Minimizing Generalized Büchi Automata

Sudeep Juvekar Nir Piterman

MTC-REPORT-2005-001
September 9, 2005

Ecole Polytechnique Fédérale de Lausanne
Faculté Informatique & Communications
CH-1015 Lausanne, Switzerland

Minimizing Generalized Büchi Automata

Sudeep Juvekar[*†] Nir Piterman[‡]

September 9, 2005

Abstract

We consider the problem of minimization of generalized Büchi automata. We show how to extend fair simulation and delayed simulation to the case where the Büchi automaton has multiple acceptance conditions. For fair simulation, we show how to efficiently compute the fair-simulation relation while maintaining the structure of the automaton. We then use the fair-simulation relation to merge states and remove transitions. Our fair-simulation algorithm works in time $O(mn^2k^2)$ where m is the number of transitions, n is the number of states, and k is the number of acceptance sets. For delayed simulation, we extend the existing definition to the case of multiple acceptance condition. We show that our definition can indeed be used for minimization and give an algorithm that computes the delayed-simulation relation. Our delayed-simulation algorithm works in time $O(mn^2k)$. We implemented the two algorithms and report on experimental results.

Keywords: Verification, Simulation, Games, Büchi Automata

[*]Indian Institute of Technology, Bombay, Mumbai 40076, India. Email: sjuvekar@cse.iitb.ac.in

[†]This work was done while this author was a summer intern at EPFL, Lausanne.

[‡]EPFL - I&C - MTC, Lausanne, Switzerland. Email: firstname.lastname@epfl.ch

1 Introduction

The standard approach to LTL model checking is to translate the given specification to a nondeterministic Büchi automaton [Var96]. By now, there are many algorithms that take an LTL formula (or formalisms that extend LTL, cf. [AFF$^+$02, AO04]) and construct an equivalent Büchi automaton [GPVW95, SB00, GO01]. All algorithms produce automata that are exponentially larger than the original LTL formula. In order to improve the efficiency of model checking we would like to produce the minimal possible automata.

Unfortunately, finding the minimal automaton equivalent to a given nondeterministic automaton is computationally expensive. Thus, we usually resort to computationally cheap methods that are not guarranteed to produce the best autoamta.

One such approach is to use simulation [Mil71]. A state t simulates a state s if it has the same observations (or more general) and for every successor s' of s there exists a successor t' of t that simulates s'. If s and t are *simulation equivalent*, i.e., t simulates s and s simulates t, then we can merge s and t to a single state. Similarly, if s has transitions to both t and t' such that t' simulates t, then the transition to t is redundant. However, simulation does not take into account fairness and thus is inadequate for Büchi automata.

There have been several suggestions how to extend simulation to include fairness [DHW91, GL94, HKR97, EWS01]. The simplest of these, is *direct simulation* where in addition to agreement on observations, we demand agreement on fairness [DHW91]. A variant is *reversed simulation* which checks the edges entering a state [SB00]. Both kinds of simulations can be applied to automata on infinite objects [SB00, GBS02]. However, direct and reversed simulation are very restrictive in demanding agreement on fairness.

Henzinger et al. suggested a more relaxed notion of simulation called *fair-simulation* [HKR97]. According to their notion, simulation comes equipped with a strategy. The strategy instructs us which successor t' of t to choose. We demand in addition that by following the strategy a fair computation on one side produces a fair computation on the other side. That is, if we have an infinite sequence of states that starts from the simulated state we can use the strategy to produce an infinite sequence of simulating states. Furthermore, if the first sequence is fair so is the second. Etessami et al. show how to efficiently compute fair simulation for the case of Büchi automata [EWS01]. They show also that fair simulation is too relaxed and cannot be used to merge states. Gurumurthy et al. show that it is still worthwhile to try minimizing with fair simulation [GBS02]. They show that by checking every merge and edge drop for soundness fair simulation can still be used for minimization. They also include an efficient implementation of all the soundness checks whose overall complexity is bounded by the complexity of checking fair simulation.

Etessami et al. provide an intermediate simulation notion called *delayed simulation* [EWS01]. The simulation again includes a strategy but this time whenever one computation visits a fair state the other computation must visit a fair state later. They show that delayed simulation can be used to merge states. That is, if s and t are delayed-simulation equivalent, then the automaton in which s and t are merged into one state is equivalent to the original.

By now most LTL to Büchi conversions use some form of simulation to minimize the size of the automaton. Translation of LTL to Büchi automata results naturally in *generalized Büchi automata*, that is, Büchi automata with multiple acceptance sets (cf. [GPVW95, SB00]). A generalized Büchi automaton with k acceptance conditions and n states can be easily converted to a simple Büchi automaton with nk states (and one acceptance condition) [Cho74]. This conversion is natural and required when explicit state model checking is used [CVWY92]. However, when using symbolic model checking this conversion is undesirable and unnecessary. In the case of generalized Büchi automata intended for symbolic model checking we would like to minimize the automaton without first converting it to a simple Büchi automaton.

The notions of direct and reversed simulation are extended naturally to generalized Büchi automata (though they are even more restrictive in this case) [EH00, SB00]. This is not the case for fair and delayed simulation. The definition of fair simulation does not rely on a specific acceptance condition. Indeed, it applies naturally to generalized Büchi automata. It is not clear, however, how to solve efficiently fair simulation with respect to generalized Büchi automata and how to extend the efficient soundness check. Here we solve these two problems. The efficient computation of fair simulation relies on Jurdzinski's ranking for parity games [Jur00, EWS01]. Based on [KPP05] we are able to extend Jurdzinski's ranking to fair simulation between generalized Büchi automata. The efficient soundness check is extended to the more general ranking as well. The overall complexity of the fair simulation minimization is $O(mn^2k^2)$ where m is the number of transitions of the automaton, n the number of states, and k the number of acceptance sets.

The definition of delayed simulation is tailored specifically for simple Büchi automata. We show how to extend this definition to the case of generalized Büchi automata. We prove that our definition, while seemingly very relaxed, has the power needed in order to be used to minimize generalized Büchi automata. We also show how to efficiently check delayed simulation for this case. The complexity of the delayed simulation minimization is $O(mn^2k)$ where m is the number of transitions of hte automaton, n the number of states, and k the number of acceptance sets.

Finally, we have implemented both these extensions in Wring [SB00]. We tested our implementation on 500 randomly generated LTL formulae and report on the results of this experiment.

2 Preliminaries

2.1 Linear Temporal Logic

We assume a countable set of Boolean variables (propositions) P. LTL formulas are constructed as follows.

$$\varphi ::= p \in P \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi U \varphi$$

As usual we denote $\neg(\neg\varphi \vee \neg\psi)$ by $\varphi \wedge \psi$, $\mathrm{T} U \varphi$ by $\diamondsuit\, \varphi$ and $\neg \diamondsuit \neg\varphi$ by $\square\, \varphi$. A formula that does not include temporal operators is a *Boolean formula.*

A *model* (alternatively, *word*) w for a formula φ is an infinite sequence of truth assignments to propositions. Namely, if $P' \subseteq P$ contains the set of propositions appearing in φ, then a word in $(\{-1, 1\}^{P'})^\omega$ is a model. We denote by w_i the set of propositions in location i, that is $w = w_0 \cdot w_1 \cdots$. We present an inductive definition of when a formula holds in model w at time i.

