Masaryk University
Faculty of Informatics

Reduction of Omega-Automata Using k-Lookahead Simulations

Bachelor's Thesis

Miriama Sasaráková

Brno, Spring 2018

Masaryk University
Faculty of Informatics

Reduction of Omega-Automata Using k-Lookahead Simulations

Bachelor's Thesis

Miriama Sasaráková

Brno, Spring 2018

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Miriama Sasaráková

Advisor: doc. RNDr. Jan Strejček, Ph.D.
Consultant: RNDr. František Blahoudek

Acknowledgements

First, I would like to thank my advisor Jan Strejček for all his advices, patience and time he spent helping me to write this thesis. I would also like to thank my supervisor Fanda for his advices and for teaching me how to use my computer.

Second, I thank my parents for giving me the opportunity to study and that they always supported and motivated me to work on myself.

Lastly I would like to thank Julko, my friends and everyone who encouraged me during writing this thesis.

Abstract

This thesis presents extended definitions of forward, backward and *k*-lookahead simulation relations for transition-based generalized Büchi automata (TGBA).

Then, it provides an implementation outline of direct-forward and backward simulation relations by utilizing parity games. Selected simulations were used in algorithms (quotienting and transition pruning) to reduce the state space of the automata.

Lastly, implemented simulations and algorithms are experimentally evaluated against simulations in SPOT.

Keywords

TGBA, Spot, LTL3BA, parity game, simulation relations, quotienting, pruning

Contents

1 Introduction

One of the basic characteristics of the era we are living in is using computers on daily basis. During any system development, it is important to apply verification techniques in order to check whether the system possesses desired properties (e.g. system should never get into a deadlock scenario). Those properties are obtained from system's specification which says how the system should and should not act in different scenarios (often protecting the system from reaching some critical state).

We are interested in instruments that enable us to analyze and verify systems. Model-checking [1] is an example of automatic verification technique. Many such techniques use finite state automata to accurately describe desired or undesired behaviors of the system. States of the automata cover, e.g., values of variables, and transitions between states describe how the system can possibly evolve from one state into another. Considering specification and verification of reactive systems (i.e., systems which are not expected to terminate at any point of time), we use automata on infinite words (ω-automata). Examples of such systems are operating systems, hardware, control systems and some network protocols. For the purposes of model-checking, it is important to have smaller automata, since the algorithm is based on a computation of a product of two automata. Therefore it is important to be concerned with automata reduction techniques.

An example of such a technique is a simulation between states of the automaton (i.e., we are looking for states which are equivalent with respect to some simulation). This thesis is based on a paper written by Lorenzo Clemente and Richard Mayr [2]. In this paper, there are presented numerous efficient algorithms used for reducing both numbers of states and transitions of NBA (Nondeterministic Büchi automata, a variation of finite automata running over infinite inputs) while preserving their languages. These algorithms combine removing dead states, quotienting and transition pruning.

Dead states are states not reachable from an initial state and those from which no accepting loop can be reached. Quotienting is an operation, where multiple states are merged into a single one according to some equivalence. Transition pruning is a technique used for removing transitions while the language of the automaton is preserved. That can be done due to the existence of some "better" transitions which are present in the automaton. However, computing simulations which can be used in aforementioned algorithms (especially those resulting in greatest reductions) present a computationally hard problem. This is the reason why we need to look for simulation relations which are feasible in practice but still lead to notable reductions.

For this purpose, in [2] were introduced $k-$lookahead simulations for NBA as an efficient and practical reduction technique. In this thesis we will extend traditional simulation relations together with their k-lookahead variants to work with different type of $\omega-$automata, namely with TGBA (Transition-based generalized Büchi automata).

1.1 Structure of the thesis

This thesis is divided into multiple parts as follows. Chapter two provides some basic definitions used later on. The third chapter recalls variety of simulation relations and language-preserving reduction algorithms on NBA using such relations. In the fourth chapter, definitions of simulations are extended to TGBA. The fifth chapter specifies how selected simulations are computed. The sixth chapter discusses the implementation of suggested simulations and reduction algorithms and presents experimental evaluation of our reduction tools QUOT and PRUNE against those implemented in Spot. The last chapter provides conclusion to this thesis.

2 Preliminaries

Definition 2.1. A nondeterministic *Büchi automaton* (NBA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$, where:

- Q is a finite set of *states*,
- Σ is a finite set of *symbols* representing the *alphabet* of $\mathcal{A}$,
- $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic *transition relation*,
- $q_I \in Q$ is an *initial state*, and
- $F \subseteq Q$ is a set of *accepting states*.

We usually write $p \xrightarrow{\sigma} q$ instead of $(p, \sigma, q) \in \delta$. NBA can be imagined as a directed graph with states representing nodes of the graph and transitions between states $p \xrightarrow{\sigma} q$ representing edges between nodes.

A *trace* of $\mathcal{A}$ on word $w = \sigma_0\sigma_1\sigma_2... \in \Sigma^\omega$ is an infinite sequence of transitions $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ starting in $q_0 \in Q$ and satisfying $q_i \xrightarrow{\sigma_i} q_{i+1} \in \delta$ for each $i \geq 0$. A *run* is a trace that starts in an initial state q_I.

We call a run *accepting* if some *accepting state* occurs in the run *infinitely often*. A word w is accepted by $\mathcal{A}$ only if there exists an *accepting run* of $\mathcal{A}$ on w. A set of words, which are accepted by $\mathcal{A}$, represent a *language* $\mathcal{L}(\mathcal{A})$ of automaton.

Definition 2.2. A *Transition-based generalized Büchi automaton (TGBA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$, where

- Q is a finite set of *states*,
- Σ is a finite set of *symbols* representing the *alphabet* of $\mathcal{A}$,
- $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic *transition relation*,
- $q_I \in Q$ is an *initial state*, and
- $F = \{F_1, F_2, ..., F_n\}$ and $F_i \subseteq \delta$ are sets of accepting transitions.

In figures, we indicate membership of transitions to accepting sets by using one accepting mark (e.g. ⓿, ❶, ❷...) per set.

A *run* of an automaton $\mathcal{A}$ on word $w = \sigma_0\sigma_1\sigma_2... \in \Sigma^\omega$ is an infinite sequence of transitions $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$, where $q_0 = q_I$ and each $(q_i, \sigma_i, q_{i+1}) \in \delta$. A run is *accepting* if it contains *infinitely many transitions* of each accepting set $F_i \in F$. A word w is accepted by $\mathcal{A}$ only if there exists an *accepting run* of $\mathcal{A}$ on w. A set of words, which are accepted by $\mathcal{A}$, represent a *language* $\mathcal{L}(\mathcal{A})$ of automaton.

Definition 2.3. *Preorder* $\sqsubseteq$ is a relation which is reflexive (for each x, $x \sqsubseteq x$ holds) and transitive ($x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$). In other words, $x \sqsubseteq y$, means that y covers x. *Partial order* is a preorder that is also antisymmetric ($x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$). *Strict partial order* $\sqsubset$ is an irreflexive ($x \not\sqsubseteq x$), asymmetric ($x \sqsubset y \implies y \not\sqsubset x$) and transitive relation. Preorder that is symmetric ($x \sqsubseteq y \implies y \sqsubseteq x$) is called *equivalence*, denoted by $\approx$.

3 Simulation relations and their applications on NBA

A simulation is a binary relation on the states of an automaton $\mathcal{A}$. Intuitively, considering $p, q \in Q$ to be states of $\mathcal{A}$, q simulates the behavior of p ($p \sqsubseteq q$), iff every move taken from state p can be repeated from state q. The main purpose of computing such relations is to reduce the size of an automaton (removing transitions as well as states) while preserving its language. Reduction techniques presented in this thesis are *quotienting* and *transition pruning*.

As in [2], we define simulations game-theoretically. The game is played by two players, Spoiler and Duplicator, where Duplicator wants to show that she can imitate any behavior of Spoiler.

In this chapter, we first recall simulation relations on Büchi automata as presented in [2]. Then, we describe algorithms using these simulations to reduce automata.

3.1 Forward simulation relations

Forward simulations firstly introduced in [3] are binary relations on the states of an NBA $\mathcal{A}$. The game between Spoiler and Duplicator starts at the initial configuration (p_0, q_0). Each round of the game unfolds in the same way. Given the configuration (p_i, q_i) (arbitrary round is considered as i^{th} round of the game), Spoiler chooses a symbol $\sigma_i \in \Sigma$ and a transition $p_i \xrightarrow{\sigma_i} p_{i+1}$. Duplicator then answers with a matching transition $q_i \xrightarrow{\sigma_i} q_{i+1}$. The game then continues with configuration (p_{i+1}, q_{i+1}).

