Minimizing Generalized Büchi Automata

Sudeep Juvekar[1] and Nir Piterman[2]

[1] Indian Institute of Technology Bombay
[2] Ecole Polytechnique Fédéral de Lausanne (EPFL)

Abstract. We consider the problem of minimization of generalized Büchi automata. We extend fair-simulation minimization and delayed-simulation minimization 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 $O(mn^3k^2)$ 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 conditions. 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 $O(mn^3k)$. We implemented the two algorithms and report on experimental results.

1 Introduction

In recent years algorithmic methods for verifying temporal-logic properties of *finite-state* systems have been discovered (cf. [CGP99]). The development of symbolic methods to reason about large state spaces [McM93, BCC$^+$99] have led to the acceptance of model checking in hardware industry [BLM01, CFF$^+$01]. The standard approach to linear temporal logic (LTL) model checking is to translate the given specification to a nondeterministic Büchi automaton [Var96]. By now, there are many algorithms that take an LTL formula (or formalisms that extend LTL, cf. [AFF$^+$02, IEE05]) and construct an equivalent Büchi automaton [GPVW95, SB00, GO01]. The resulting automata may be exponentially larger than the original LTL formula.

To improve model-checking efficiency 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 guaranteed to produce the best automata.

One such approach is to use simulation [Mil71]. A state t simulates a state s if it has the same observations 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. Simulation considers only the transition structure of the automaton and not its acceptance condition. Thus, simulation is inadequate for minimization of Büchi automata.

T. Ball and R.B. Jones (Eds.): CAV 2006, LNCS 4144, pp. 45–58, 2006.

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

Fair-Simulation is a more relaxed notion of simulation [HKR97]. According to this 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 removal for soundness, fair simulation can still be used for minimization. The total complexity of all successful soundness checks 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 an accepting state the other computation must visit an accepting 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 even required) when explicit state model checking is used [CVWY92].[1] However, when using symbolic model checking this conversion is undesirable and unnecessary. Symbolic algorithms for checking emptiness of automata easily handle the generalized Büchi condition without loosing efficiency. On the other hand, converting generalized Büchi to simple Büchi results in model checking a problem that may be k times larger. Counter examples may be significantly longer (even more than a factor of k as the order between the acceptance sets may be important). A similar situation arises when considering complementation of generalized Büchi automata; handling generalized Büchi directly is exponentially

[1] In the case that a simple Büchi automaton is required it would be best to apply first the conversion to a simple Büchi automaton. The conversion from generalized Büchi to a simple Büchi involves the addition of a deterministic part; this implies that simulation on the generalized automaton translates to simulation on the simple automaton. It follows that every modification done using our techniques on the generalized automaton would be done on the simple automaton.

more efficient [KV04]. Also when we use LTL in the context of synthesis, handling the generalized Büchi condition directly produces algorithms that are exponentially better than converting them to simple Büchi [KPV06]. Thus, it is extremely important to be able to further minimize generalized Büchi automata without first converting them to simple Büchi automata.

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. In the case of delayed simulation it is not even clear how to extend the definition to the case of generalized Büchi automata. As mentioned, generalized Büchi automata that are used for symbolic model checking are not converted to simple Büchi automata. As we do not know how to use fair-simulation minimization and delayed-simulation minimization on these automata, we use only the simple optimization techniques. Here we show how the more advanced minimization techniques can be applied to generalized Büchi automata.

In the context of fair simulation, the efficient computation of fair simulation for Büchi automata relies on Jurdziński's ranking for parity games [Jur00, EWS01]. We show how to define a ranking for this type of fair simulation, how to compute this ranking efficiently, and how to check efficiently whether fair-simulation minimization is sound. The overall complexity of the fair-simulation minimization for all successful merges / edge removals is $O(mn^3k^2)$ 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 [EWS01]. 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 $O(mn^3k)$ where m is the number of transitions of the automaton, n the number of states, and k the number of acceptance sets.

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

2 Preliminaries

2.1 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 $\pi = v_0v_1 \cdots$ 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. For an infinite play π we denote by $inf(\pi)$ the set of locations that recur infinitely often in π. Formally, $inf(\pi) = \{v \in V \mid v = v_i \text{ for infinitely many } i\}$.

