Fair Simulation Minimization*

Sankar Gurumurthy[1], Roderick Bloem[2], and Fabio Somenzi[1]

[1] University of Colorado at Boulder
{gurumurt,Fabio}@Colorado.EDU
[2] Technical University of Graz
rbloem@ist.tu-graz.ac.at

Abstract. We present an algorithm for the minimization of Büchi automata based on the notion of *fair simulation* introduced in [6]. Unlike direct simulation, fair simulation allows flexibility in the satisfaction of the acceptance conditions, and hence leads to larger relations. However, it is not always possible to remove edges to simulated states or merge simulation-equivalent states without altering the language of the automaton. Solutions proposed in the past consisted in checking sufficient conditions [11, Theorem 3], or resorting to more restrictive notions like *delayed simulation* [5]. By contrast, our algorithm exploits the full power of fair simulation by efficiently checking the correctness of changes to the automaton (both merging of states and removal of edges).

1 Introduction

Optimizing Büchi automata is an important step in efficient model checking for linear-time specification [13, 9]. It is usually cost-effective to invest time in the optimization of the automaton representing the negation of the LTL property because this small automaton is composed with the much larger system to be verified. Any savings obtained on the automaton are therefore amplified by the size of the system. As a side effect of minimizing the automaton, the acceptance conditions may also simplify, thus compounding the advantages of state space reduction. Omega-regular automata are also used to specify properties directly [8]; minimization techniques are applicable to this case as well.

An automaton $\mathcal{A}'$ can replace another automaton $\mathcal{A}$ in model checking if $\mathcal{A}$ and $\mathcal{A}'$ accept the same language. Since checking language equivalence is in general hard, practical approaches [11, 4, 5] resort to various notions of simulations [10] that account for the acceptance conditions of the automata. Simulation is a stronger notion than language containment because the simulating automaton cannot look ahead the moves of the simulated one. On the other hand, several variants of simulation relations can be computed in polynomial time; among them, direct simulation, fair simulation, and delayed simulation.

Direct simulation (BSR-aa in [2]) is the most restrictive of these notions: It requires that the simulating state satisfy all the acceptance conditions of the

* This work was supported in part by SRC contract 2001-TJ-920 and NSF grant CCR-99-71195.

© Springer-Verlag Berlin Heidelberg 2002

simulated one.[1] *Fair simulation*, proposed in [6], relaxes the restriction on the acceptance condition, but it can still be computed in polynomial time. However, its use for minimization of a Büchi automaton is non-trivial because, unlike with direct simulation, one cannot always collapse two states that are fair-simulation equivalent without changing the language accepted by the automaton [5, Proposition 4]. It is also not always possible to remove an edge from state r to state p provided there is a edge from r to q and q fair simulates p. An example is the automaton for $\mathsf{G}\,\mathsf{F}\,a$ and the corresponding game automaton shown in Fig. 4 and discussed in Section 2.

Two approaches have been described in the literature to overcome these limitations of fair simulation. Theorem 3 of [11] says that it is safe to remove the edge described above, provided there is no path in the automaton from q to p. Indeed, the removed edge cannot be used in the accepting run going through q whose existence is guaranteed by the fact that q simulates p.

Etessami et al. [5], on the other hand, have proposed a new notion of simulation called *delayed simulation*, which guarantees that states that are simulation equivalent can be safely merged. Delayed simulation restricts fair simulation by imposing an additional constraint on the non-accepting runs from two related states: If q simulates p, and the i-th state of a run from p is accepting, then there must be a matching run from q such that its j-th state is accepting, and $j \geq i$.

Neither palliative dominates the other. Minimization of the automata family A_n of [5, Proposition 3] is not allowed by [11, Theorem 3] but is possible using delayed simulation, while for the automaton of Fig. 1 the situation is reversed. The word $a\neg a\neg a a^\omega$ has (unique) infinite non-accepting runs from both n_2 and n_3. The run starting from n_3 has an accepting state in first position that is not matched in the run from n_2. Hence, n_2 does not delayed-simulate n_3. However, it does fair-simulate n_3, and [11, Theorem 3] leads to the removal of the edge from n_1 to n_3, effectively eliminating n_3 and n_5 from the automaton.