The game either halts (Spoiler does not have any transition to choose or Duplicator can not respond to Spoiler's move) in which case the one who can not move looses or the game goes on forever, while the players build two infinite traces $\pi_1 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_2 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. Winner of the game is then chosen, depending on whether Duplicator fulfills the winning condition (for each type of simulation will be specified separately).

In this thesis we focus only on three forward simulations, namely *direct*, *delayed* and *fair* simulations.

Direct forward simulation $\sqsubseteq^{di}$ [4] is a relation where the strongest winning condition is put on Duplicator. She wins the game only if she visits accepting states in all rounds where Spoiler does.

In *delayed forward simulation* $\sqsubseteq^{de}$ [5], Duplicator is allowed to visit accepting states few (finitely many) steps later.

Fair forward simulation $\sqsubseteq^{f}$ [6] has the weakest demands on Duplicator. We call a trace $\pi_1 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ *fair*, when $p_i \in F$ holds for infinitely many i. In order to win the game, Duplicator needs to visit infinitely many accepting states only if Spoiler does so.

Formally, the winning condition for Duplicator for each type of simulation $x \in \{di, de, f\}$ is a predicate on both Spoiler's and Duplicator's traces π_1, π_2. Duplicator wins the game as far as $C^x(\pi_1, \pi_2)$ holds, where:

$$\begin{aligned}
C^{di}(\pi_1,\pi_2) &\overset{def}{\iff} \forall(i \geq 0) \,.\, p_i \in F \implies q_i \in F \\
C^{de}(\pi_1,\pi_2) &\overset{def}{\iff} \forall(i \geq 0) \,.\, p_i \in F \implies \exists(j \geq i) \,.\, q_j \in F \\
C^{f}(\pi_1,\pi_2) &\overset{def}{\iff} \text{if } \pi_1 \text{ is fair then } \pi_2 \text{ is also fair}
\end{aligned}$$

We define x-simulation relation $\sqsubseteq^x \subseteq Q \times Q$, for $x \in \{di, de, f\}$ so that $p \sqsubseteq^x q$ holds if Duplicator has a winning strategy in the x-simulation game starting at configuration (p, q).

3.2 Backward simulation relations

As with direct forward simulation, *backward simulation relation* [7] is defined similarly. The main difference is that the transitions are taken backwards.

A turn starting from the configuration (p_i, q_i) can be described as follows: Spoiler chooses a transition $p_{i+1} \xrightarrow{\sigma_i} p_i$ and Duplicator replies with transition $q_{i+1} \xrightarrow{\sigma_i} q_i$. The game continues at the configuration (p_{i+1}, q_{i+1}). As indicated in previous section, the game either halts (player with no move in the game looses, i.e. Spoiler looses if she can not take any transition, Duplicator looses when she is unable to respond to transition taken by Spoiler) or both players build two infinite traces $\pi_1 = \cdots \xrightarrow{\sigma_1} p_1 \xrightarrow{\sigma_0} p_0$ and $\pi_2 = \cdots \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_0} q_0$. Duplicator wins the game as far as she can match both initial and final states visited by Spoiler. Formally, we define a winning condition $C^{bw}(\pi_1, \pi_2)$ as:

$$\begin{aligned} C^{bw}(\pi_1, \pi_2) \overset{def}{\iff} & \forall (i \geq 0) \,.\, (p_i \in F \implies q_i \in F) \wedge \\ & \wedge \; (p_i = q_I \implies q_i = q_I) \end{aligned}$$

Backward simulation $\sqsubseteq^{bw}$ between two states p, q holds, written $p \sqsubseteq^{bw} q$, if Duplicator has a winning strategy in the backward simulation game beginning at configuration (p, q).

3.3 Lookahead simulation relations

Lookahead simulations [2] represent a type of simulations where Duplicator is given some more information on Spoiler's choices. At each round of the game, Spoiler chooses a sequence of length k ($k \geq 1$ and is fixed for the whole simulation game) of transitions and depending on this move Duplicator decides how much lookahead he really needs and matches m (where, $1 \leq m \leq k$) of Spoiler's transitions. With greater lookahead we obtain coarser relations, however, they are harder to compute. In the case $k = 1$, the lookahead simulation is identical to aforementioned simulations.

We first describe the k-lookahead forward simulation game. A configuration of the game is represented as a pair (p_i, q_i) and from this configuration the forward k-lookahead game unfolds as follows:

Spoiler chooses a sequence of transitions $p_i \xrightarrow{\sigma_i} p_{i+1} \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{i+k-1}} p_{i+k}$. According to his move, Duplicator needs to choose the degree of lookahead m, where $1 \leq m \leq k$ and replies with a sequence of transitions $q_i \xrightarrow{\sigma_i} q_{i+1} \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{i+m-1}} q_{i+m}$. The rest of Spoiler's moves ($p_{i+m} \xrightarrow{\sigma_{i+m}} p_{i+m+1} \xrightarrow{\sigma_{i+m+1}} \cdots \xrightarrow{\sigma_{i+k-1}} p_{i+k}$) is forgotten and the game then continues at configuration (p_{i+m}, q_{i+m}). As in previous simulations, this game either halts (Spoiler looses when she has no transition to move to, Duplicator looses when she can not respond to Spoiler's move) or continues forever, and two infinite traces $\pi_1 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_2 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ are created.

Duplicator wins the game for each type of the k-lookahead forward simulation, when $C^x(\pi_1, \pi_2)$ holds, where $x \in \{di, de, f\}$.

The k-lookahead backward simulation game from configuration (p_i, q_i) unfolds similarly:

Spoiler chooses a sequence of transitions $p_{i+k} \xrightarrow{\sigma_{i+k-1}} \cdots \xrightarrow{\sigma_{i+1}} p_{i+1} \xrightarrow{\sigma_i} p_i$. According to his move, Duplicator needs to choose the degree of lookahead m ($1 \leq m \leq k$) and replies with a sequence of transitions $q_{i+m} \xrightarrow{\sigma_{i+m-1}} \cdots \xrightarrow{\sigma_{i+1}} q_{i+1} \xrightarrow{\sigma_i} q_i$. Again, the rest of Spoiler's moves ($p_{i+k} \xrightarrow{\sigma_{i+k-1}} \cdots \xrightarrow{\sigma_{i+m+1}} p_{i+m+1} \xrightarrow{\sigma_{i+m}} p_{i+m}$) is forgotten and the game then continues at configuration (p_{i+m}, q_{i+m}). As mentioned before, this game either halts (the player without a move looses) or goes on forever, and two infinite traces $\pi_1 = \cdots \xrightarrow{\sigma_1} p_1 \xrightarrow{\sigma_0} p_0$ and $\pi_2 = \cdots \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_0} q_0$ are created. Duplicator wins the k-lookahead backward simulation game when $C^{bw}(\pi_1, \pi_2)$ holds.

Two states p, q are in k-lookahead x-simulation relation, for $x \in \{di, de, f\}$, denoted as $p \sqsubseteq^{k-x} q$, when Duplicator has a winning strategy in the k-lookahead x-simulation game starting at configuration (p, q). In spite of k-lookahead simulations being non-transitive relations (for $k \neq 1$, see Figure 3.1 and [2]) it does not hinder their applications. We consider their transitive closures $\preceq^{k-x}$ (which is a preorder) and their asymmetric restrictions $\prec^{k-x}$ instead.

Figure 3.1 shows an example of 2-lookahead simulation (not depending on which $x \in \{di, de, f\}$ we choose, thus denoted $\sqsubseteq^2$). Notice that $q_0 \not\sqsubseteq^1 r_0$, since Duplicator has to choose whether she goes to r_1 (Spoiler wins playing b) or to r_2 (Spoiler wins playing a). However, with lookahead $k = 2$, Duplicator can decide on taking route via r_1 or r_2, depending on Spoiler's second transition (whether she plays word $(a+b)a$ or $(a+b)b$). Reasoning mentioned above induces, that $q_0 \sqsubseteq^2 r_0$ holds. This Figure is also an example of non-transitivity of lookahead relations. As we have shown above $q_0 \sqsubseteq^2 r_0$, using similar reasoning one can see that $r_0 \sqsubseteq^2 s_0$ (as $r_1 \sqsubseteq^2 s_1$ and $r_2 \sqsubseteq^2 s_2$). However, states q_0 and s_0 are not in 2-lookahead simulation (not even in any k-lookahead simulation, with $k > 1$). Duplicator would always need to know the first step taken by Spoiler in the next round (if Duplicator ends at state s_1 (s_2), Spoiler can win by playing b (a), respectively).