A *strategy* for player 0 is a partial function $f : V^* \cdot V_0 \to V$ 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 first winning conditions we use two sets $P = \{P_1, \ldots, P_k\}$ and $Q = \{Q_1, \ldots, Q_l\}$ of subsets of the states in G. The *generalized Streett[1]* condition on P and Q is the set of sequences $\pi \in V^\omega$ such that either there exists i such that $inf(\pi) \cap P_i = \emptyset$ or forall j we have $inf(\pi) \cap Q_j \neq \emptyset$. That is, either there exists some set in P that appears finitely often in π, or every set in Q appears infinitely often in π. 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*. We use a set $P = \{\langle P_1, Q_1\rangle, \ldots, \langle P_k, Q_k\rangle\}$ of pairs of subsets of V. In order to define the winning condition we add to the game a counter that ranges over $\{1, \ldots, k\}$. The counter is controlled by player 1 and before every move of player 1 she may change this counter arbitrarily. Player 0 wins the generalized response condition on P if either player 1 changes the counter infinitely often, or if eventually the counter is set to i and along the suffix of the play along which the counter is i every visit to P_i is followed by a visit to Q_i. That is, player 1 chooses a pair $\langle P_i, Q_i\rangle \in P$. While playing according to this pair a visit to P_i should be followed later by a visit to Q_i. At every given point in time player 1 may decide to change the target pair to j and start following $\langle P_j, Q_j\rangle$. If player 1 changes her mind infinitely often she looses. Notice that this is very different from ensuring that for every $j \in \{1, \ldots, k\}$ every visit to P_j is followed by a visit to Q_j. 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). From every state in the winning region of player 0, she has a strategy to win the delayed game with respect to every one of the pairs. This strategy cannot leave the region from which she can win with respect to the other pairs. In order to ensure that forall $j \in \{1, \ldots, k\}$ every visit to P_j is followed by a visit to Q_j, player 0 has to memorize to which pairs she owes a visit. This is not necessary in our case. Notice that in the case that P is a singleton $\{\langle P_1, Q_1\rangle\}$, this game is exactly the game defined in [EWS01] for delayed simulation. We explain below the motivation for these two conditions and in Section 3 show how to solve these two types of games.

2.2 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 set. 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 $inf(r) = \{s \in S \mid s = s_i \text{ for infinitely many } i\text{'s}\}$ 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, all NBW have more than one acceptance set.

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. An *infinite word* over P is an infinite sequence $w = w_0 w_1 \cdots \in \{-1, 1\}^P$ of truth assignments to the propositions in P. 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. We say that two automata are *equivalent* if they have the same language.

Another way to characterize sets of sequences of propositions is by LTL formulas [Pnu77, Eme90]. For every LTL formula φ, there exists an NBW N_φ with $2^{O(|\varphi|)}$ states, such that $L(N_\varphi) = L(\varphi)$ [VW94]. We would like the produced NBW to have a minimal number of states, transitions, and acceptance sets.

2.3 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, an equivalence criterion that implies language containment and is easy to compute.

Simulation does not consider the acceptance condition. We use the extensions *fair simulation* [HKR97] and *delayed simulation* [EWS01] that consider acceptance. Both simulations are defined via games. Consider two NBW $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$ and $N' = \langle \Sigma, R, R_0, \eta, T', \mathcal{F}' \rangle$. Let $G_{N,N'} = \langle V_0 \cup V_1, V_0, V_1, \rho, W \rangle$ be the *simulation game* where (a) $V_0 = S \times R \times \{0\}$ (b) $V_1 = \{(s,t,1) : s \in S,\ t \in R, \text{ and } T(s) \sqsubseteq T'(t)\}$ (c) $\rho = \{((s,t,1),(s',t,0)) : (s,s') \in \delta\} \cup \{((s,t,0),(s,t',1)) \mid (t,t') \in \eta\}$ Note that the game has $O(|S| \cdot |R|)$ states and $O(|\delta| \cdot |R| + |\eta| \cdot |S|)$ transitions. In order to define the winning conditions we define sets of subsets of the locations 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 sets $P_1, \ldots, P_k$ and $Q_1, \ldots, Q_l$. The set P_i contains all locations $(s,t,1)$ such that $s \in F_i$. The set Q_i contains all locations $(s,t,1)$ such that $t \in F'_i$.

In order to consider *fair simulation* we consider the generalized Streett[1] game $G_{N,N'}$ over $P = \{P_1, \ldots, P_k\}$ and $Q = \{Q_1, \ldots, Q_l\}$. 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 $s \leq_f t$. We call $H = \{(s,t) \mid (s,t,1) \in W_0\}$ 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 $t\leq_f s$. Fair simulation implies language containment [HKR97]. Gurumurthy et al. show how to use fair simulation to reduce the number of states and transitions of an NBW where $|\mathcal{F}| = 1$ [GBS02].