For the family of automata $\mathcal{A}_n$ exemplified in Fig. 2 for $n = 4$, neither method allows any reduction in the number of states. State n_{ij} delayed-simulates $n_{i'j}$ for $i > i'$, but not vice-versa; hence collapsing is impossible. The automata consist of one SCC, and thus [11, Theorem 3] does not apply either. However, the equivalence class of state n_{ij} according to fair simulation is $[n_{ij}] = \{n_{kj} : 1 \leq k \leq n\}$, and each such equivalence class can be collapsed reducing the number of states from $n^2 + 2$ to $n + 2$.

Another problem with delayed simulation is that it is not safe for edge removal. Consider the automaton of Fig. 3, which accepts the language $\Sigma^* \cdot \{a\}^\omega$. It is not difficult to see that q delayed-simulates p. Indeed, a run moving from p can only take the self-loop, which can be matched from q by going to p.

Even though q is a predecessor of both q and p, one cannot remove the edge (q, p). That is, one cannot use delayed simulation as one would use direct simulation. Since optimization methods based on removal and addition of edges are strictly more powerful than methods based on collapsing simulation equivalent

[1] *Reverse simulation* [11] is a variant of direct simulation that looks at runs reaching a state, instead of runs departing from it.

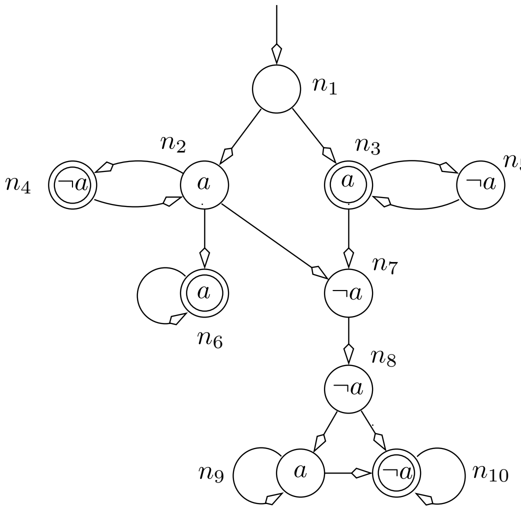


Fig. 1. A sub-optimal automaton that cannot be minimized by delayed simulation

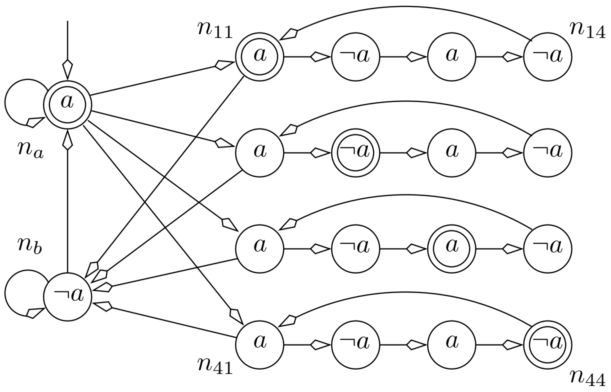


Fig. 2. An automaton that cannot be minimized by either delayed simulation or application of [11, Theorem 3]

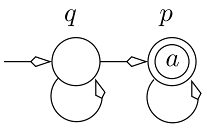


Fig. 3. An automaton showing that delayed simulation is not safe for edge removal

states (collapsing can be achieved by adding and removing edges), this inability limits the optimization potential of methods based on delayed simulation.

The method we propose overcomes the problems seen so far by using fair simulation to select states to be merged and edges to be removed, but checking for their validity before accepting them. The check amounts to verifying whether the modified automaton is still fair-simulation equivalent to the given one. To gain efficiency, we incrementally compute the new simulation relation from the self-simulation relation of the given automaton.

As in [6, 5], the computation of a fair simulation is reduced to the computation of the winning positions for the protagonist in a Streett game [3, 12]. Noting that in the case of non-generalized Büchi automata, the Streett game is equivalent to a parity game with three priorities, Etessami et al. [5] have applied to the problem the recent algorithm of Jurdziński [7], specialized for the case at hand.