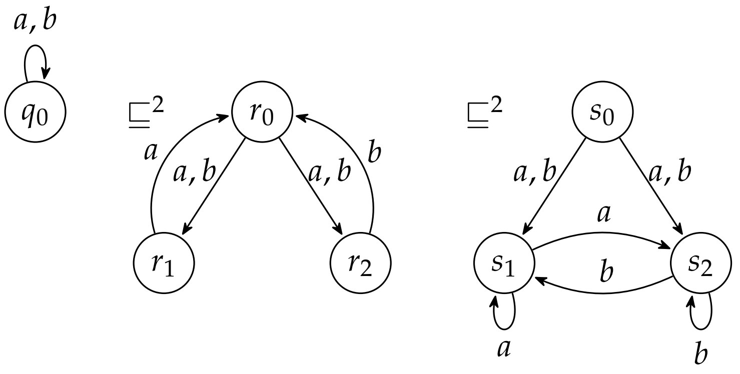


Figure 3.1: Example of 2-lookahead simulation. Adapted from Clemente and Mayr [2].

k-lookahead simulation can be seen as a case of *multipebble simulations* (n-pebble) [8]. Multipebble simulations are a type of simulations where Duplicator is given n pebbles that she can use to hedge her bets. The main purpose of having more pebbles is that Duplicator does not have to decide (when she has more transitions to use) which transition to take and thus track more of them at one time. The k-lookahead simulations can be expressed as restricted n-pebble simulation (in this case n is the number of states of an automaton), but after $m \leq k$ rounds (m is chosen by Duplicator - the lookahead she really uses in her move) she can keep only a single pebble.

Table 3.1 provides a summary of relations introduced in this chapter and notations of winning conditions for Duplicator for each simulation game.

Table 3.1: Overview of winning condition notations and forward and backward simulation relations for NBA. Fourth column shows notations of transition closures of k-lookahead simulations.

Winning condition	**Forward simulation**	**Backward simulation**	**k-lookahead simulation**
C^{di}	$\sqsubseteq^{di}$	-	$\preceq^{k-di}$
C^{de}	$\sqsubseteq^{de}$	-	$\preceq^{k-de}$
C^{f}	$\sqsubseteq^{f}$	-	$\preceq^{k-f}$
C^{bw}	-	$\sqsubseteq^{bw}$	$\preceq^{k-bw}$

Figure 3.2 shows an overview of relationships between preorders introduced in this chapter. An arrow means inclusion between preorders. Notice that backward simulation relation as well as transitive closure of its k-lookahead variation is incomparable with forward simulation relations and transitive closures of their k-lookahead variations.

$$\begin{tikzcd}
\sqsubseteq^{di} \arrow[rr] \arrow[dr] & & \preceq^{k-di} \arrow[dr] & & & \sqsubseteq^{bw} \arrow[rr] & & \preceq^{k-bw} \\
& \sqsubseteq^{de} \arrow[rr] \arrow[dr] & & \preceq^{k-de} \arrow[dr] & & & & \\
& & \sqsubseteq^{f} \arrow[rr] & & \preceq^{k-f} & & &
\end{tikzcd}$$

Figure 3.2: Scheme showing relationships of forward and backward simulation relations and transitive closures of their k-lookahead variants. Adapted from Clemente and Mayr [2]

3.4 Quotienting

Quotienting is one of the basic language preserving techniques used for reducing the state space of an automaton. Its purpose is merging states that are equivalent (according to some given preorder) into a single state.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be an automaton and $\sqsubseteq$ be a preorder on Q, with induced equivalence $\equiv := (\sqsubseteq \cap \sqsupseteq)$. By $[p]$ we denote the equivalence class of state $p \in Q$ with respect to $\equiv$, and for subset of states $P \subseteq Q$, let $[P]$ be a set of equivalence classes $[P] = \{[p] \mid p \in P\}$. We define the *quotient* of an automaton $\mathcal{A}$ by preorder $\sqsubseteq$ as a tuple $\mathcal{A}/\sqsubseteq = ([Q], \Sigma, \delta', [q_0], [F])$, where $\delta' = \{([q], \sigma, [q']) \mid \exists q_1 \in [q], q_1' \in [q'] \,.\, (q_1, \sigma, q_1') \in \delta\}$.

One can easily confirm, that for every trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ of $\mathcal{A}$ on $w = \sigma_1\sigma_2\sigma_3\cdots$, there exists a corresponding trace $\pi' = [q_0] \xrightarrow{\sigma_0} [q_1] \xrightarrow{\sigma_1} \cdots$ in $\mathcal{A}/\sqsubseteq$, which is fair/initial if the former one is fair/initial, respectively. Thereby, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}/\sqsubseteq)$ holds for any preorder $\sqsubseteq$. We call a preorder *good for quotienting* (GFQ), as far as the converse inclusion $\mathcal{L}(\mathcal{A}/\sqsubseteq) \subseteq \mathcal{L}(\mathcal{A})$ holds.

Our main interest is to find coarse and efficiently computable GFQ preorders (the coarser the preorder is, the 'bigger' the equivalence is, which may lead to smaller automata). Classical examples of GFQ preordes are x-simulations, where x $\in \{di, de\}$, and backward simulation relations [9, 5] (recall their definitions from Sections 3.1 and 3.2) as well as their k-lookahead variations [2] (see Section 3.3). Formally, $\sqsubseteq^{di}, \sqsubseteq^{de}, \sqsubseteq^{bw}, \preceq^{k-di}, \preceq^{k-de}$ *and* $\preceq^{k-bw}$ are GFQ preorders.

As showed in [5], quotienting the automaton with respect to the fair simulation relation does not preserve the language of the automaton. Hence, neither $\sqsubseteq^{f}$, nor $\preceq^{k-f}$ are GFQ preorders.

Figure 3.2 *a*) shows an automaton where states q_1 and q_3 are equivalent with respect to direct simulation and therefore are merged into a single state $r_{1,3}$ in *b)*. Notice that, though states q_1, q_3 directly simulate state q_2, the converse simulation does not hold. Hence, states q_2 and q_1, q_3 do not belong to one equivalence class.

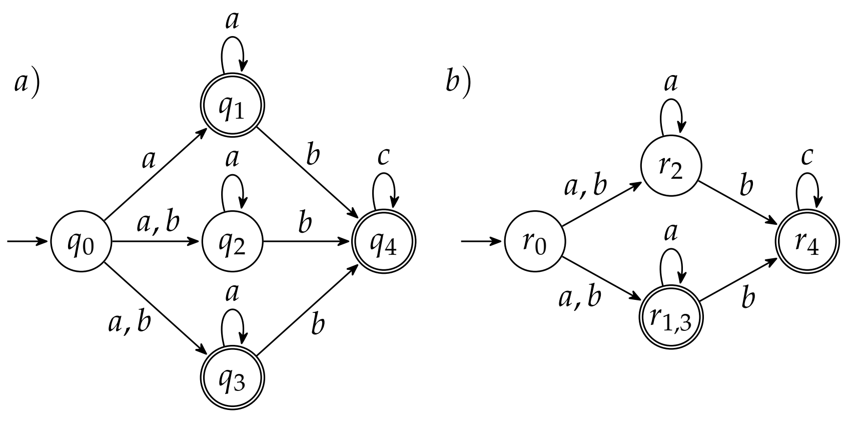


Figure 3.3: *a)* shows an automaton $\mathcal{A}$, *b)* shows its quotient automaton $\mathcal{A}/\sqsubseteq^{di}$. State $r_{1,3} = [q_1] = [q_3]$ and for remaining states r_0, r_2, r_4 of $\mathcal{A}/\sqsubseteq^{di}$ holds $r_i = [q_i]$, for $i \in \{0, 2, 4\}$.