In order to consider *delayed simulation* we require that $|\mathcal{F}| = |\mathcal{F}'|$ (i.e., $k = l$). Consider the generalized response game $G_{N,N'}$ over $P = \{\langle P_1, Q_1\rangle, \ldots, \langle P_k, Q_k\rangle\}$. 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. That is, $H = \{(s, t) \mid (s, t, 1) \in W_0\}$ is the simulation relation. From every pair $(s, t) \in H$ and for every pair $\langle P_i, Q_i\rangle \in P$ 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. 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 8.

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 a play winning if for every $\langle P_i, Q_i\rangle \in P$ we have that every visit to P_i is followed by a visit to Q_i. In Section 4 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 correctness of delayed-simulation minimization (as long as the strategies remain in the winning region of player 0).

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

3 Solving Games

3.1 Generalized Streett[1] Games

In [EWS01] and [GBS02], fair games are solved using a reduction to parity[3] games. Then Jurdziński's algorithm for solving parity games is used [Jur00]. Here we generalize this approach to our case.

Let $G = \langle V, \rho\rangle$ be a generalized Streett[1] game over $P = \{P_1, \ldots, P_k\}$ and $Q = \{Q_1, \ldots, Q_l\}$. We define a set of ranking functions $R = \langle r_1, \ldots, r_l\rangle$. The ranking r_i measures what is the minimum over j of the maximal number of visits to P_j until a visit to Q_i is enforced by player 0. If the rank of some state is finite, it means that either for some j we have P_j is visited finitely often or within a finite number of steps player 0 forces a visit to Q_i. We use the ranking to define a winning strategy for player 0 and show that whenever player 0 wins, such a ranking system exists.

We now define formally the range of the ranking functions and the ranking functions themselves. We denote by $|P_i{-}Q_j|$ the number of states in $P_i{-}Q_j$. For $j \in [k]$, let

$|j| = max_i\{|P_i - Q_j|\}$. We set $D_j = ([0..|j|] \times [1..k]) \cup \{\infty\}$. We order D_j according to the lexicographic order with ∞ as maximal element. This induces a well order on D_j and we define increment by one in the natural way according to this order. Namely $(r,i)+1$ is $(r,i+1)$ if $i < k$ and $(r+1,1)$ if $i = k$ and $r < |j|$. We set $(|j|,k)+1 = \infty = \infty + 1$. 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 $r_j : V \to D_j$. 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 $v \in Q_j$ we take the minimal / maximal according to $r_{j\oplus 1}$, otherwise according to r_j. Formally,

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

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

- If $v \in Q_j$ and $best_j(v) < \infty$ then $r_j(v) = (0,1)$.
- If $v \notin Q_j$, $best_j(v) = (r,i)$, and $v \in P_i$ 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 $v \in Q_j$. 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 $v \in Q_j$ she starts playing according to the $j\oplus 1$ strategy. If player 0 changes her strategy infinitely often then forall $1 \leq i \leq l$ we have Q_i is visited infinitely often and player 0 wins. If player 0 eventually plays according to some fixed strategy i, it follows that the rank eventually remains constant (r,i). It follows that P_i is not visited again and player 0 wins.

We say that a ranking is *tight* if it is good and in addition for every winning state v of player 0 we have $r_1(v) < \infty$. In [KPP05] we give a symbolic algorithm for the solution of generalized Streett[1] games. The algorithm consists of a μ-calculus formula that characterizes the set of winning states of player 0. In the full version we prove that the strategy proposed above is winning and use the algorithm of [KPP05] to prove that whenever there exists a winning strategy for player 0 a tight ranking system exists.

If we can produce a tight ranking system, it provides a partition of the states of the game to W_0 and W_1. In order to efficiently compute tight ranking system, we generalize Jurdziński's rank lifting algorithm [Jur00] to our case. For a state $v \in V$ and a ranking function $r_j : V \to D_j$, let $incr_v^j(i,o)$ be $(0,1)$ in the case that $v \in Q_j$ and $(i,o) < \infty$, $(i,o)+1$ in the case that $v \notin Q_j$ and $v \in P_o$, and (i,o) otherwise[2]. 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 *lifting* algorithm that computes the good ranking is:

1 Let $R := \forall v, j : r_j(v) = (0,1)$
2 While $(\exists v, j$ s.t. $r_j(v) \neq update_j(r_j, v))$ do
3 Let $r_j := update_j(r_j, v)$

[2] Notice that in the case that $v \in Q_j$ and $(i,o) = \infty$ then $incr_v^j(i,o) = \infty$.