Jurdziński's algorithm for parity games assigns a progress measure to the nodes of the game graph by computing the least fixpoint of a monotonic function. If the game graph is changed judiciously, it is therefore possible to update the new progress measure starting the fixpoint computation from the old one. We show how one can produce a locally optimal automaton by a sequence of state mergings or, alternatively, edge removals. Because of the incremental update of the simulation relation, the worst-case complexity of the resulting algorithm is within a factor of k from the complexity of computing the fair simulation once, where k is the number of attempted changes in the sequence that are rejected.

An automaton produced by our procedure is optimal in the sense that if any states are merged at the end of a sequence of mergings, or any edge removed at the end of a sequence of removals, the resulting automaton is not fair simulation equivalent to the old one.

We have implemented the new algorithm for fair simulation minimization in Wring [11], and we report the results of its evaluation in Section 5.

2 Preliminaries

Definition 1. *An* infinite game *is a tuple* $(Q_a, Q_p, \delta, \mathcal{F})$. *Here,* Q_a *and* Q_p *are finite, disjoint sets of antagonist and protagonist states, respectively. We write* $Q = Q_a \cup Q_p$ *for the set of all states. Furthermore,* $\delta \subseteq Q \times Q$ *is the transition relation, and* $\mathcal{F} \subseteq Q^\omega$ *is the acceptance condition.*

An infinite game is played by an antagonist and a protagonist. Starting from a given state $q_0 \in Q$, the antagonist moves from the states in Q_a and the protagonist moves from the states in Q_p. In this manner, the two build a *play* $\rho = q_0, q_1, \ldots$. The game ends if a state with no successors is reached, in which case the protagonist wins the game iff the last state is an antagonist state. If a state without successors is never reached, an infinite play results. In this case, the protagonist wins the game iff $\rho \in \mathcal{F}$. The antagonist wins iff the protagonist does not.

We shall consider Streett acceptance conditions, which depend on $\inf(\rho)$, the set of states that occur infinitely often in a play ρ. A *Streett acceptance condition* is described by a set of pairs of sets of states $\{(E_1, F_1), \dots, (E_n, F_n)\} \subseteq 2^Q \times 2^Q$. A play is winning if for all $1 \leq i \leq n$, either $\inf(\rho) \cap E_i = \emptyset$ or $\inf(\rho) \cap F_i \neq \emptyset$. Of special interest to us are *1-pair Streett conditions*, Streett conditions for which $n = 1$. A *parity condition* is a sequence of sets of states $(F_0, F_1, \dots)$ such that the sets listed form a partition of Q. A play is winning if the lowest index i such that $\inf(\rho) \cap F_i \neq \emptyset$ is even. The 1-pair Streett condition $\{(E, F)\}$ is equivalent to the parity condition $(F, E \setminus F, Q \setminus E \setminus F)$. We shall identify the description of an acceptance condition with the subset of Q^ω that it describes.

A *(memoryless) strategy* for the protagonist is a function $\sigma : Q_p \to Q$ such that for all $q \in Q_p$, $(q, \sigma(q)) \in \delta$. A state $q_0 \in Q$ is *winning* for the protagonist if there is a strategy for the protagonist such that any play $\rho = q_0, q_1, \dots$ for which $q_i \in Q_p$ implies $q_{i+1} = \sigma(q_i)$ is winning for the protagonist. The definitions of a strategy and a winning state for the antagonist are analogous. For parity, and hence for 1-pair Streett games, there is a partition (Q_w, Q_l) of Q such that all states in Q_w are winning for the protagonist, and all states in Q_l are winning for the antagonist. Hence, a state is winning for one player iff it is *losing* for the other. As usual, we shall identify with the protagonist, and simply call a state winning if it is winning for the protagonist.