3.5 Transition pruning

Another approach reducing the number of states of an automaton is *transition pruning* (i.e. removing). Intuitively, certain transition can be removed due to the existence of some 'better' transitions without changing the language of an automaton.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be an automaton and R_f, R_b be a binary state relations for forward and backward endpoints of transitions, respectively. We define a binary relation $P(R_b, R_f) \subseteq \delta \times \delta$ on transitions as:

$$P(R_b, R_f) = \{((p, \sigma, q), (p', \sigma, q')) \in \delta \times \delta \mid p\ R_b\ p' \text{ and } q\ R_f\ q'\}$$

The pruned automaton [2] is defined as a tuple $Prune(\mathcal{A}, P) = (Q, \Sigma, \max P, q_I, F)$, where $\max P$ is a set of maximal elements of P:

$$\max P = \{(p, \sigma, q) \in \delta \mid \nexists (p', \sigma', q') \in \delta \,.\, ((p, \sigma, q), (p', \sigma', q')) \in P\}$$

Since transition pruning can not introduce new words into the language, $\mathcal{L}(Prune(\mathcal{A}, P)) \subseteq \mathcal{L}(\mathcal{A})$. When also the converse inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(Prune(\mathcal{A}, P))$ holds, we say that P is *good for pruning* (GFP).

In this thesis, we are interested mainly in the following GFP relations: $P(id, \prec^{k-\mathrm{di}})$ and $P(\prec^{k-\mathrm{bw}}, id)$ (in [10] called little brothers), $P(\sqsubset^{bw}, \preceq^{k-\mathrm{di}})$ and $P(\preceq^{k-\mathrm{bw}}, \sqsubset^{di})$ [2].

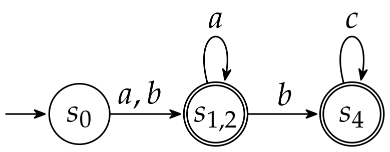


Figure 3.4: Simplified automaton from Figure 3.2 b) w.r.t. $P(id, \preceq^{di})$.

In Figure 3.3, we show that automaton from Figure 3.2 $b)$ can be further simplified. Notice that both $r_2 \sqsubseteq^{di} r_{1,3}$ and $r_2 \sqsubseteq^{bw} r_{1,3}$ holds. Therefore, we can prune the automaton both with respect to $P(id, \prec^{di})$ and $P(\prec^{bw}, id)$. In the first case, both transition $r_0 \xrightarrow{a} r_2$ and $r_0 \xrightarrow{b} r_2$ will be removed, since they are both subsumed by $r_0 \xrightarrow{a,b} r_{1,3}$ with $r_2 \sqsubseteq^{di} r_{1,3}$. This causes r_2 to become a dead state (it is no more reachable from the initial state) and thus can be removed as well. Considering the second case, we remove transition $r_2 \xrightarrow{b} r_4$, since it is subsumed by $r_{1,3} \xrightarrow{b} r_4$ with $r_2 \sqsubseteq^{bw} r_{1,3}$. As in previous case, state r_2 becomes dead (any accepting state is no longer reachable from it), and thus r_2 can be removed.

Each of the aforementioned approaches to the reduction of automata leads to reduction to some extent. However, by combining these algorithms, greater overall reductions of automata are obtained in general.

4 Simulation relations on TGBA

One of the goals of this thesis was to extend definitions of the considered simulations to TGBA. The definitions of following *simulation relations for TGBA* are similar to the ones showed in previous sections. One of the changes of the winning conditions is that we no longer consider Spoiler and Duplicator visiting infinite number of accepting states during the run, but we concentrate on the transitions. For each simulation relation, we give a definition of the winning condition for Duplicator (see Sections 3.1, 3.2, 3.3 to compare with previous definitions).

In next sections, consider $M = (Q, \Sigma, \delta, q_I, F)$ to be a TGBA with $F = \{F_1, F_2, ..., F_n\}$ being the set of accepting sets of transitions.

4.1 Forward simulation relations

The forward simulation game between Spoiler and Duplicator either halts (Spoiler has no transition to take and thus looses, or Duplicator looses, since she can not respond to Spoiler's move) or creates two infinite traces $\pi_1 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_2 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. For $x \in \{di, de, f\}$, Duplicator wins the x-simulation game when $D^x(\pi_1, \pi_2)$ holds.

4.1.1 Direct forward simulation relation

Duplicator wins the direct forward simulation game if the following holds: Whenever Spoiler takes a transition belonging to any accepting set of transitions, she can respond with matching transition belonging to the same accepting set. Formally we define the winning condition for Duplicator in the *direct forward simulation game*:

$$D^{di}(\pi_1, \pi_2) \overset{def}{\iff} \forall (i \geq 0) \forall (1 \leq j \leq n) \,.\, \Big[(p_i \xrightarrow{\sigma_i} p_{i+1}) \in F_j \implies \\ \implies (q_i \xrightarrow{\sigma_i} q_{i+1}) \in F_j \Big]$$

4.1.2 Delayed forward simulation relation

Considering the *delayed simulation relation*, the definition of the winning condition for Duplicator can be extended to TGBA in multiple ways leading to relations with different coarseness. We present three possible extensions of $C^{de}(\pi_1, \pi_2)$ marked as D_1^{de}, D_2^{de} and D_3^{de}. When using the winning condition D_3^{de}, we get a coarser simulation relation than if we consider D_2^{de}, which is still coarser than relation obtained with D_1^{de} (remember that the coarser the simulation relation is, the greater reductions we can achieve).

$$D_1^{de}(\pi_1, \pi_2) \overset{def}{\iff} \forall(i \geq 0)\forall(1 \leq j \leq n) \,.\, \Big[(p_i \xrightarrow{\sigma_i} p_{i+1}) \in F_j \implies \\ \implies \exists(k \geq i) \,.\, (q_k \xrightarrow{\sigma_k} q_{k+1}) \in F_j\Big]$$

Intuitively, the first winning condition for Duplicator $D_1^{de}(\pi_1, \pi_2)$ says that, each time Spoiler takes a transition belonging to any accepting set, Duplicator needs to take some rounds later (finitely many) a transition belonging to the same accepting set.

$$D_2^{de}(\pi_1, \pi_2) \overset{def}{\iff} \forall(i \geq 0) \,.\, \Big[\Big(\exists(i_1, \ldots, i_n \geq i) \,.\, (p_{i_1} \xrightarrow{\sigma_{i_1}} p_{i_1+1}) \in F_1 \wedge \\ \wedge \ldots \wedge (p_{i_n} \xrightarrow{\sigma_{i_n}} p_{i_n+1}) \in F_n\Big) \implies \exists(j_1, \ldots, j_n \geq i). \\ .\, (q_{j_1} \xrightarrow{\sigma_{j_1}} q_{j_1+1}) \in F_1 \wedge \ldots \wedge (q_{j_n} \xrightarrow{\sigma_{j_n}} q_{j_n+1}) \in F_n\Big]$$

The second winning condition $D_2^{de}(\pi_1, \pi_2)$ says that, in every step of the game holds if Spoiler takes transitions belonging each accepting set, then Duplicator also has to take some rounds later (finitely many) transitions belonging to each accepting set.

$$D_3^{de}(\pi_1, \pi_2) \overset{def}{\iff} \forall(i \geq 0). \Big[\Big(\exists(i_1, \ldots, i_n \text{ such that } i \leq i_1 \leq \ldots \leq i_n). \\ (p_{i_1} \xrightarrow{\sigma_{i_1}} p_{i_1+1}) \in F_1 \wedge \ldots \wedge (p_{i_n} \xrightarrow{\sigma_{i_n}} p_{i_n+1}) \in F_n\Big) \implies \\ \implies \exists\, (j_1, \ldots, j_n \geq i) \,.\, q_{j_1} \xrightarrow{\sigma_{j_1}} q_{j_1+1}) \in F_1 \wedge \ldots \wedge \\ \wedge\, (q_{j_n} \xrightarrow{\sigma_{j_n}} q_{j_n+1}) \in F_n\Big]$$

The third winning condition $D_3^{de}(\pi_1, \pi_2)$ says that, in every step of the game holds if Spoiler takes transitions belonging to each accepting set (in order $F_1, F_2, ..., F_n$), then Duplicator has to take some rounds later (finitely many) transitions belonging to each accepting set.