- For $p \in P$ we have $w, i \models p$ iff $w_i(p) = 1$.
- $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$
- $w, i \models \varphi \vee \psi$ iff $w, i \models \varphi$ or $w, i \models \psi$
- $w, i \models \bigcirc\, \varphi$ iff $w, i + 1 \models \varphi$

- $w, i \models \varphi U \psi$ iff there exists $k \geq i$ such that $w, k \models \psi$ and forall $i \leq j < k$ we have $w, j \models \varphi$

For a formula φ and a position $j \geq 0$ such that $w, j \models \varphi$, we say that φ *holds at position* j of w. If $w, 0 \models \varphi$ we say that φ *holds* on w and denote it by $w \models \varphi$. We denote by $L(\varphi)$ the set of models that satisfy φ.

2.2 Games

A *game* is a tuple $G = \langle V, V_0, V_1, \rho, W \rangle$ where V is the set of locations of the game, V_0 and V_1 are a partition of V to locations of player 0 and player 1 respectively, $\rho \subseteq V \times V$ is the transition relation, and $W \subseteq V^\omega$ is the winning set of G.

A *play* in G is a maximal sequence of locations $\pi = v_0 v_1 \cdots$ such that forall $i \geq 0$ we have $(v_i, v_{i+1}) \in \rho$. A play π is winning for player 0 if $\pi \in W$ or π is finite and the last location in π is in V_1 (i.e., player 1 cannot move from the last location in π). Otherwise, player 1 wins.

A strategy for player 0 is a partial function $f : V^* \times V_0 \to V$ such that whenever $f(\pi v)$ is defined $(v, f(\pi v)) \in \rho$. We say that a play $\pi = v_0 v_1 \cdots$ is f*-conform* if whenever $v_i \in V_0$ we have $v_{i+1} = f(v_0 \cdots v_i)$. The strategy f is *winning from* v if every f-conform play that starts in v is winning for player 0. We say that *player 0 wins* from v if she has a winning strategy. The *winning region* of player 0, is the set of states from which player 0 wins. We denote the winning region of player 0 by W_0. A strategy, winning strategy, win, and winning region are defined dually for player 1. We *solve* a game by computing the winning regions W_0 and W_1. For the kind of games handled by this paper W_0 and W_1 form a partition of V [GH82].

In this paper we are interested in two types of winning conditions. In order to define the winning condition we assume that the game is equipped with a set of propositions $\mathcal{V}$ and a labeling $L : V \to 2^{\mathcal{V}}$ that labels every state v with the set of propositions that are true in it. When a proposition p is in $L(v)$ we say that v satisfies p or that v is a p-state. When $p \notin L(v)$ we say that v satisfies $\neg p$ or that v is a $\neg p$-state. We define the winning conditions by LTL formulas. The first winning condition is built using two sets of propositions $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_l\}$. The *generalized Streett[1]* condition on P and Q is $(\bigwedge_{p \in P} \Box \Diamond p) \to (\bigwedge_{q \in Q} \Box \Diamond q)$. That is, a play is winning for player 0 if for some $p \in P$ the play visits finitely many p-states or for every $q \in Q$ the play visits infinitely many q-states. Notice that when P and Q are singletons then the generalized Streett[1] condition on P and Q is in fact a Streett[1] condition [Str82] or a parity[3] condition [EJ91]. The second winning condition is *generalized response*. This time we are using a set of pairs of propositions $P = \{\langle p_1, q_1 \rangle, \ldots, \langle p_k, q_k \rangle\}$. Player 0 wins the generalized response condition on P if for every pair $\langle p, q \rangle \in P$ player 0 wins $\Box(p \to \Diamond q)$. That is, player 0 wins if for every pair if a play visits a state where p is true then eventually it visits a state where q is true. Notice that this is different from winning $\bigwedge_{\langle p,q \rangle \in P} \Box(p \to \Diamond q)$. In our setting player 0 can work with each of the pairs separately. She does not care about other pairs while playing according to one pair (at least not directly). In the other setting player 0 has to memorize to which pairs she owes a visit. In order to reason about the generalized response condition, we reason about player 1. Player 1 wins if for some pair $\langle p, q \rangle$ she can force a visit to p and remain in $\neg q$ states. Notice that in the case that P is a singleton $\{(p, q)\}$ then this game is exactly the game defined in [EWS01] for delayed simulation. We explain below the motivation for considering these two conditions and in Section 3 show how to solve these two types of games.

2.3 Nondeterministic Büchi Automata

A *nondeterministic Büchi automaton* (or *NBW* for short) is $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$, where $\Sigma = \{-1, 0, 1\}^P$ for some set of propositions P is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\delta \subseteq S \times S$ is a transition relation, $T : S \to \Sigma$ is a labeling function, and $\mathcal{F} = \{F_1, \ldots, F_k\} \subseteq 2^S$ is a set of winning conditions. We call $F \in \mathcal{F}$ a winning set or acceptance condition. For $v \in V$ we denote $\delta(v) = \{w \mid (v, w) \in \delta\}$ and $\delta^-1(v) = \{w \mid (w, v) \in \delta\}$ the set of successors and predecessors of v. A *run* of N is an infinite sequence of states $s_0, s_1, \ldots \in S^\omega$ such that $s_0 \in S_0$ and forall $j \geq 0$ we have $(s_j, s_{j+1}) \in \delta$. For a run $r = s_0, s_1, \ldots$, let $inf(r) = \{s \in S \mid s = s_i$ for infinitely many i's$\}$ be the set of all states occurring infinitely often in the run. A run r is *accepting* if for every $1 \leq i \leq k$ we have $inf(r) \cap F_i \neq \emptyset$. Usually, we distinguish between Büchi automata where $|\mathcal{F}| = 1$ and *generalized Büchi* automata where $|\mathcal{F}| > 1$. In this paper we are interested mainly in generalized Büchi automata. Unless mentioned explicitly, we assume an NBW to have more than one acceptance set. NBW can be classified as strong, weak, or terminal. Intuitively terminal is better than weak which is better than strong. For the definitions of these notions as well as for the explanation as to why one is better from the other we refer the reader to [BRS99].