Definition 2. *A* Büchi automaton *over a finite domain Σ is a tuple $\mathcal{A} = \langle V, V_0, T, C, \Lambda \rangle$, where V is the finite set of states, $V_0 \subseteq V$ is the set of initial states, $T : V \times V$ is the transition relation, $C \subseteq V$ is the acceptance condition, and $\Lambda : V \to 2^\Sigma$ is the labeling function.*

As usual, for a set of states $V' \subseteq V$, we shall write $T(V')$ to mean $\{v' \mid \exists v \in V' : (v, v') \in T\}$, and we shall write $T(v)$ for $T(\{v\})$. A *run* of $\mathcal{A}$ is an infinite sequence $\rho = \rho_0, \rho_1, \dots$ over V, such that $\rho_0 \in V_0$, and for all $i \geq 0$, $\rho_{i+1} \in T(\rho_i)$. A run ρ is *accepting* if $\inf(\rho) \cap C \neq \emptyset$.

The automaton accepts an infinite word $\sigma = \sigma_0, \sigma_1, \dots$ in Σ^ω if there exists an accepting run ρ such that, for all $i \geq 0$, $\sigma_i \in \Lambda(\rho_i)$. The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the subset of Σ^ω accepted by $\mathcal{A}$. We write $\mathcal{A}^v$ for the Büchi automaton $\langle V, \{v\}, T, C, \Lambda \rangle$.

Simulation relations play a central role in this paper. A simulation relation is a relation between nodes of two graphs. If p is simulated by q, from state q we can mimic any run from p without knowing the input string ahead of time. Hence, simulation implies language inclusion. We recapitulate the notions of fair simulation [6] and delayed simulation [5].

Definition 3. *Given Büchi automata*

$$\mathcal{A}_1 = \langle V_1, V_{01}, T_1, C_1, \Lambda_1 \rangle \text{ and } \mathcal{A}_2 = \langle V_2, V_{02}, T_2, C_2, \Lambda_2 \rangle \ ,$$

we define the game automaton $\mathcal{G}_{\mathcal{A}_1,\mathcal{A}_2} = (Q_a, Q_p, \delta, \mathcal{F})$, *where*

$$\begin{aligned}
Q_a &= \{[v_1, v_2] \mid v_1 \in V_1,\ v_2 \in V_2,\ \textit{and}\ \Lambda_1(v_1) \subseteq \Lambda_2(v_2)\},\\
Q_p &= \{(v_1, v_2) \mid v_1 \in V_1\ \textit{and}\ v_2 \in V_2\},\\
\delta &= \{([v_1, v_2], (v_1', v_2)) \mid (v_1, v_1') \in \delta_1, [v_1, v_2] \in Q_a\} \cup\\
&\quad\ \{((v_1, v_2), [v_1, v_2']) \mid (v_2, v_2') \in \delta_2, [v_1, v_2'] \in Q_a\},\\
\mathcal{F} &= \big\{\{(v, w) \mid v \in C_1, w \in V_2\}, \{(v, w) \mid v \in V_1, w \in C_2\})\big\}\ .
\end{aligned}$$

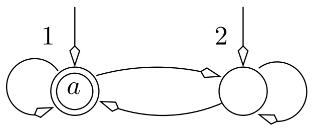
The first subscript in $\mathcal{G}_{\mathcal{A}_1,\mathcal{A}_2}$ identifies the antagonist, while the second identifies the protagonist. The style of brackets is used to differentiate between antagonist and protagonist states: Square brackets denote an antagonist state, while round parentheses indicate a protagonist state. Intuitively, the protagonist tries to prove the simulation relation by matching the moves of the antagonist. The antagonist's task is to find a series of moves that cannot be matched.

State v of automaton $\mathcal{A}$ is *fairly simulated* by v' of automaton $\mathcal{A}'$ if $[v, v']$ is winning in $\mathcal{G}_{\mathcal{A},\mathcal{A}'}$. For different simulation relations we adapt the acceptance criteria of the game graph. We say that v is *delayed simulated* by v' if there is a strategy such that for any play ρ starting from $[v, v']$, if $\rho_i = (w, w')$ with $w \in C_1$, then there is a $j \geq i$ such that $\rho_j = (w, w')$ and $w' \in C_2$.