Simple example of delayed simulation game can be seen in Figure 4.1, where q_1, q_2 are delayed-simulation equivalent since both $q_1 \sqsubseteq^{de} q_2$ and $q_2 \sqsubseteq^{de} q_1$ holds. Thus, the states q_1, q_2 can be merged by quotienting into one equivalence class. Since we are now working with TGBA, we need to modify the definition of the quotient automaton. $\mathcal{A}/\sqsubseteq = ([Q], \Sigma, \delta', [q_I], F')$, where $\delta' = \{([q], \sigma, [q']) \mid \exists q_1 \in [q],\ q_1' \in [q'] \ .\ (q_1, \sigma, q_1') \in \delta\}$ and $F' = \{F_1', F_2', ...F_n'\}$, where $F_i' = \{([q], \sigma, [q']) \mid (q, \sigma, q') \in F_i\}$

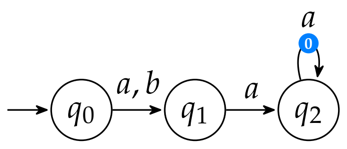


Figure 4.1: Example of forward delayed simulation.

4.1.3 Fair forward simulation relation

For *fair simulation relation* we present two definitions of winning conditions for Duplicator (depending of the definition of *fairness*) D_1^f and D_2^f. Using the D_2^f we gain coarser relations than when we consider D_1^f.

- If we define a *fair* trace for each acceptance set separately: a trace is F_j-fair for given F_j, if $(p_i \xrightarrow{\sigma_i} p_{i+1}) \in F_j$ holds for infinitely many i. Then the winning condition for Duplicator can be defined as:

$$D_1^f(\pi_1, \pi_2) \overset{def}{\Longleftrightarrow} \text{for each acceptance set } F_j \text{ it holds}$$
$$\text{if } \pi_1 \text{ is } F_j\text{-fair then } \pi_2 \text{ is also } F_j\text{-fair}$$

- Consider that we define a *fair* trace such that: $(p_i \xrightarrow{\sigma_i} p_{i+1}) \in F_j$ holds for each acceptance set F_j for infinitely many i (i.e. a trace contains infinitely many transitions of each accepting set F_j), then the winning condition for Duplicator is:

$$D_2^f(\pi_1, \pi_2) \overset{def}{\iff} \text{if } \pi_1 \text{ is fair then } \pi_2 \text{ is also fair}$$

4.2 Backward simulation relations

Let us consider the backward simulation relation. The game either halts (Spoiler can not take any transition or Duplicator can not respond to Spoilers' move) or continues forever while the players built two infinite traces $\pi_1 = \cdots \xrightarrow{\sigma_1} p_1 \xrightarrow{\sigma_0} p_0$ and $\pi_2 = \cdots \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_0} q_0$. The winning condition $D^{bw}(\pi_1, \pi_2)$ for Duplicator on TGBA is defined as follows:

$$D^{bw}(\pi_1, \pi_2) \overset{def}{\iff} \forall (i \geq 0) \forall (1 \leq j \leq n) \,.\, \Big((p_i, \sigma_i, p_{i+1}) \in F_j \implies$$
$$\implies (q_i, \sigma_i, q_{i+1}) \in F_j \Big) \text{ and } (p_i = q_I \implies q_i = q_I)$$

Intuitively, each time Spoiler takes transition belonging to any accepting set, Duplicator has to respond by taking a transition belonging to the same accepting set.

Figure 4.2 brings an example of both direct forward ($q_2 \sqsubseteq^{di} q_3$ and $q_1 \sqsubseteq^{di} q_3$) and backward ($q_2 \sqsubseteq^{bw} q_3$) simulation relations on TGBA.

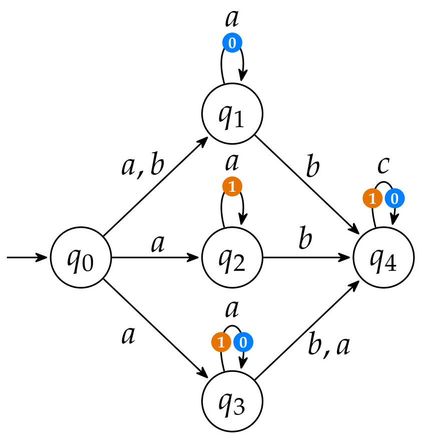


Figure 4.2: Examples of backward and direct forward simulation.

4.3 Lookahead simulation relations

The game between Duplicator and Spoiler runs similarly as described in Section 3.3. Hence, the game either halts (Spoiler can not take any transition, in which case she looses or Duplicator looses, since she can not respond to Spoilers' move) or the players built two infinite traces $\pi_1 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_2 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. Duplicator has a winning strategy in the x-forward simulation game if $D^x(\pi_1, \pi_2)$ holds for $x \in \{di, de, f\}$. Considering the k-lookahead backward simulation game, the game also either halts or creates two infinite traces $\pi_1 = \cdots \xrightarrow{\sigma_1} p_1 \xrightarrow{\sigma_0} p_0$ and $\pi_2 = \cdots \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_0} q_0$. Duplicator wins this game if $D^{bw}(\pi_1, \pi_2)$ holds.

A summary showing definitions of winning conditions for Duplicator for each simulation relation can be found in Table 4.1.

Table 4.1: Overview of winning condition notations and forward and backward simulation relations for TGBA. Fourth column shows notations of transition closures of k-lookahead simulations.

Winning condition	Forward simulation	Backward simulation	k-lookahead simulation
D^{di}	$\sqsubseteq^{di}$	-	$\preceq^{k-di}$
$D^{de}_1, D^{de}_2, D^{de}_3$	$\sqsubseteq^{de}_1, \sqsubseteq^{de}_2, \sqsubseteq^{de}_3$	-	$\preceq^{k-de}_1, \preceq^{k-de}_2, \preceq^{k-de}_3$
D^f_1, D^f_2	$\sqsubseteq^f_1, \sqsubseteq^f_2$	-	$\preceq^{k-f}_1, \preceq^{k-f}_2$
D^{bw}	-	$\sqsubseteq^{bw}$	$\preceq^{k-bw}$

5 Computation of selected simulations

In this chapter we present a simple algorithm reducing the computation of selected simulation relations to parity game solving. *Parity game* is a game played on a graph G with two disjoint sets of vertices, where each vertex is given some priority. Formally, $G = (V_0, V_1, E, p)$, where V_0, V_1 are two disjoint sets of vertices (their union is denoted as V), $E \subseteq V \times V$ are edges between the graph vertices and p is a function assigning each vertex a *priority*.

Parity game $P = (G, v_0)$ played on a graph G, starting from $v_0 \in V$, is played by two players, *Even* (having V_0 vertices) and *Odd* (having vertices V_1). At the beginning of the game, the pebble is placed on vertex v_0. In each step of the game, the pebble is moved from vertex $v_i \in V_0(V_1)$ by Even (Odd) Player to vertex v_{i+1}, such that $(v_i, v_{i+1}) \in E$. If one of the players can not move, since there is no outgoing edge she would take, the game halts and the player looses. Otherwise, the game goes on forever creating a path $\pi = v_0, v_1, v_2 ....$ The winner of the game is determined depending on the lowest priority s_π occurring infinitely often in π. Even wins the game if s_π is even, Odd wins when s_π is odd.

Games between two players, Spoiler and Duplicator, as presented in Chapter 3, can easily be transformed into a parity game graph $G^x_{\mathcal{A}}$, where $\mathcal{A}$ is NBA and $x \in \{di, de, f\}$ [5]. The construction of parity game graphs for each simulation game follows the same pattern.

We provide the construction of $G^{di}_{\mathcal{A}} = (V^{di}_0, V^{di}_1, E^{di}_{\mathcal{A}}, p^{di}_{\mathcal{A}})$. Each vertex of the game will be given the same priority $p^{di}_{\mathcal{A}}(v) = 0$. For each pair of states $(p, q) \in Q^2$ of $\mathcal{A}$, we create a corresponding vertex $v_{(p,\, q)} \in V^{di}_1$, such that if q directly simulates p, then Even player has a winning strategy from $v_{(p,\, q)}$. Since we need to take care of the winning condition, we eliminate incoming and outgoing edges of vertices in the parity game, where Odd player has his pebble on a final state and Even player does not.