Given two labels $\sigma, \sigma' \in \{-1, 0, 1\}^P$, we say that σ' abstracts σ ($\sigma \sqsubseteq \sigma'$) if for every $q \in P$ such that $\sigma(q) = 1$ we have $\sigma'(q) = 1$ and for every q such that $\sigma(q) = -1$ we have $\sigma'(q) = -1$. It is simple to see that the abstraction relation is reflexive and transitive. A run $r = s_0, s_1, \ldots$ *induces* an infinite word $w = w_0 w_1 \cdots$ if for every $i \geq 0$ we have that $T(s_i)$ abstracts w_i (notice that a single run may induce many different words). A word w is *accepted* by N if it is induced by some accepting run. The *language* of N, denoted $L(N)$, is the set of words accepted by N.

Given an LTL formula φ we would like to construct an NBW that accepts the set of models of φ. We would like the produced NBW to have a minimal number of states, transitions, and acceptance sets. We use the simulation relation defined in the next subsection to reduce the size of NBW.

Theorem 2.1 [VW94] *For every LTL formula φ, there exists an NBW N_φ with $2^{O(|\varphi|)}$ states, such that $L(N_\varphi) = L(\varphi)$.*

2.4 Simulation

A natural way of comparing automata is by considering language equivalence and language containment. However, these problems are computationally expensive and impractical. In many cases, we resort to using simulation which is an easy to compute precondition to language containment.

Let $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$ and $N' = \langle \Sigma, R, R_0, \eta, T', \mathcal{F}' \rangle$ be two NBW. A relation $H \subseteq S \times R$ is a simulation relation between the states of N and N' if for every $(s, t) \in H$ all the following hold:

- $T(s) \sqsubseteq T'(t)$.
- For every $s' \in S$ such that $(s, s') \in \delta$ there exists a $t' \in R$ such that $(t, t') \in \eta$ and $(s', t') \in H$.

In case that $(s, t) \in H$ we say that t *simulates* s and denote it by $s \leq t$. We say that s and t are *simulation equivalent* if $s \leq t$ and $t \leq s$.

We can compute the largest simulation relation on N and N' by considering the following simple game. The locations of the game are triplets consisting of a pair of states s, t, and a tag from $\{0, 1\}$.

Triplets labeled by 1 additionally supply $T(s) \sqsubseteq T'(t)$. The locations of player 0 are all those locations tagged by 0. Transitions from locations of player 0 correspond to transitions from t and transitions from locations of player 1 correspond to transitions from s. Player 0 tries to prove simulation while player 1 tries to disprove it. Accordingly, Player 1 wins if the play is finite. Otherwise, player 0 wins. It is fairly simple to see that player 0 wins from $(s, t, 1)$ iff $s \leq t$.

Simulation does not take fairness into account. We consider two definitions that extend simulation to include fairness. *Fair-simulation* of Henzinger et al. [HKR97] and *delayed-simulation* of Etessami et al. [EWS01]. Both simulations are defined via games. Given N and N' as above, let $G_{N,N'} = \langle V, V_0, V_1, \rho, W\rangle$ be the *simulation game* with the following components.

- $V = (S \times R \times \{0\}) \cup \{(s, t, 1) \mid s \in S,\ t \in R, \text{ and } T(s) \sqsubseteq T'(t)\}$
- $V_0 = S \times R \times \{0\}$
- $V_1 = \{(s, t, 1) \ :\ s \in S,\ t \in R, \text{ and } T(s) \sqsubseteq T(t)\}$
- $\rho = \{((s, t, 1), (s', t, 0)) \ :\ (s, s') \in \delta\} \cup \{((s, t, 0), (s, t', 1)) \mid (t, t') \in \eta\}$

In order to define the winning conditions we first give labels to the states that depend on the winning conditions of N and N'. Let $\mathcal{F} = \{F_1, \ldots, F_k\}$ and $\mathcal{F}' = \{F'_1, \ldots, F'_l\}$. We define the propositions $\{p_1, \ldots, p_k\}$ and $\{q_1, \ldots, q_l\}$. Proposition p_i is true for a state $(s, t, 1)$ such that $s \in F_i$. Proposition q_i is true for a state $(s, t, 1)$ such that $t \in F'_i$. All propositions are always false on states of player 0.

In order to consider *fair simulation* we consider the generalized Streett[1] game of sets P and Q. It follows that player 0 wins an infinite play if the projection of the play on the first component is fair implies that the projection of the play on the second component is fair. We call this game the *fair-simulation game* or just the *fair game*. If player 0 wins the fair game from state $(s, t, 1)$ then t *fair simulates* s, denoted by $s \leq_f t$. It follows that $H = \{(s, t) \mid (s, t, 1) \in W_0\}$ is the simulation relation. From every pair $(s, t) \in H$ player 0 has a strategy so that the play remains in H and if the projection of an infinite outcome on the first component is fair then so is the projection on the second component. We say that s and t are *fair equivalent*, denoted $s =_f t$ if both $s \leq_f t$ and $t \leq_f s$. Henzinger et al. show that fair simulation implies language containment [HKR97]. Gurumurthy et al. show that the simulation relation between an NBW and itself can be used to reduce the number of states and transitions of the automaton.

In order to consider *delayed simulation* we require that $|\mathcal{F}| = |\mathcal{F}'|$. We consider the generalized response game of the set $P = \{(p_1, q_1), \ldots, (p_k, q_k)\}$. We call this game the *delayed-simulation game* or just the *delayed game*. As before, if player 0 wins from $(s, t, 1)$ then t *delayed simulates* s. It follows, that $H = \{(s, t) \mid (s, t, 1) \in W_0\}$ is the simulation relation. From every pair $(s, t) \in H$ and for every pair $(p_i, q_i) \in P$ player 0 has a strategy so that the play remains in H and if the projection of an infinite outcome on the first component visits p_i then the projection on the second component visits q_i sometime later. By composing strategies we can easily show that H is reflexive and transitive. The notations $\leq_d$ and $=_d$ are defined like for fair simulation. We consider delayed simulation between an automaton and itself. When $|\mathcal{F}| = 1$ our definition is equivalent to the definition in [EWS01]. Etessami et al. study delayed simulation for the case where $|\mathcal{F}| = 1$. They show that delayed simulation is implied by direct simulation (which we do not define here) and it implies fair simulation. These two claims are true also for the general definition above. The first claim is immediate and the second can be proved much like Theorem 5.1. We note that the generalization of delayed simulation to the case of generalized Büchi automata is not straight forward.

The most straight forward extension would be to consider the game with the winning condition $\bigwedge_{(p,q)\in P} \Box(p \rightarrow \Diamond q)$. In Section 5 we show that our definition is strong enough to be used for minimization of NBW. Having different strategies for every one of the pairs is exactly what is needed to establish that minimization with respect to delayed simulation is correct. We do not know how to check efficiently the winning condition $\bigwedge_{(p,q)\in P} \Box(p \rightarrow \Diamond q)$. We believe it is too strong for our purposes.

We use simulation to reduce the number of states and transitions of an automaton. Thus, we usually compute simulation between an NBW and itself. In order to reason about the changes done to an automaton, we have to be able to compute simulation between two different automata.

