are reduced by 33 % in average. As in experiment 7, the direct simulation-based quotienting contributes most of the reduction gain (26 %). Totally, for these ABTA, the blowup of the MH-construction is reduced by 39 %.

6 Summary and Conclusions

In Chapter 3, we presented simple techniques for removing trivial states of an ABTA. The detection of the trivial states is based on parity games, which are modifications of the acceptance game for Büchi tree automata. We showed that these games are correct and efficiently solvable in $O(nm^2)$, but incomplete for ABTA.

In Chapter 4, we have adapted direct, delayed and fair simulation relations to alternating Büchi tree automata and studied how useful they are for minimization. As a result, simulation relations are even for tree automata an appropriate method for checking language containment and state space reductions. We showed that direct and delayed simulation-based quotienting on ABTA are language preserving, as in the case of Büchi word automata. Also, we analyzed the complexities for computing the simulation relations and we can state that they can be computed efficiently, namely in $O(mkn(m + kn))$, $O(m^2kn(m + kn)^2)$ and $O(m^2kn(m + kn)^2)$ for direct, delayed and fair simulation, respectively.

We have implemented a prototypical system for our methods. Chapter 5 describes this prototype and experiments for benchmarking the minimization techniques. We investigated the average reduction of two types of ABTA and the average reduction of the blowup of ABTA that can occur due to the Miyano-Hayashi construction.

The simulation-based methods for ABTA and the delayed simulation-based quotienting for the MH-NBTA did not achieve much reduction gains. Since MH-NBTA can be relatively huge, the minimization techniques could be expensive. Thus, in particular for delayed simulation-based quotienting, the reduction gain compared to the cost might not pay off. Unlike, the elimination of trivial states of certain ABTA and the direct simulation-based quotienting of MH-NBTA achieved high reductions.

All in all, ABTA of type 1 were reduced by up to 70 % and the blowups of ABTA of type 1 and type 2 were reduced by 70 % and 39 %, respectively. Consequently, we conjecture that our minimization techniques might be of considerable use for the ABTA employed in verification and synthesis processes.

References

[1] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *CONCUR '98: Proceedings of the 9th International Conference on Concurrency Theory*, pages 163–178, London, UK, 1998. Springer-Verlag.

[2] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[3] J. Richard Büchi. On a decision method in restricted second-order arithmetic. In *E. Nagel, P. Suppes, and A. Tarski, editors, Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress*, pages 1–11. Stanford University Press, 1962.

[4] Orna Bernholtz and Moshe Y. Vardi. An automata-theoretic approach to branching-time model checking. In *Journal of the ACM*, pages 142–155. Springer-Verlag, 1994.

[5] J.A. Brzozowski and E. Leiss. Finite automata, and sequential networks. volume 10, pages 19–25, 1980.

[6] Ahmed Saoudi David E. Muller and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *In 3rd IEEE Ann. Symp. on Logic in Computer Science*, pages 422–427, 1988.

[7] David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for language inclusion using simulation preorders. In *Computer Aided Verification*, pages 255–265, 1991.

[8] E. Allen Emerson and Chin-Laung Lei. Temporal model checking under generalized fairness constraints. In *18th Hawaii International Conference on System Sciences*, pages 277–288, 1985.

[9] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987.

[10] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Proceedings of 11th Int. Conf. on Concurrency Theory (CONCUR)*, 2000.

[11] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for buechi automata. In *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, pages 694–707, London, UK, 2001. Springer-Verlag.

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

[13] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 60–65, New York, NY, USA, 1982. ACM.

[14] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In *In CONCUR 97: Theories of Concurrency, Lecture Notes in Computer Science*, volume 1243, pages 273–287. Springer-Verlag, 1997.

[15] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[16] Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, February 2000. Springer-Verlag.

[17] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[18] Leslie Lamport. "Sometime" is sometimes "not never": on the temporal logic of programs. In *POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 174–185, New York, NY, USA, 1980. ACM.

[19] Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.

[20] Robin Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.

[21] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[22] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1-2):69–107, 1995.

[23] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

[24] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *CAV 2000, LNCS*, volume 1855, pages 247–263. Springer-Verlag, 2000.

[25] Moshe Y. Vardi. Alternating automata and program verification. In *In Computer Science Today. LNCS*, volume 1000, pages 471–485. Springer-Verlag, 1995.

[26] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.