Formally, we define the $G^{di}_{\mathcal{A}}$ as follows:

$$\begin{aligned}
V_1^{di} =& \{v_{(p,\,q)} \mid p, q \in Q\},\\
V_0^{di} =& \{v_{(p',\,q,\,\sigma)} \mid p', q \in Q,\ \exists p \in Q \,.\, (p, \sigma, p') \in \delta\},\\
E_{\mathcal{A}}^{di} =& \{(v_{(p,\,q)},\ v_{(p',\,q,\,\sigma)}) \mid (p, \sigma, p') \in \delta,\ p \in F \implies q \in F\}\\
& \cup \{(v_{(p,\,q,\,\sigma)},\ v_{(p,\,q')}) \mid (q, \sigma, q') \in \delta,\ p \in F \implies q' \in F\}
\end{aligned}$$

A situation in a simulation game when Spoiler takes a transition $p \xrightarrow{\sigma} p' \in \delta$ will in a parity game be represented by an edge between vertices $v_{(p,\,q)}$ and $v_{(p',\,q,\,\sigma)}$. On the other hand, an edge between vertices $v_{(p',\,q,\,\sigma)}$ and $v_{(p',\,q')}$ will denote Duplicators' response to Spoilers' move, taking a transition $q \xrightarrow{\sigma} q' \in \delta$.

The winning vertices for Even player in the parity game graph $G_{\mathcal{A}}^x, x \in di, de, f$ represent the pairs of states (p, q) of $\mathcal{A}$ such that $p \sqsubseteq^x q$. We now extend the construction of the game graphs for $direct forward$ and *backward* simulations to work with TGBA.

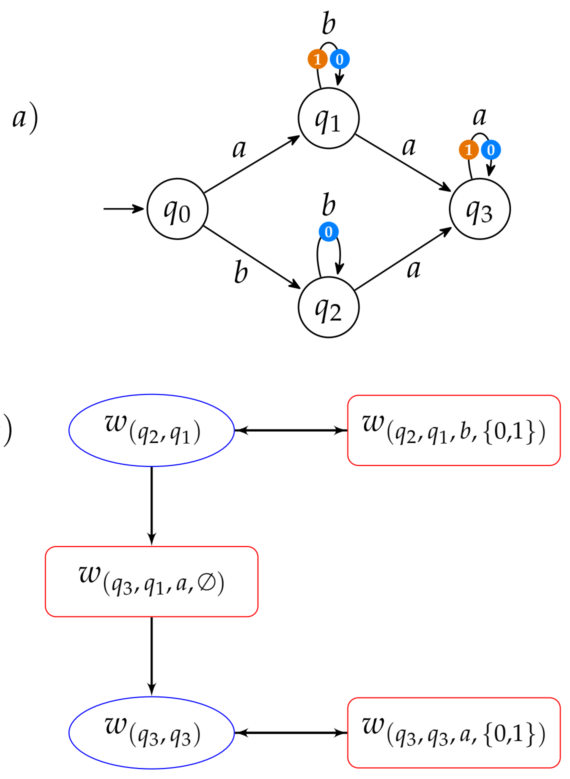
For the next sections, let us fix $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ to be a TGBA, where $F = \{F_1, F_2, \ldots F_n\}$ is the set of accepting sets of transitions. We indicate membership of a transition to accepting sets by using accepting marks on transitions. (i.e. consider $(p_i, \sigma_i, p_{i+1}) \in F_1$, the membership of this transition to F_1 is denoted with accepting mark ❶).

5.1 Direct parity game for TGBA

We denote the game graph of the direct parity game for the TGBA as $H_{\mathcal{A}}^{di}$, such that $H_{\mathcal{A}}^{di} = (W_0^{di}, W_1^{di}, Ed_{\mathcal{A}}^{di}, r_{\mathcal{A}}^{di})$, where W_0^{di} and W_1^{di} are two disjoint sets of vertices (their union is denoted as W), $Ed_{\mathcal{A}}^{di} \subseteq W \times W$ are edges between the vertices and $r_{\mathcal{A}}^{di}$ is a function giving each vertex some priority. In this simulation game we also have only one priority (i.e. every vertex has assigned priority 0). For each pair of states $(p, q) \in Q \times Q$, we create a vertex $w_{(p,\,q)} \in W_1^{di}$, such that if q directly simulates p ($p \sqsubseteq^{di} q$), then Even has a winning strategy from vertex $w_{(p,\,q)}$. Example of a parity game graph and automaton from which the game was constructed is shown in Figure 5.1.

Formally, $H^{di}_{\mathcal{A}}$ is defined as follows:

$$
\begin{aligned}
W^{di}_1 =& \{w_{(p,\,q)} \mid p, q \in Q\} \\
W^{di}_0 =& \{w_{(p',\,q,\,\sigma,\,F')} \mid p', q \in Q,\ \exists p \in Q \,.\, (p, \sigma, p') \in \delta), \text{ such that, } F' \\
& \qquad \text{is the set of accepting marks on the transition } (p, \sigma, p')\} \\
Ed^{di}_{\mathcal{A}} =& \{(w_{(p,\,q)}, w_{(p',\,q,\,\sigma,\,F')}) \mid (p, \sigma, p') \in \delta \text{ with } F' \text{ being the set} \\
& \qquad \text{of accepting marks on the transition } (p, \sigma, p')\} \\
& \cup \{(w_{(p,\,q,\,\sigma,\,F')}, w_{(p,\,q')}) \mid (q, \sigma, q') \in \delta \text{ with } F'' \text{ being the set of} \\
& \qquad \text{accepting marks on the transition } (q, \sigma, q') \text{ and } F' \subseteq F''\}
\end{aligned}
$$



a)
b)
q_0
q_1
q_2
q_3
a
b
1
0
$w_{(q_2,q_1)}$
$w_{(q_2,q_1,b,\{0,1\})}$
$w_{(q_3,q_1,a,\emptyset)}$
$w_{(q_3,q_3)}$
$w_{(q_3,q_3,a,\{0,1\})}$

Figure 5.1: b) presents a part of the parity game graph $H^{di}_{\mathcal{A}}$ for automaton in a). This part of $H^{di}_{\mathcal{A}}$ shows that Even player has a winning strategy from vertex $w_{(q_2,q_1)}$.

The parity game mimics the direct forward simulation game. A vertex $w_{(p,\,q)}$ represents a situation in which Spoiler is on state p, Duplicator is on state q and it is Spoiler's time to move. A vertex $w_{(p,\,q,\,\sigma,\,F')}$ represents a situation when Duplicator is on move and needs to respond to Spoiler's move, who has taken a transition under σ with a set of accepting marks F'. In order to respect the winning condition of the direct simulation game, we need to eliminate edges where Odd player uses transition belonging into any accepting set, and Even player does not have such a transition. Therefore we store the information about any accepting marks Spoiler has visited in his run and we require Duplicator to visit at least the same set of accepting marks.

5.2 Backward parity game for TGBA

The computation of the backward simulation relation can be transformed into backward parity game similarly as direct simulation relation. We denote the game graph of the backward parity game as $H^{bw}_{\mathcal{A}} = (W^{bw}_0, W^{bw}_1, Ed^{bw}_{\mathcal{A}}, r^{bw}_{\mathcal{A}})$. For each pair of states $p, q \in Q$, we create a vertex $w_{(p,\,q)} \in W^{bw}_1$ such that Even player has a winning strategy from this vertex iff q backwardly simulates p ($p \sqsubseteq^{bw} q$). As in previous game, each vertex will be assigned priority 0.

Formally, the game graph of the backward parity game $H^{bw}_{\mathcal{A}}$ is defined as follows:

$$\begin{aligned}
W^{bw}_1 =& \{w_{(p,\,q)} \mid p, q \in Q\}\\
W^{bw}_0 =& \{w_{(p,\,q,\,\sigma,\,F')} \mid p, q \in Q,\ \exists p' \in Q \,.\, (p, \sigma, p') \in \delta),\text{ such that, } F'\\
&\quad \text{is the set of accepting marks on the transition } (p, \sigma, p')\}\\
Ed^{bw}_{\mathcal{A}} =& \{(w_{(p,\,q)}, w_{(p',\,q,\,\sigma,\,F')}) \mid (p', \sigma, p) \in \delta \text{ with } F' \text{ being the set}\\
&\quad \text{of accepting marks on the transition } (p', \sigma, p)\}\\
&\cup \{(w_{(p,\,q,\,\sigma,\,F')}, w_{(p,\,q')}) \mid (q', \sigma, q) \in \delta \text{ with } F'' \text{ being the set}\\
&\quad \text{of accepting marks on the transition } (q', \sigma, q),\ F' \subseteq F''\\
&\quad \text{and } (p = q_I) \implies (q' = q_I)\}
\end{aligned}$$

The edge $(w_{(p,\,q)}, w_{(p',\,q,\,\sigma,\,F')})$ represents Spoilers' move in which she takes an edge $p' \xrightarrow{\sigma} p \in \delta$ with F' being a set of accepting marks on this edge. The edge $(w_{(p,\, q,\, \sigma,\, F')}, w_{(p,\, q')})$ represents the move of Duplicator (response to Spoilers' move) in which she takes an edge $q' \xrightarrow{\sigma} q \in \delta$ with a set F'' of accepting marks on this transition. In order to respect the winning condition of the backward simulation game we need to eliminate edges which represent a situation when Odd player took an edge belonging to any accepting set and Even does not have such an edge. The second thing that needs to hold is that any time Odd player visits an initial state in his move, Even player must respond by visiting an initial state as well. Since the winner of the game is determined depending on the lowest priority occurring in the parity game infinitely often, we need to create an additional vertex w'. This vertex together with edges $(w_{(q_I,\, q_I)}, w')$ and $(w', w_{(q_I,\, q_I)})$ guarantees that a parity game will continue forever and at the same time does not influence the computation of simulating states.