2.5 Modifications to NBW and Games

Given an automaton $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$ and two states $s, t \in S$ we would like to merge states s and t. We denote by $N(t \leftarrow s)$ the automaton N where state s is merged with state t. That is, we remove state s from the automaton, replace every occurrence of s in S_0, δ, and $\mathcal{F}$ by t. Formally, $N(t \leftarrow s) = \langle \Sigma, S', S'_0, \delta', T, \mathcal{F}' \rangle$ with the following components.

- $S' = S - \{s\}$ - remove s from the set of states.
- If $s \in S_0$ then $S'_0 = (S_0 \cup \{t\}) - \{s\}$, otherwise $S'_0 = S_0$ - replace s by t in the set of initial states if necessary.
- $\delta' = (\delta \cup \{(t, s') \ : \ (s, s') \in \delta\} \cup \{(s', t) \ : \ (s', s) \in \delta\}) - (\delta \cap (\{s\} \times S \cup S \times \{s\}))$ - replace every transition entering or leaving s by the respective transition from / to t.
- For every set $F \in \mathcal{F}$, if $s \in F$ we add $(F \cup \{t\}) - \{s\}$ to $\mathcal{F}'$, otherwise we add F to $\mathcal{F}'$.

In the case where $|\mathcal{F}| = 1$, Etessami et al. use delayed simulation to merge states [EWS01]. They show that if s and t are delayed equivalent then N and $N(t \leftarrow s)$ agree on their languages. Formally, we have the following.

Theorem 2.2 [EWS01] *For an NBW N such that $|\mathcal{F}| = 1$, and s, t such that $s =_d t$ we have $L(N) = L(N(t \leftarrow s))$.*

Etessami et al. show how to encode the delayed game as a parity[3] game and compute the winning region. Then they use the result to reduce the size of NBW. In this paper we extend their result to NBW with many acceptance sets.

Merging two fair-equivalent states may result in automata that are not equivalent [EWS01]. Gurumurthy et al. show that it is still worthwhile to try and merge fair-equivalent states, however, every such merge has to be verified to make sure that it has not changed the automaton [GBS02]. We show how to extend the efficient algorithm for computing fair-simulation between NBW to the case of multiple acceptance set.

Let $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$ and $N' = \langle \Sigma, R, R_0, \eta, L, \mathcal{F}' \rangle$ be two NBW such that $R = S$. Let $\Delta \subseteq S \times S$ be a set of transitions. We define $rem(N, \Delta) = \langle S, S_0, \delta - \Delta, T, \mathcal{F} \rangle$ and $add(N, \Delta) = \langle S, S_0, \delta \cup \Delta, T, \mathcal{F} \rangle$. Let $G_{N,N'} = \langle V, V_0, V_1, \rho, W \rangle$ be the simulation game for N and N'. We define $rem(G_{N,N'}, \Delta) = \langle V, V_0, V_1, \rho', W \rangle$ where $\rho' = \rho - \{((s, t, 0), (s, t', 1)) \mid (t, t') \in \Delta\}$. That is, we restrict the moves of player 0 by removing the moves in Δ. We define $add(G_{N,N'}) = \langle V, V_0, V_1, \rho'', W \rangle$ where

$\rho'' = \rho \cup \{((s,t,1),(s',t,0)) \mid (s,s') \in \Delta\}$. That is, we add options to player 1 by adding the moves in Δ. Intuitively, if we add transitions to an automaton we know that the new automaton simulates the old one. We only check that the old automaton simulates the new one. Dually, when we remove transitions we know that the old automaton simulates the new one. We have to check only the other direction.

Theorem 2.3 [GBS02] *Let N be an NBW and Δ a set of transitions. All the following are true.*

- $G_{N,rem(N,\Delta)} = rem(G_{N,N},\Delta)$.
- $rem(rem(G_{N,N},\Delta),\Delta') = rem(G_{N,N},\Delta \cup \Delta')$.
- $G_{add(N,\Delta),N} = add(G_{N,N},\Delta)$.
- $add(add(G_{N,N},\Delta),\Delta') = add(G_{N,N},\Delta \cup \Delta')$.

The theorem means that it does not matter whether we handle the game graph directly or build it from scratch from the modified automata. Furthermore, a series of transitions can be removed one at a time without rebuilding the game. This theorem is used to efficiently check whether merging of fair equivalent states is allowed [GBS02].

3 Solving Generalized Streett[1] and Generalized Response Games

3.1 Generalized Streett[1] Games

In [EWS01] and [GBS02], fair games are solved using a reduction to parity[3] games. Then Jurdzinski's algorithm for solving parity games is used [Jur00]. We give a short exposition of Jurdzinski's ranking method and the way it is used to solve parity[3] games. We then extend the two to the case of generalized Streett[1].

Let $G = \langle V, V_0, V_1, \rho, W\rangle$ be a game, let $P = \{p\}$ and $Q = \{q\}$ be sets of propositions and L the labeling of the states in V by p and q. We denote by $|p{-}q|$ the number of states in V labeled by p that are not labeled by q. Let $D = [0..|p{-}q|] \cup \{\infty\}$. We order D according to the natural order on $[0..|p{-}q|]$ with ∞ as maximal element. We set $|p{-}q| + 1 = \infty = \infty + 1$. The generalized Streett[1] condition on P and Q is $\Box \Diamond p \rightarrow \Box \Diamond q$. Given a ranking function $r : V \rightarrow D$, we set the function $best(v)$ to be the rank of the minimal successor of v in case $v \in V_0$ and the rank of the maximal successor of V in case $v \in V_1$. A ranking is *good* if for every $v \in V$ all the following hold.

- If $q \in L(v)$ and $best(v) < \infty$ then $r(v) = 1$.
- If $q \notin L(v)$ and $p \in L(v)$ then $r(v) > best(v)$.
- Otherwise $r(v) \geq best(v)$.

Given a good ranking, every state v such that $r(v) < \infty$ is winning for player 0. The ranking also defines a winning strategy for player 0 from these states. All player 0 has to do is choose the successor with minimal rank. Jurdzinski also shows that whenever there exists a winning strategy for player

0 there exists a good ranking such that all the winning states for player 0 have finite rank. We say that such a good ranking is *tight*.