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

Etessami et al. give an efficient implementation that computes Jurdziński's ranking for parity[3] games [EWS01]. In Fig. 1 we generalize their approach to our ranking.

```
1  foreach v ∈ V and j ∈ [n] do
2      B_j(v) := 0; C_j(v) := |{w : (v,w) ∈ δ}|; r_j(v) := (0,1);
3  L := {(v,j) ∈ V | q_j ∉ L(v) and p_1 ∈ L(v)};
4  while L ≠ ∅ do
5      let (v,j) ∈ L; L := L \ {(v,j)};
6      t := r_j(v);
7      B_j(v) := best_j(v); C_j(v) := cnt_j(v);
8      r_j(v) := incr_v^j(best_j(v));
9      P := {w ∈ V | (w,v) ∈ ρ};
10     foreach w ∈ P such that (w,j) ∉ L do
11      if w ∈ V_0 and t = B_j(w) and C_j(w) > 1 then C_j(w)−−;
12      if w ∈ V_0 and t = B_j(w) and C_j(w) = 1 then L := L ∪ {(w,j)};
13      if w ∈ V_1 and t = B_j(w) then C_j(w)++;
14      if w ∈ V_1 and t > B_j(w) then L := L ∪ {(w,j)};
15     endforeach
16 endwhile
```

Fig. 1. Efficient solution of generalized Streett[1] games

Theorem 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, $k = |P|$, and $l = |Q|$.*

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 1. *We can compute the fair simulation on an NBW N in time proportional to $O(mn^3k^2)$ 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 if P and Q are singletons then our ranking and Jurdziński's ranking for parity[3] are one and the same. In this case the two algorithms are identical.

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. 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 the delayed game over $P = \{\langle P_1, Q_1\rangle, \ldots, \langle P_k, Q_k\rangle\}$. In Fig. 2 we give an algorithm that solves delayed games. Intuitively, player 1 wins

'immediately' from P_i states from which player 1 can avoid Q_i states. Additional winning states are states from which player 1 can force the game to immediate wins or to previously recognized winning states. The algorithm computes the immediate winning according to some pair and the states from which player 1 can force visits to them. Then it proceeds to do the same thing for other pairs until no new winning states for player 1 are discovered. 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 X. The function $avoid\_set(X, Y)$ computes the set of states from which player 1 can avoid X or reach Y.

```
1  foreach ⟨P_i, Q_i⟩ ∈ P do
2      win_i = ∅; old_win_i = V;
3  i := 1;
4  while win_i ≠ old_win_i do
5      old_win_i := win_{i⊖1};
6      avoid := avoid_set(Q_i, old_win_i);
7      imm_win := (avoid ∩ P_i) ∨ old_win_i;
8      win_i := back_reach(imm_win);
9      i := i ⊕ 1;
10 endwhile
```

Fig. 2. Efficient solution of generalized response games

Theorem 3. *The algorithm in Fig. 2 computes W_1 in generalized response games.*

We prove soundness by showing that every state collected by the algorithm has some winning strategy for player 1. We prove completeness by showing that the winning region of player 1 can be partitioned to regions winning by each of the pairs.

Theorem 4. *We can solve generalized response games in time proportional to $O(tgk)$ where t is the number of transitions, g the number of locations, and k the size of P.*

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

Corollary 2. *We can compute the delayed simulation on an NBW N in time proportional to $O(mn^3k)$ 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 Simulation Minimization

4.1 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.
- $\delta' = (\delta \cup \{(t, s') \,:\, (s, s') \in \delta\} \cup \{(s', t) \,:\, (s', s) \in \delta\}) - (\{s\} \times S \cup S \times \{s\})$ - replace transitions entering or leaving s by the respective transition from / to t.
- For every $F \in \mathcal{F}$, if $s \in F$ add $(F \cup \{t\}) - \{s\}$ to $\mathcal{F}'$, otherwise 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 5. [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 that in the case of NBW with one acceptance set, delayed simulation can be used for minimization. We show that this is the case also with our definition and NBW with multiple 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 to the case of NBW with multiple acceptance sets.

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 $rem(N, \Delta) = \langle \Sigma, S, S_0, \delta - \Delta, T, \mathcal{F} \rangle$ and $add(N, \Delta) = \langle \Sigma, S, S_0, \delta \cup \Delta, T, \mathcal{F} \rangle$. 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 6. [GBS02] *Let N be an NBW and Δ a set of transitions. All the following are true.*

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

According to this theorem 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].

4.2 Fair-Simulation Minimization

As mentioned fair simulation cannot be used for merging states. Gurumurthy et al. show that it is still worthwhile 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.

In order to use fair-simulation for minimization we have to check whether the changes done to the automaton are sound, i.e., the new automaton accepts the same language. We change the automaton by adding or removing transitions. In order to check soundness of changes we try to prove that the original automaton and the modified automaton are fair-simulation equivalent. In order to check a series of additions / removals efficiently, we show how to reuse the ranks computed in previous stages.