The winning vertices for Even player $w_{(p,\, q)}$ in the parity game graph $H^{bw}_{\mathcal{A}}$ represent the pairs of states (p, q) of $\mathcal{A}$ such that $p \sqsubseteq^{bw} q$.

6 Implementation and evaluation

In this chapter we present details of implementation of our tools QUOT and PRUNE, its usage and evaluation of the results.

6.1 Implementation

In this thesis we have implemented direct forward and backward simulation relations fot TGBA using parity games and then algorithms employing these simulations for automata reduction. Both simulations and algorithms are implemented in Python language over the Spot library [11]. In order to represent and modify the automata we use classes and functions implemented in Spot. Besides Spot, our implementation uses several other Python libraries, namely `sys`, `os` and `argparse`.

Our implementation does not involve computation of the parity game. We provide only the algorithm reducing computation of selected simulation relations to parity game solving (thus creating vertices of the game and edges between them). The parity game itself is computed by [12]. The results of the game are then parsed and used for computing the simulation relations.

6.2 Usage

Our implementation contains two Linux command line tools, namely QUOT and PRUNE. The core of our QUOT tool is quotienting the automata with respect to direct forward or backward simulation. The tool PRUNE performs transition pruning with respect to these GFP relations: $P(id, \sqsubseteq^{di})$, $P(\sqsubseteq^{bw}, id)$ and $P(\sqsubseteq^{bw}, \sqsubseteq^{di})$. Tables 6.1 and 6.2 show arguments used in tools with their description. Apart from these arguments, the tools can also read automaton in the *Hanoi Omega format* (HOA) [13] from `stdin`, which enables to apply the algorithms consecutively.

Table 6.1: Arguments used in QUOT.

Argument	Choices	Description
`-h`	-	prints possible arguments and their description
`--simulation`[1]	`di,bw`	simulation to apply
`-f, --formula`	-	formula to translate
`--translator`	`ltl3ba [14], ltl2tgba [15]`	translator for translating the formula
`--preprocess`	`0,1`	simplify the formula before calling tools
`--postprocess`	`0,1`	simplify the produced automaton using Spot

[1] Obligatory argument

Table 6.2: Arguments used in PRUNE. In this case postprocessing of resulted automata is redundant, since it is already a part of the tool.

Argument	Choices	Description
`-h`	-	prints possible arguments and their description
`--simulation`[1]	`id-di,bw-id, bw-di`	simulation to apply
`-f, --formula`	-	formula to translate
`--translator`	`ltl3ba, ltl2tgba`	translator for translating the formula
`--preprocess`	`0,1`	simplify the formula before calling tools

[1] Obligatory argument

6.3 Experimental evaluation

In this section we evaluate our tools QUOT and PRUNE. Results presented in this section were gained using `ltlcross-runner` from [16], which calls `ltlcross`. Ltlcross [11] is a tool for cross-comparing the output of translation of formulas in linear temporal logic (LTL [17]) into automata. The core of this tool is to translate the formula and its negation with selected translator into automata. On these automata, sanity check is then performed.

Experiments have been performed on a notebook with Intel i7-7500U (2.70GHz) processor and 8GB RAM. All computations have been run with 10 minute timeout.

We used `randltl` [18] for creating 500 random formulas in LTL. Since we did not remove equivalent formulas, the tests were performed on together 497 formulas and their negations (thus 994 formulas in total). Formulas were then translated with the tool LTL3BA (implemented based on improved translation of LTL formulas presented in [14])

6.3.1 Evaluation of QUOT

In our presentation, let us fix QUOT-di to denote quotienting the automaton with respect to direct forward simulation relation and QUOT-bw to denote quotienting with respect to backward simulation relation. In following tables we present evaluation of our tool.

Table 6.3 presents overall number of states of automata produced by four tools LTL3BA, QUOT-di, QUOT-bw and Spot (where the automaton produced by LTL3BA was an input automaton for other thee tools).

Table 6.4 shows number of times QUOT-di had better results compared to LTL3BA, based on either number of states or edges as criterion.

Table 6.5 sums, how many times QUOT-di reached better, same or worse results compared to Spot, considering the reduction in either states or edges.

Table 6.3: Overall results of our tools compared to LTL3BA and post-processing reductions implemented in Spot.

Tool	**Number of states**
LTL3BA	4827
QUOT-di	4272
QUOT-bw	4272
Spot	4255

Figure 6.1 presents a graph showing number of states of automata produced by QUOT-di compared to LTL3BA.

These results show, that our tool QUOT brings significant reduction of the automata produced by LTL3BA (smaller automaton on 246 formulas and removes 555 states in total). Considering the number of states produced by QUOT and Spot, reduction algorithms implemented in QUOT resulted in smaller automata in 21 cases, whereas Spot was more successful in 34 automata reductions. Computation of direct forward and backward simulation relations are already implemented in Spot, however our QUOT tool approaches the solving in a different way, which still proves to be successful.

Table 6.4: Results of QUOT-di compared to LTL3BA.

	QUOT-di better than LTL3BA	**QUOT-di same as LTL3BA**
States	246	748
Edges	246	748

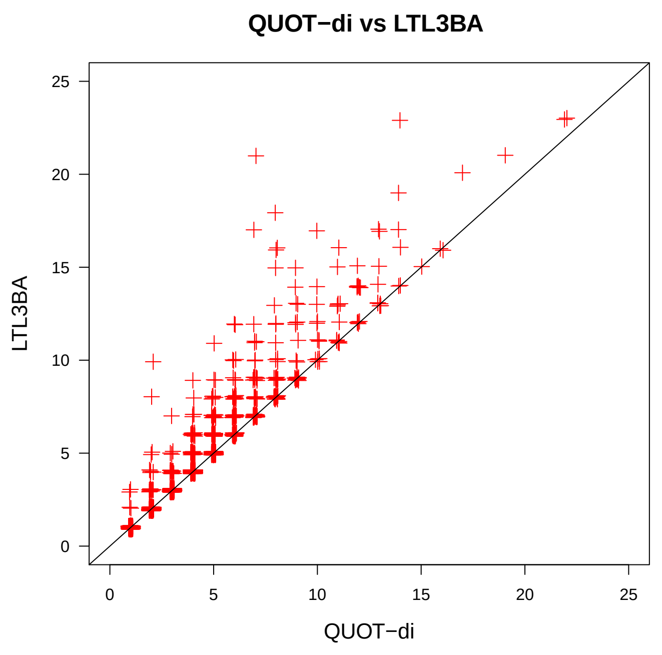


Figure 6.1: Plot showing number of states of automata produced by LTL3BA compared to the result of QUOT-di. Notice that, there are no point under the diagonal, since our implementation works with automata produced by LTL3BA and can only reduce them (i.e. never creates any additional states).

Table 6.5: Results of QUOT-di compared to postprocessing reductions implemented in Spot.

	QUOT-di better than Spot	**Spot same as QUOT-di**	**Spot better than QUOT-di**
States	21	939	34
Edges	29	852	113

6.3.2 Evaluation of PRUNE

In our presentation, let us fix PRUNE-bwdi to denote pruning the transitions of automata with respect to $P(\sqsubseteq^{bw}, \sqsubseteq^{di})$ relation. As proposed in Chapter 3, combining pruning and quotienting (we used only direct forward simulation relation for quotienting) algorithms, we achieved the best results of all presented algorithms and their combinations (see Table 6.6 showing overall results of our tools compared to LTL3BA and Spot).