Example 1. A Büchi automaton $\mathcal{B}$ for the LTL property $\mathsf{G}\,\mathsf{F}\,a$ and the corresponding game automaton $\mathcal{G}_{\mathcal{B},\mathcal{B}}$ are shown in Fig. 4. The set of winning antagonist states is

$$\{[1, 1], [1, 2], [2, 2]\}\ .$$

Therefore, State 2 fair-simulates State 1. However, $\mathcal{B}'$ obtained from $\mathcal{B}$ by removing transition $(2, 1)$, is not simulation equivalent to $\mathcal{B}$. Fig. 5 shows the modified automaton and the game graph required to prove that $\mathcal{B}'$ fair simulates $\mathcal{B}$. (The transition from $(1, 2)$ to $[1, 1]$ is missing.) Notice that, irrespective of the starting



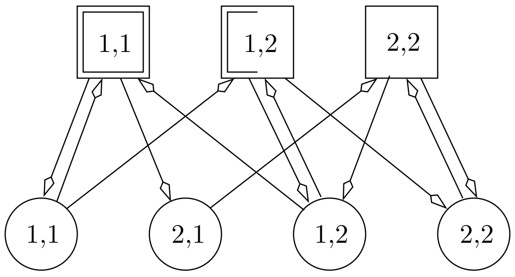


Fig. 4. Automaton for $\mathsf{G}\,\mathsf{F}\,a$ (left) and corresponding game automaton (right). Boxes are antagonist nodes, and circles are protagonist nodes. The label shows the antagonist and protagonist components, respectively. A double border on the left indicates antagonist acceptance; a double border on the right or on the entire node indicates protagonist acceptance

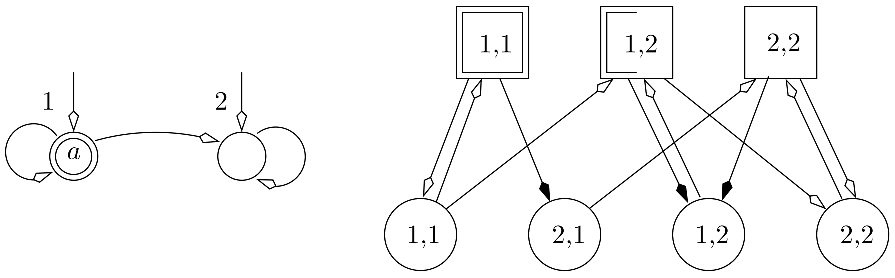


Fig. 5. Automaton $\mathcal{B}'$ (left) and game automaton $\mathcal{G}_{\mathcal{B},\mathcal{B}'}$ (right). Black arrowheads identify the antagonist's winning strategy

state, the antagonist can constrain the play to the states $[1,2]$ and $(1,2)$ of $\mathcal{G}_{\mathcal{B},\mathcal{B}'}$, and therefore win from any initial position. One can verify that removal from $\mathcal{B}$ of the self-loop on State 1 corresponds to removing the transition from $(1,1)$ to $[1,1]$ from $\mathcal{G}_{\mathcal{B},\mathcal{B}}$. Since the protagonist still has a winning strategy from all states, the removal from $\mathcal{B}$ of the self-loop preserves simulation equivalence. □

3 Computing Fair Simulations Incrementally

In this section we shall describe the theory underlying the algorithm. We shall describe how we can use modifications of the game graph to verify proposed changes of a given automaton. Then, we shall quickly review Jurdziński's algorithm for parity games. We shall show that for a series of successful modifications of one kind, we can extend upon the evaluation of the original game graph with no overhead in complexity.

Our use of simulation relations is based on the fact that if v simulates w, then $L(\mathcal{A}^v) \supseteq L(\mathcal{A}^w)$. Hence, given two automata $\mathcal{A}$ and $\mathcal{A}'$, for the language of $\mathcal{A}'$ to be included in that of $\mathcal{A}$, it is sufficient (though not necessary) that for all initial states v_0' of $\mathcal{A}'$ there is an initial state v_0 of $\mathcal{A}$ that fairly simulates v_0'. In this case, we say that $\mathcal{A}$ *fairly simulates* $\mathcal{A}'$. We consider simulation instead of language equivalence since computing the latter is prohibitively expensive.