The following algorithm computes a tight ranking if such a ranking exists. For a state $v \in Q$ and a ranking function $r : V \to D$, let $incr_v(i)$ be 0 in case $q \in L(v)$ and $best(v) \neq \infty$, $i+1$ in case $q \notin L(v)$ and $p \in L(v)$, and i otherwise. Let $update(r, v)$ be the ranking r' such that $r'(v') = r(v')$ for $v' \neq v$ and $r'(v) = max\{r(v), incr_v(best(v)\}$. The algorithm that computes the good ranking is:

Let $r := \forall v : r(v) = 0$
While ($\exists v$ s.t. $r(v) \neq update(r, v)(v)$) do
    Let $r := update(r, v)$

We reproduce the efficient implementation of Jurdzinski's algorithm for the case of parity[3] games in Fig. 1 [EWS01]. Here $C(v)$ denotes the number of successors w of v such that $r(w) = best(v)$ and $B(v)$ denotes $best(v)$.

foreach $v \in V$ do
  $B(v) := 0$; $C(v) := |\{w \;:\; (v,w) \in \delta\}|$; $r(v) := 0$;
$L := \{v \in V \mid q \notin L(v)$ and $p \in L(v)\}$;
while $L \neq \emptyset$ do
  let $v \in L$;$L := L \setminus \{v\}$;
  $t := r(v)$;
  $B(v) := best(v)$; $C(v) := cnt(v)$; $r(v) := incr_v(best(v))$;
  $P := \{w \in V \mid (w, v) \in \rho\}$;
  foreach $w \in P$ such that $w \notin L$ do
    if $w \in V_0$ and $t = B(w)$ and $C(w) > 1$ then $C(w) := C(w){-}1$;
    if $w \in V_0$ and $t = B(w)$ and $C(w) = 1$ then $L := L \cup \{w\}$;
    if $w \in V_1$ and $t = B(w)$ then $C(w) := C(w){+}1$;
    if $w \in V_1$ and $t > B(w)$ then $L := L \cup \{w\}$;
  endforeach
endwhile

Figure 1: Efficient implementation of Jurdzinski's algorithm [EWS01]

We extend all the above to the case where P and Q have more than one proposition. Let $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_l\}$. The generalized Streett[1] condition on P and Q is $(\bigwedge_{i=1}^{k} \Box \Diamond p_i) \to (\bigwedge_{j=1}^{l} \Box \Diamond q_i)$. We denote by $|p_i - q_j|$ the number of states in V labeled by p_i that are not labeled by q_j. Let $|j| = max_i\{|p_i - q_j|\}$. We set $D_j = [0..|j|] \times [1..k] \cup \infty$. We set $(r, i) < (r', i')$ if $r < r'$ or $r = r'$ and $i < i'$. We set $(r, i) < \infty$ for all (r, i). This induces a well order on D_j and we define addition in the natural way according to this order. Namely $(r, i)+1$ is $(r, i+1)$ if $i < k$ and $(r+1, 1)$ if $i = k$. As before $(|j|, k) + 1 = \infty = \infty + 1$. Let $j \oplus 1$ denote $(j \ mod \ l)+1$. Consider a set of ranking functions $R = \langle r_1, \ldots, r_l \rangle$ such that $r_j : V \to D_j$. Similar to the case of single acceptance we define $best_j(v)$ to be the rank of the minimal successor of v in case $v \in V_0$ and the maximal successor in case $v \in V_1$. If $q_j \in L(v)$ we take the minimal / maximal according to $r_{j\oplus 1}$, otherwise according to r_j. Formally,

$$best_j(v) = \begin{cases} min_{(v,w)\in\rho}\{r_{j\oplus 1}(w)\} & v \in V_0 \text{ and } q_j \in L(v) \\ min_{(v,w)\in\rho}\{r_j(w)\} & v \in V_0 \text{ and } q_j \notin L(v) \\ max_{(v,w)\in\rho}\{r_{j\oplus 1}(w)\} & v \in V_1 \text{ and } q_j \in L(v) \\ max_{(v,w)\in\rho}\{r_j(w)\} & v \in V_1 \text{ and } q_j \notin L(v) \end{cases}$$

A ranking is *good* if for every $v \in V$ and for every $j \in [1..l]$ all the following hold.

- If $q_j \in L(v)$ and $best_j(v) < \infty$ then $r_j(v) = (0,1)$.
- If $q_j \notin L(v)$, $best_j(v) = (r,i)$, and $p_i \in L(v)$ then $r_j(v) > best_j(v)$.
- Otherwise $r_j(v) \geq best_j(v)$.

Notice that there is a circular dependency between all the rankings through the definition of $best_j(v)$ when $q_j \in L(v)$. We claim that given a good ranking, every state v such that $r_1(v) < \infty$ is winning for player 0. The ranking defines a winning strategy for player 0. More accurately, every ranking ([1..l]) defines a different strategy. Player 0 chooses one such strategy and tries to decrease it. When playing according to strategy j and the play reaches a state v for which $r_j(v) = (0,1)$ and $q_j \in L(v)$ she starts playing according to the $j \oplus 1$ strategy. We say that a ranking is *tight* if it is good and in addition for every winning state v of player 0 we have $r_1(v) < \infty$. The fact that the strategy described above is winning and that whenever there exists a winning strategy for player 0 a tight ranking system exists follows from [KPP05].

A simple generalization of Jurdzinski's algorithm computes the good ranking in our case. For a state $v \in Q$ and a ranking function $r_j : V \rightarrow D_j$, let $incr_v^j(i,o)$ be $(0,1)$ in the case that $q_j \in L(v)$ and $(i,o) < \infty$, $(i,o)+1$ in the case that $q_j \notin L(v)$ and $p_o \in L(v)$, and (i,o) otherwise[1]. Let $update_j(r_j, v)$ be the ranking r'_j such that $r'_j(v') = r_j(v')$ for $v' \neq v$ and $r'_j(v) = max\{r_j(v), incr_v^j(best(v))\}$. The algorithm that computes the good ranking is:

```
Let    R := ∀v, j : r_j(v) = (0,1)
While (∃v, j s.t. r_j(v) ≠ update_j(r_j, v)) do
       Let r_j := update_j(r_j, v)
```

Theorem 3.1 [Jur00, KPP05] *Given a generalized Streett[1] game G, player 0 wins from a location v iff after the lifting algorithm $r_1(v) \neq \infty$.*

In Fig. 2 we generalize the algorithm in Fig. 1 to achieve an efficient implementation of this algorithm. Here $cnt_j(v)$ denotes the number of successors w of v such that $r_j(w) = best_j(v)$.

Theorem 3.2 *We can solve a generalized Streett[1] game in time $O(tgkl)$ where t is the number of transitions, g the number of locations, k the size of P, and l the size of Q.*

Proof: Every iteration of the while loop handles one value $(v,j) \in L$. The loop body increments $r_j(v)$. The maximal number of increments possible is gk. It follows that every location can be handled at most gkl times. Handling a location takes time proportional to the in-degree of this location. □

When we use this algorithm to compute the fair simulation relation (i.e., solve $G_{N,N}$) we get the bounds stated in the following corollary.

Corollary 3.3 *We can compute the fair simulation on an NBW N in time $O(mn^2k^2)$ where m is the number of transitions of N, n is the number of states of N, and k is the size of $\mathcal{F}$.*

We note that in the case that P and Q are singletons then our ranking is exactly the ranking of Jurdzinski for the case of parity[3]. In particular, for such a case the two algorithms are identical.