Table 6.7 shows the number of formulas in which our tools performed better compared to LTL3BA, considering either states or edges of the automata.

Table 6.8 presents number of formulas in which our tools achieved better, equivalent or worse results compared to Spot. For better conception of the number of states of produced automata see Figure 6.2 (each point of a plot represents the number of states produced by Spot compared to our tool).

Table 6.6: Overall results of our tools compared to LTL3BA and post-processing reductions implemented in Spot.

Tool	**Number of states**
LTL3BA	4827
PRUNE-bwdi + QUOT-di	4262
Spot	4255

Table 6.7: Results of our application of both pruning and quotienting algorithms compared to LTL3BA.

	PRUNE-bwdi + QUOT-di better than LTL3BA	**PRUNE-bwdi + QUOT-di same as LTL3BA**
States	250	744
Edges	250	744

We have managed to significantly decrease the number of state space of automata produced by LTL3BA (our algorithms on 250 formulas saved 565 states). Although our tools QUOT and PRUNE did not manage to produce smaller automata that Spot, considering the number of states, our results draw close to results obtained by Spot.

Table 6.8: Results of our application of both pruning and quotienting algorithms compared to reduction of Spot.

	PRUNE-bw-di + QUOT-di better than Spot	**Spot same as PRUNE-bw-di + QUOT-di**	**Spot better than PRUNE-bw-di + QUOT-di**
States	33	923	38
Edges	40	839	115

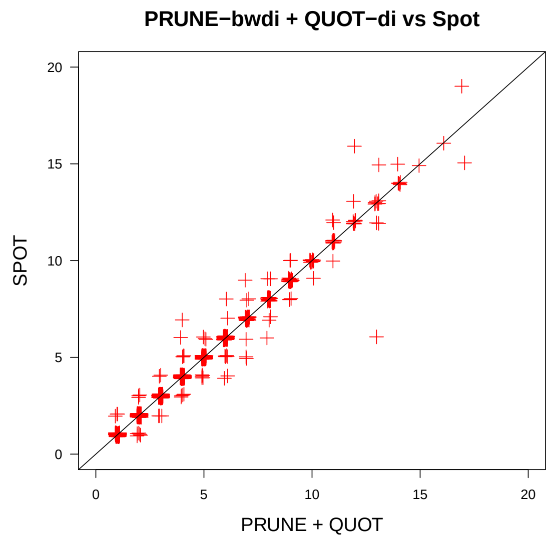


Figure 6.2: Plot showing number of states of automata after reduction of PRUNE-bwdi + QUOT-di compared to reduction of Spot.

7 Conclusion

In this thesis we have extended well known definitions of forward (direct, delayed and fair) and backward simulation relations as well as their k-lookahead variants to work directly with TGBA. These definitions may be implemented in the future and lead to more significant reductions of the automata.

Then, we have implemented direct forward and backward simulations for TGBA using parity games. Since our main intention was to reduce the state space of the automata, we have implemented algorithms quotienting and transition pruning using these simulations for simplifying the automata.

Lastly we experimentally evaluated our implementation against reduction algorithms of Spot. Using our implementation we reached similar reductions compared to Spot.

Bibliography

1. BAIER, Christel; KATOEN, Joost-Pieter; LARSEN, Kim Guldstrand. *Principles of model checking*. MIT press, 2008.
2. CLEMENTE, Lorenzo; MAYR, Richard. Efficient reduction of nondeterministic automata with application to language inclusion testing. *CoRR*. 2017, vol. abs/1711.09946.
3. PARK, David. Concurrency and Automata on Infinite Sequences. In: Springer-Verlag, 1981, vol. 104, pp. 167–183. LNCS.
4. DILL, David L; HU, Alan J; WONG-TOI, Howard. Checking for language inclusion using simulation preorders. In: *International Conference on Computer Aided Verification*. Springer-Verlag, 1991, vol. 575, pp. 255–265. LNCS.
5. ETESSAMI, Kousha; WILKE, Thomas; SCHULLER, Rebecca A. Fair simulation relations, parity games, and state space reduction for Büchi automata. *SIAM Journal on Computing*. 2005, vol. 34, pp. 1159–1175.
6. HENZINGER, Thomas A; KUPFERMAN, Orna; RAJAMANI, Sriram K. Fair simulation. *Information and Computation*. 2002, vol. 173, pp. 64–81.
7. SOMENZI, Fabio; BLOEM, Roderick. Efficient Büchi automata from LTL formulae. In: *International Conference on Computer Aided Verification*. 2000, vol. 1855, pp. 248–263. LNCS.
8. ETESSAMI, Kousha. A Hierarchy of Polynomial-Time Computable Simulations for Automata. In: *International Conference on Concurrency Theory*. Springer-Verlag, 2002, vol. 2421, pp. 131–144. LNCS.
9. AZIZ, Adnan; SINGHAL, Vigyan; BRAYTON, Robert; SWAMY, Gitanjali. In: *Proceedings of the International Conference on Computer Design : VLSI in Computers and Processors*. IEEE Computer Society, 1994, pp. 255–261.
10. BUSTAN, Doron; GRUMBERG, Orna. Simulation-based Minimization. *ACM Transactions on Computational Logic (TOCL)*. 2003, vol. 4, pp. 181–206.

11. DURET-LUTZ, Alexandre; LEWKOWICZ, Alexandre; FAUCHILLE, Amaury; MICHAUD, Thibaud; RENAULT, Etienne; XU, Laurent. Spot 2.0 — a framework for LTL and ω-automata manipulation. In: *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis*. Springer-Verlag, 2016, pp. 122–129. LNCS.

12. FRIEDMANN, Oliver; LANGE, Martin. *The PGSolver Collection of Parity Game Solvers – Version 3*. 2010.

13. BABIAK, Tomáš; BLAHOUDEK, František; DURET-LUTZ, Alexandre; KLEIN, Joachim; KŘETÍNSKÝ, Jan; MÜLLER, David; PARKER, David; STREJČEK, Jan. The Hanoi Omega-Automata Format. In: *Computer Aided Verification*. Springer, 2015, vol. 9206, pp. 479–486. LNCS.

14. BABIAK, Tomás; KRETÍNSKÝ, Mojmír; REHÁK, Vojtech; STREJCEK, Jan. LTL to Büchi Automata Translation: Fast and More Deterministic. In: *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Proceedings*. Springer-Verlag, 2012, vol. 7214, pp. 95–109. LNCS.

15. DURET-LUTZ, Alexandre. LTL Translation Improvements in Spot 1.0. *International Journal on Critical Computer-Based Systems*. 2014, vol. 5, no. 1/2, pp. 31–54.

16. KLOKOČKA, Mikuláš. *Semi-Determinization of Omega-Automata*. 2017.

17. PNUELI, Amir. The Temporal Logic of Programs. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, pp. 46–57. SFCS '77.

18. DURET-LUTZ, Alexandre. Manipulating LTL formulas using Spot 1.0. In: *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*. Springer-Verlag, 2013, vol. 8172, pp. 442–445. Lecture Notes in Computer Science.

A Apendix

A.1 Running QUOT and PRUNE

- install Spot v.2.3+ [1] and PGSolver v.4.1+ [2] and LTL3BA [3] into the `PATH` of your system
- Example: If you want to run QUOT with respect to direct simulation relation on a formula $G(a \mid Fb)$ translated with ltl3ba use:
```
python3 quotienting_reductions.py --simulation=di
-f 'G(a | Fb)' --translator=ltl3ba
```
- Example: If you want to run PRUNE with respect to backward and forward simulation relations on a formula $G(a \mid Fb)$ translated with ltl2tgba use:
```
python3 quotienting_reductions.py --simulation=bw-di
-f 'G(a | Fb)' --translator=ltl2tgba
```

A.2 Content of thesis.zip

I have submitted following electronic attachments:

- `quotienting_reductions.py, pruning_reductions.py` - command line tools released under GNU GPL paper
- `parity_game.py, direct_tgba_parity_game.py,` and `backward_tgba_parity_game.py` - implementation of simulations
- `quotienting.py, pruning.py` - implementation of algorithms
- `ltl_to_test.ltl` - ltl formulas for testing
- scripts and settings for generating data
- `pruning.csv, quotienting.csv` - results of experiments

1. https://spot.lrde.epita.fr/install.html
2. https://github.com/tcsprojects/pgsolver
3. https://sourceforge.net/projects/ltl3ba/