Given a Büchi automaton $\mathcal{A} = \langle V, V_0, T, C, \Lambda\rangle$, we build the game graph $\mathcal{G}_{\mathcal{A},\mathcal{A}}$. We consider edges of $\mathcal{A}$ for removal or addition , and we check correctness of the proposed change by a modification of the game graph.

Definition 4. *Let $\mathcal{A} = \langle V, V_0, T, C, \Lambda\rangle$ and $\mathcal{A}'$ be Büchi automata with the same state space, and let $\Delta T \subseteq V \times V$ be a set of transitions. We define* $\mathrm{rem}(\mathcal{A}, \Delta T) = \langle V, V_0, T \setminus \Delta T, C, \Lambda\rangle$. *For an infinite game $\mathcal{G}_{\mathcal{A},\mathcal{A}'} = (Q_a, Q_p, \delta, \mathcal{F})$,* $\mathrm{rem}(\mathcal{G}_{\mathcal{A},\mathcal{A}'}, \Delta T)$ *is the game graph $(Q_a, Q_p, \delta', \mathcal{F})$, where*

$$\delta' = \delta \setminus \{((v_1, v), [v_1, v']) \mid (v_1, v) \in Q_p, [v_1, v'] \in Q_a, (v, v') \in \Delta T\} \ .$$

Similarly, $\mathrm{add}(\mathcal{A}, \Delta T) = \langle V, V_0, T \cup \Delta T, C, \Lambda\rangle$*, and* $\mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}'}, \Delta T)$ *is the game graph* $(Q_a, Q_p, \delta', \mathcal{F})$*, where*

$$\delta' = \delta \cup \{([v, v_2], (v', v_2)) \mid [v, v_2] \in Q_a, (v', v_2) \in Q_p, (v, v') \in \Delta T\} \ .$$

Intuitively, if we add transitions to the automaton, we know that the new automaton simulates the old one. We have to check whether simulation holds in the opposite direction. To do this, we add transitions to the antagonist states in the game, reflecting the new edges in the modified automaton. The following theorem is easily proven.

Theorem 1. *Let $\mathcal{A}$ be a Büchi automaton, and let $\Delta T \subseteq V \times V$ be a set of transitions. We have* $\mathcal{G}_{\mathcal{A},\mathrm{rem}(\mathcal{A},\Delta T)} = \mathrm{rem}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T)$*. Furthermore,*

$$\mathrm{rem}(\mathrm{rem}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T), \Delta T')) = \mathrm{rem}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T \cup \Delta T') \ .$$

Similarly, $\mathcal{G}_{\mathrm{add}(\mathcal{A},\Delta T),\mathcal{A}} = \mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T)$*. Furthermore,*

$$\mathrm{add}(\mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T), \Delta T')) = \mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T \cup \Delta T') \ .$$

This theorem says that we can obtain the game graph of the original automaton $\mathcal{A}$ and a modified version $\mathcal{A}'$, that is obtained by adding or removing edges, by modifying the game graph. Furthermore, it states that edges can be deleted a few at a time, or all at once. Hence, we can modify the game graph instead of building it from scratch. After a recapitulation of Jurdziński's algorithm, we shall show that this means that we can efficiently reuse information.

We use Jurdziński's algorithm [7] for parity games as specialized in [5] to compute the simulation relation. We can use this algorithm because 1-pair Streett conditions correspond to length-3 parity conditions. Let n_1 be the number of protagonist states (v_1, v_2) such that v_1 satisfies the fairness constraint of $\mathcal{A}_1$, but v_2 does not satisfy the fairness constraint of $\mathcal{A}_2$, i.e., $n_1 = |\{(v_1, v_2) \in Q \mid v_1 \in C_1, v_2 \notin C_2\}|$. Jurdziński's algorithm for three priorities computes a *progress measure* on the states of the game automaton: $r : Q \to \{0, \dots, n_1\} \cup \{\infty\}$, such that $r(q) \neq \infty$ iff n_1 is a winning state. The measure is computed as a least fixpoint of the following lifting function.

$$\mathrm{lift}(r, q) = \lambda p \, . \begin{cases} \mathrm{update}(r, q) & \text{if } p = q, \\ r(p) & \text{otherwise.} \end{cases}$$

Here, $\mathrm{update}(r, q)$ is a function that is monotonic in the measures of the successors of q, and hence lift is (pointwise) monotonic. Because of monotonicity, the measure can be updated at most $n_1 + 1$ times per node. Combined with the fact that $\mathrm{update}(r, q)$ can be performed in time proportional to the number of successors of q, this implies that the complexity of the algorithm is $O(|\delta| \cdot n_1) = O(|Q|^2 \cdot n_1)$.