Consider an NBW $N = \langle \Sigma, S, S_0, \delta, T, \mathcal{F} \rangle$. It induces the fair game $G_{N,N} = \langle V, V_0, V_1, \rho, W \rangle$. 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 rank computation algorithm. 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 $r_j(v) \geq r'_j(v)$. 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 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, if we want to add transitions to the automaton, we add these transitions to the locations of player 1. If we want to remove transitions we remove these transitions from the locations of player 0. When we do that, the game becomes easier for player 1 and harder for player 0. It follows that the ranking in the modified game increases. This means, that if we start from the ranks computed in previous stages and only increase them we are safe. However, the ranks are bounded by values that are not changed by addition / removal of edges. When we measure the amount of work done in all stages of the algorithm (that include several lifting rounds) it cannot be more than $O(mn^3k^2)$ total. Essentially, we do the extra lifting rounds for free.

We would like to be able to merge fair equivalent states of N and check if the resulting automaton is equivalent to the original. We would like to use only addition / removal of transitions to do that. In order to check if a merge is possible, we create an automaton with two states with the same predecessors and the same successors. That is, if $s =_f t$ we add all outgoing / incoming transitions from /to s to t and vice versa. We show now that if we have two states with equivalent incoming / outgoing transitions, one of them can be removed.

Theorem 7. *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 $L(N) = L(N(t \leftarrow s))$.*

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

$\Delta = \{(v,t) \mid (v,s) \in \delta\} \cup \{(v,s) \mid (v,t) \in \delta\} \cup \{(t,v) \mid (s,v) \in \delta\} \cup \{(s,v) \mid (t,v) \in \delta\}$. 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. Little brothers are handled similarly.

4.3 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 sets. 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 consider the next acceptance set in cyclic order.

Theorem 8. *Given an NBW N and states s, t s.t. $s =_d t$ then $L(N(t \leftarrow s)) = L(N)$.*

We show that if there exists a run r of $N(t \leftarrow s)$ that starts with a fair state according to F_i we can find a run segment r' or N that simulates the prefix of r and ends with a fair state from F_i. Given an accepting run of $N(t \leftarrow s)$ every fair set is visited infinitely often. So we create a run of N that visits each fair set in turn. While going for a visit in F_i we ignore other sets in $\mathcal{F}$.

In a similar way we can prove 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.

5 Experimental Results

In this section, we present experimental results for our algorithms. We have implemented the approach described in Section 4 in Wring [SB00]. In order to test the efficiency 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 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. We compare the original version of Wring, which applies direct and reverse minimization, to our version, which adds fair minimization, delayed minimization, or 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, fairness conditions, and CPU time.

Table 1. Experimental results for 537 random LTL formulae

Method	States	Trans	Fair	Init	Time
original	26836	104605	1213	3586	3006
fair	26262	100912	1153	3518	6107

Method	States	Trans	Fair	Init	Time
delayed	26776	104236	1204	3585	3732
fair+delayed	26070	99672	1141	3518	6666

The results above show that our algorithm can improve generalized Büchi automata that have already undergone optimization. We save approximately 3% of the states of the automata, which is comparable to the 1% saved by the original implementation of fair and delayed simulation to NBW with one acceptance set [GBS02]. In the case of fair simulation the CPU time is considerable. We note that our automata are larger by a factor of 10 than the automata used in [GBS02] (where in average an NBW has 55 states and 100 transitions). When combined, delayed and fair simulation may produce better results. 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 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 (about 15% of the states and 25% of the transitions).

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. We thank C. Fritz for spotting an error in an earlier version.

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.

[BCC+99] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *36th DAC*, pp. 317–320. IEEE, 1999.

[BLM01] P. Biesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessors using satisfiability solvers. In *13th CAV*, LNCS 2102, pp. 454–464. 2001.

[CFF+01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *13th CAV*, LNCS 2102, pp. 436–453. Springer-Verlag, 2001.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 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.

[Eme90] E.A. Emerson. Temporal and modal logic. In *Handbook of TCS*, 1990.

[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, 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*, 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 *PSTV*, 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.

[IEE05] IEEE. IEEE standard for property specification language (PSL), October 2005.

[Jur00] M. JurzińSki. 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.

[KPV06] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *18th CAV*, LNCS. Springer-Verlag, 2006.

[KV04] O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th TACAS*, LNCS 2988, pp. 591–606. Springer-Verlag, 2004.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th FOCS*, pages 46–57, 1977.

[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, 1982.

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

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