[1]Notice that in the case that $q \in L(v)$ and $(i,o) = \infty$ then $incr_v^j(i,o) = \infty$

foreach $v \in V$ and $j \in [n]$ do
$\quad B_j(v) := 0;\ C_j(v) := |\{w\ :\ (v,w) \in \delta\}|;\ r_j(v) := (0,1);$
$L := \{(v,j) \in V \mid q_j \notin L(v)$ and $p_1 \in L(v)\};$
while $L \neq \emptyset$ do
$\quad$ let $(v,j) \in L;\ L := L \setminus \{(v,j)\};$
$\quad t := r_j(v);$
$\quad B_j(v) := best_j(v);\ C_j(v) := cnt_j(v);$
$\quad r_j(v) := incr_v^j(best_j(v));$
$\quad P := \{w \in V \mid (w,v) \in \rho\};$
$\quad$ foreach $w \in P$ such that $(w,j) \notin L$ do
$\quad\quad$ if $w \in V_0$ and $t = B_j(w)$ and $C_j(w) > 1$ then $C_j(w)--;$
$\quad\quad$ if $w \in V_0$ and $t = B_j(w)$ and $C_j(w) = 1$ then $L := L \cup \{(w,j)\};$
$\quad\quad$ if $w \in V_1$ and $t = B_j(w)$ then $C_j(w)++;$
$\quad\quad$ if $w \in V_1$ and $t > B_j(w)$ then $L := L \cup \{(w,j)\};$
$\quad$ endforeach
endwhile

Figure 2: Efficient solution of generalized Streett[1] games

3.2 Generalized Response Games

In [EWS01], delayed games with one pair are solved using a reduction to parity[3] games. In order to remember whether the play owes a visit to the acceptance set they add a Boolean flag. As mentioned we prefer to take the view of player 1. This allows us to remove the Boolean flag. The treatment of delayed games becomes completely different from the treatment of fair games.

Let $G = \langle V, V_0, V_1, \rho, W\rangle$ be a delayed game over $P = \{\langle p_1, q_1\rangle, \ldots \langle p_k, q_k\rangle\}$. Let $L : V \rightarrow 2^{\{p_1,\ldots,p_k,q_1,\ldots,q_k\}}$ be the labeling.

The following iterative algorithm computes the winning set in the case that $k = 1$. Let B_0 denote the set of states from which player 1 can force the game to avoid q_1. Let A_0 denote the set of states from which player 1 can force the game to reach p_1 states in B_0. Let B_i denote the set of states from which player 1 can force the game to avoid q_1 or get to A_{i-1}. Let A_i denote the set of states from which player 1 can force the game to reach A_{i-1} or p_1 states in B_i. It follows that for some i we have $A_i = A_{i-1}$. This is the set of winning states of player 1. The set of winning states of player 0 is the complement.

In Fig. 3 we generalize this approach to the case that $k > 1$. Here $\ominus$ and $\oplus$ denote cyclic subtraction and addition in $[1..k]$. The function $back\_reach(X)$ computes the set of states from which player 1 can force the play to reach X. The function $avoid\_set(X, Y)$ computes the set of states from which player 1 can avoid X or reach Y.

Theorem 3.4 *The algorithm in Fig. 3 computes the winning set in generalized response game.*

Proof: We prove that every state in $win(p_i, q_i)$ is winning by induction on the number of iterations in the while loop. In the first iteration $win(p_i, q_i)$ is empty. Suppose that all the locations in $w = win(p_{i\ominus 1}, q_{i\ominus 1})$ are winning for player 1. From every state included in *avoid* player 1 has a strategy to either reach w or avoid q_i locations forever. Every state in *imm_win* is either in w or

foreach $(p,q) \in P$ do
    $win(p,q) = \emptyset$; $old\_win(p,q) = V$;
$i := 1$;
while $win(p_i,q_i) \neq old\_win(p_i,q_i)$ do
    $old\_win(p_i,q_i) := win(p_{i\ominus 1}, q_{i\ominus 1})$;
    $avoid := avoid\_set(q_i, old\_win(p_i,q_i))$;
    $imm\_win := (avoid \cap p_i) \vee old\_win(p_i,q_i)$;
    $win(p_i,q_i) := back\_reach(imm\_win)$;
    $i := i \oplus 1$;
endwhile

Figure 3: Efficient solution of generalized response games

it is a p_i state from which player 1 either avoids q_i forever or reaches w. It follows that the next approximation of $win(p_i,q_i)$ contains winning states for player 1.

We have to show that every winning state is returned by the algorithm. Obviously, the winning region for player 1 is backward closed. That is, if from location v player 1 can force the game to reach winning states then v is winning for player 1. We show that as long as the set computed by the algorithm does not contain all the winning states of player 1 there exists some pair (p,q) and some location v such that player 1 wins from v when considering only the pair (p,q) (or reaching previously computed winning states). Suppose that W is the set of states computed by the algorithm and that W is backward closed. Suppose that there exists $v \notin W$ such that v is winning for player 1. Then there exists some strategy f for player 1 winning from v. We restrict our attention to the f-conform plays from v. We show that there exists some state v' and some pair (p,q) such that player 1 wins $\Diamond(p \wedge \Box(\neg q))$ from v'. Assume to the contrary. Consider the pair (p_1,q_1), either all f conform plays satisfy $\neg p$ or there exists some f conform play that reaches v_1 such that $q_1 \in L(v_1)$. By induction we build a play that is f-conform and for every pair $(p,q) \in P$ it either does not visit p or visits q infinitely often. This play cannot be winning for player 1. We conclude there exists such a pair and such a location and the algorithm computes the winning region for player 1. $\Box$

Theorem 3.5 *We can solve generalized response games in time proportional to tgk where t is the number of transitions, g the number of locations, and k the size of P.*

Proof: We can construct the functions *avoid_set* and *back_reach* so that they work in time proportional to the number of transitions of the game. In order to find one additional location the while loop may need to cycle through all the pairs in P. Hence, the while loop may repeat gk times. The overall run time is $O(tgk)$. $\Box$

When we use this algorithm to compute the delayed simulation relation we get the bounds stated in the following corollary.

Corollary 3.6 *We can compute the delayed simulation on an NBW N in time $O(mn^2k)$ where m is the number of transitions of N, n is the number of states of N, and k is the size of $\mathcal{F}$.*

4 Fair-Simulation Minimization

As mentioned fair simulation cannot be used for merging states. Gurumurthy et al. show that it is still worth while to try to merge using fair simulation provided that all merges are checked [GBS02]. Their algorithm is efficient in the sense that it does not start the fair simulation computation anew for every merge. In a similar way, if there exists a state s such that s has transition to both t and t' where $t \leq_f t'$ they try to remove the transition from s to t. In such a case, they say that t is a *little brother* of t'. Again, they show how to check efficiently all the edge removals. In this section we extend their approach to the case of generalized Büchi automata.

Consider an NBW $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$ and the fair game $G_{N,N} = \langle V, V_0, V_1, \rho, W \rangle$ induced by it. Let $R = \{r_1, \ldots, r_k\}$ be the ranking computed by the algorithm in Section 3. We say that R is the ranking of a game G when R is the result of applying the algorithm in Fig. 2. Given two ranking systems R and R', we say that R is *at least* R' if for every location v and every $1 \leq j \leq k$ we have $r_j(n) \geq r'_j(n)$. The following lemma is stated and proved in [GBS02] for NBW with $|\mathcal{F}| = 1$. The lemma and its proof are identical for the case of NBW where $|\mathcal{F}| > 1$.

Lemma 4.1 *For every set of transitions Δ, the ranking of $rem(G_{N,N}, \Delta)$ is at least the ranking of $G_{N,N}$ and the ranking of $add(G_{N,N}, \Delta)$ is at least the ranking of $G_{N,N}$.*

Intuitively, when we add transitions to player 1, or remove transitions of player 0 the game becomes harder to win. This means that we can essentially check additions of transitions for free. The ranking cannot increase beyond the stated upper bound of $O(mn^2k^2)$. We do pay extra for modifications that do not preserve fair simulation.

When we recognize two states s and t of N as fair equivalent, we try to merge them. In order to check if a merge is possible, we create an automaton with two states with the same predecessors and the same successors. One of such a pair of states can be removed. We add the remaining state to all the acceptance sets in $\mathcal{F}$ in which the other state belonged.

Theorem 4.2 *Let $N = \langle \Sigma, S, S_0, \rho, T, \mathcal{F} \rangle$ be an NBW. Given s and t in S such that $\rho(s) = \rho(t)$ and $\rho^{-1}(s) = \rho^{-1}(t)$ then $L(N) = L(N(t \leftarrow s))$.*

Proof: We show that $N(t \leftarrow s)$ simulates N. Player 1 follows player 0. Whenever player 0 enters either s or t player 1 goes to t.

Similarly, N fair simulates $N(t \leftarrow s)$. The strategy includes a one bit memory. Player 0 follows exactly what player 1 does. The only problem is when player 1 enters the state t. In this case, if the memory bit is set player 0 enters s and flips the bit, if the memory bit is unset player 0 enters t and flips the bit. It is simple to see that player 0 wins. We alternate between s and t because in $N(t \leftarrow s)$ the state t belongs to all the fairness sets which both s and t belong to in N. $\square$

Suppose that we have the game $G_{N,N}$ and the ranking R resulting from running our algorithm. This gives us the fair-simulation relation H. Consider two states s and t such that $s =_f t$. We would like to check whether we can merge s and t. In order to do that we make s and t have the same incoming edges and the same outgoing edges. Formally, let $\Delta = \{(v, w) \mid (v, w) \in \delta \text{ and } |\{v, w\} \cap \{s, t\}| = 1\}$. We consider the game $add(G_{N,N}, \Delta)$. We update the ranking according to the addition. If the new automaton fair simulates the old automaton we conclude the merge to be successful and continue. If the new automaton does not fair simulate the old automaton we conclude the merge to be unsuccessful and revert to the ranking before considering $add(G_{N,N}, \Delta)$. We then proceed to the next pair of candidates to merge. As explained we can now consider the game $add(add(G_{N,N}, \Delta), \Delta')$ where Δ' is the set of transitions that relate to the new pair of states to be merged.

We handle transitions to little brothers in a similar way. This time we use the operator *rem* instead of *add*.

5 Delayed-Simulation Minimization

Delayed simulation as defined for NBW with single acceptance condition can be used for minimization [EWS01]. That is, if $s{=_d}t$ then $L(N(t \leftarrow s)) = L(N)$. Our definition extends delayed simulation for the case of NBW with multiple acceptance conditions. We show that also under our definition $s{=_d}t$ implies $L(N(t \leftarrow s)) = L(N)$. Although our definition is weaker than the straight forward extension of delayed simulation it is strong enough. When considering an infinite fair computation of one automaton, there are infinitely many visits to every one of the acceptance set. We use delayed simulation on every set separately. When the first automaton visits some acceptance set we force a visit to the same acceptance set in the second automaton. Until this goal is achieved we ignore accepting states belonging to other sets. Once this goal is achieved we turn to consider the next acceptance set in cyclic order.

Theorem 5.1 *Given an NBW N and two states s and t such that $s{=_d}t$ then $L(N(t \leftarrow s)) = L(N)$.*

Proof: Clearly, $L(N) \subseteq L(N(t \leftarrow s))$. Consider an accepting run of $L(N)$. Replace every occurrence of s in this run by t. The resulting run is a valid and accepting run of $N(t \leftarrow s)$.

In order to show that $L(N(t \leftarrow s)) \subseteq L(N)$ we use the following. For two sequences of states $r = s_0 s_1 \cdots$ and $r' = p_0 p_1 \cdots$ we write $r{\leq_d}r'$ to denote that forall l we have $s_l{\leq_d}p_l$. In the case that r' is finite we consider $0 \leq l < |r'|$. Consider a run $r = s_0 s_1 \cdots$ of $N(t \leftarrow s)$ such that $s_0 \in F_i$ (i.e., $s_0 \in F_i$ or $s_0 = t$ and $s \in F_i$). We show that for every q_0 such that $s_0{\leq_d}q_0$ we can construct a finite run $r' = q_0 q_1 \cdots q_p$ such that $r{\leq_d}r'$ and $q_p \in F_i$. The construction proceeds in two phases. First, as r is a run of $N(t \leftarrow s)$ it may have transitions to t that are not allowed in N. We convert r to a run $r'' = p_0 p_1 \cdots$ of N such that $r{\leq_d}r''$ and $p_0 \in F_i$. Then, we construct r' such that $r''{\leq_d}r'$. From transitivity of $\leq_d$ we conclude that $r{\leq_d}r'$. We start with the construction of r''. If $s_0 \neq t$ or $s_0 = t$ and $t \in F_i$ then we set $p_0 = s_0$. If $s_0 = t$ and $s \in F_i$ then we set $p_0 = s$. Clearly, $s_0 \leq p_0$ and $p_0 \in F_i$. Consider a prefix $p_0 \cdots p_l$ of r'' such that $r{\leq_d}r''$. Consider the state s_{l+1}. If the transition (s_l, s_{l+1}) is a transition of N then from $s_l{\leq_d}p_l$ we can find p_{l+1} such that $s_{l+1}{\leq_d}p_{l+1}$. If the transition (s_l, s_{l+1}) is not a transition of N, then it must be the transition (s_l, t) in $N(t \leftarrow s)$ and in N the transition (s_l, s) exists. However, as $s_l{\leq_d}p_l$ there exists a successor p_{l+1} of p_l such that $s{\leq_d}p_{l+1}$. As $t{\leq_d}s$ we conclude that $t{\leq_d}p_{l+1}$. This completes the construction of r''. We now turn to r'. As $p_0{=_d}s_0$ it follows that $p_0{\leq_d}q_0$. However, r'' is a run of N so we use directly the strategy of player 0 that wins from $(p_0, q_0, 1)$ in $G_{N,N}$ to produce $r'' = q_0 q_1 \cdots$. As $p_0 \in F_i$, it follows that there exists some p such that $q_p \in F_i$.

We are now ready to complete the proof. Consider an accepting run $r = s_0 \cdots$ of $N(t \leftarrow s)$. We construct an accepting run r' of N such that $r{\leq_d}r'$. Let s_l be the first visit to F_1 in r. We construct $r' = q_0 \cdots q_l$ some run of N such that $r{\leq_d}r'$. Consider now the suffix $\pi = s_l s_{l+1} \cdots$ of r. We use the

Method	States	Trans	Fair	Init	Strong	Weak	Term	Time(sec)
original	26836	104605	1213	3586	537	0	0	3006
fair	26262	100912	1153	3518	535	2	0	6107
delayed	26776	104236	1204	3585	532	3	2	3732
fair+delayed	26070	99672	1141	3518	529	4	4	6666

Table 1: Experimental results for 537 random LTL formulae.

claim to construct a run π' of N such that $\pi \leq_d \pi'$ and π' visits F_1 in location p. We concatenate p' to r'. We now proceed by induction, as r is accepting it visits $F_{i \oplus 1}$ after $|r'|$. We extend r' until a visit to $F_{i \oplus 1}$. It follows that r' is an accepting run of N such that $r \leq_d r'$. Local requirement of abstraction now gives language containment. $\square$

The proof above can be easily modified to show that delayed simulation implies fair simulation (which in turn implies trace containment). As delayed simulation implies fair simulation, every delayed equivalent states are also fair equivalent. This means, that if we try delayed minimization after fair minimization, the only candidates for merging are the states that we try merging but fail to pass the fair simulation test.

6 Experimental Results

In this section, we present experimental results for our algorithms. We have implemented the approach described in Sections 4 and 5 in Wring [SB00]. In order to test the efficiently of our application we tested it on randomly generated LTL formulas.

In Wring, the sequence of optimization steps applied to an NBW starts with a pruning step that removes states that cannot reach a fair cycle. This is followed by a minimization step that includes direct, reverse, fair, and delayed simulation minimization. Finally, there is another pruning step. Obviously, on NBW with multiple acceptance conditions only direct and reverse simulation are applied (in the original version of Wring).

We compare our extension to generalized Büchi automata, with the previously implemented algorithms. We have generated 537 random LTL formulas which produce NBW with more than one acceptance condition (that is, for these formulas direct and reversed simulation leave an NBW with more than one acceptance set). We report on the results of running our application on these formulas. The original version of Wring applies direct and reverse simulation minimization. We compare that to our version running fair minimization, delayed minimization, and both fair and delayed minimization. The results are given in Table 1. For each option we give the total number of states, transitions, initial states, and fairness conditions. We report also on the number of strong, weak and terminal automata. Finally, we include also the total CPU time.

The results above show that our algorithm can improve generalized Büchi automata that have already undergone optimization. In the case of fair simulation the CPU time is considerable. We note that our automata are relatively large. For example, automata resulting from 1000 formulas distributed with Wring have about 5500 states and 10000 transitions (total) [GBS02]. In average our automata are larger by a factor of 10. We note that when combined, delayed and fair simulation may produce a much better result. On one example (not included above), starting from 183 states, each separately hardly reduced the automaton while together they reduced about 90 states. On this example alone, our application requires about 2000 seconds while original Wring requires about 200. Out of 537 NBW, only on 70 NBW our algorithm saves more than 2 states. On these automata it reduced the number of states from 4234 to 3572 and the number of transitions from 17029 to 13077.

7 Conclusions

We have shown how to use delayed and fair simulation to minimize generalized Büchi automata without first converting them to Büchi automata. In order to use fair simulation we extend the explicit ranking computation of Jurdzinski to the case of generalized Streett[1] conditions. Gurumurthy et al. [GBS02] give an efficient implementation that merges fair-equivalent states and checks every such merge for soundness. Similarly, they remove transitions to little brothers and check every such removal. We show that their efficient implementation can be extended to the case of multiple acceptance sets. In order to use delayed simulation we extend the definition of Etessami et al. [EWS01] to the case of multiple acceptance conditions. We show how to compute the delayed simulation relation according to our definition and that this definition allows to merge delayed-equivalent states.

Our definition of delayed simulation to the case of multiple acceptance conditions is not straight forward. Instead of considering all acceptance conditions together we check them one at a time. We wonder whether a similar approach can be applied to Streett automata.

8 Acknowledgments

We thank F. Somenzi for the opportunity to use Wring and for supplying us with the most up to date sources. We thank R. Bloem for his help in acquiring Wring sources and S. Toneta for his help in connecting Wring with lbtt.

References

[AFF+02] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *8th TACAS*, LNCS 2280, pp. 296–211, 2002.

[AO04] Inc. Accellera Organization. Formal semantics of Accellera(c) property specification language. Appendix B of http://www.eda.org/vfv/docs/PSL-v1.1.pdf, 2004.

[BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *11th CAV*, LNCS 1633, pp. 222–235. Springer-Verlag, 1999.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *JCSS*, 8:117–141, 1974.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *FMSD*, 1:275–288, 1992.

[DHW91] D.L. Dill, A.J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *3rd CAV*, LNCS 575, pp. 255–265. Springer-Verlag, 1991.

[EH00] K. Etessami and G. Holzmann. Optimizing büchi automata. In *11th Concur*, LNCS 1877, pp. 153–167. Springer-Verlag, 2000.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd FOCS*, pp. 368–377, 1991.

[EWS01] K. Etessami, Th. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In *28th ICALP*, LNCS 2076, pp. 694–707, 2001.

[GBS02] S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *14th CAV*, LNCS 2404, pp. 610–623. Springer-Verlag, 2002.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *14th STOC*, pp. 60–65, 1982.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM TOPLAS*, 16(3):843–871, 1994.

[GO01] P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In *13th CAV*, LNCS 2102, pp. 53–65. Springer-Verlag, 2001.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pp. 3–18, 1995.

[HKR97] T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *8th Concur*, LNCS 1243, pp. 273–287. Springer-Verlag, 1997.

[Jur00] M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pp. 290–301. Springer-Verlag, 2000.

[KPP05] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. *IC*, 200(1):35–61, 2005.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conference on Artificial Intelligence*, pp. 481–489. 1971.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *12th CAV*, LNCS 1855, pp. 248–263. Springer-Verlag, 2000.

[Str82] R.S. Streett. Propositional dynamic logic of looping and converse. *IC*, 54:121–141, 1982.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pp. 238–266. Springer-Verlag, 1996.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *IC*, 115(1):1–37, 1994.