To be more precise, if $q \in Q_a$, then $\mathrm{update}(r, q)$ is monotonic in $\max\{r(p) \mid (q, p) \in \delta\}$, and if $q \in Q_p$, then $\mathrm{update}(r, q)$ is monotonic in $\min\{r(p) \mid (q, p) \in \delta\}$.

It should be noted that the measure of an antagonist (protagonist) node without successors is 0 (∞).

We can check the validity of a proposed addition of an edge to, or removal of an edge from $\mathcal{A}$ by constructing the game graph $\mathcal{G}_{\mathcal{A},\mathcal{A}}$, and modifying it as described above. If for every initial state v there is an initial state w such that $[v, w]$ is winning, then the proposed modification does not change the language of the graph. In the naive implementation, this implies that for every modification Jurdziński's algorithm has to be rerun. We shall now show the modification of the game graph allows us to quickly evaluate a proposed modification.

Lemma 1. *If transitions from protagonist states are removed from the game graph, the measure of a node cannot decrease. Similarly, if transitions from antagonist states are added, the measure cannot decrease.*

Proof. Since the measure of a protagonist node is a monotonic function of the minimum of the measures of its successors, removing one successor cannot decrease the measure. Similarly for antagonist nodes. □

Intuitively, if we add transitions from antagonist states, or remove transitions from protagonist states, the game becomes harder to win. This result has the advantage that for a given sequence of additions of transitions from antagonist states, the correctness of all additions can be checked within the same complexity bound that holds for the original algorithm: $O(|Q|^2 \cdot n_1)$, assuming that all such modifications are legal.

Given a sequence of additions or removals of sets of edges, there may be candidates that change the language of the automaton. Work done evaluating such modifications is lost, and hence the complexity of validating such a set of modifications is $O(|Q|^2 \cdot n_1 \cdot k)$, where k the number of failed modifications. Clearly, $k = O(|Q|^2)$.

To merge fair-simulation equivalent states, the algorithm will try to change the graph in such a way as to create states with the same predecessors and successors. One of such a pair of states can be dropped, assuming that either the remaining state is accepting or the dropped state is not.

Theorem 2. *Let $\mathcal{A} = \langle V, V_0, T, C, \Lambda\rangle$ be a Büchi automaton such that there are $v, v' \in V$ with $T(v) = T(v')$, $T^{-1}(v) = T^{-1}(v')$, $\Lambda(v) \subseteq \Lambda(v')$ and $v \in C$ implies $v' \in C$. Then, $L(\mathcal{A}) = L(\mathcal{A}')$, where $V' = V \setminus \{v\}$, $V_0' = V_0 \cup \{v'\} \setminus \{v\}$ if $v \in V_0$ and $V_0 = V_0$ otherwise, $T' = T \cap (V' \times V')$, and $C' = C \setminus \{v\}$.*

We do not have to consider changes to the graph more than once. This is another consequence of monotonicity of the measure. If $\mathrm{add}(\mathcal{A}, \Delta T)$ is not simulated by $\mathcal{A}$, then $\mathrm{add}(\mathrm{add}(\mathcal{A}, \Delta T'), \Delta T)$ is not simulated by $\mathrm{add}(\mathcal{A}, \Delta T')$. This follows because the measure of the game graph $\mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T')$ is not smaller than that of $G_{\mathcal{A},\mathcal{A}}$, and hence the measure of $\mathrm{add}(\mathrm{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T'), \Delta T)$ is not smaller than that of $\mathrm{add}(G_{\mathcal{A},\mathcal{A}}, \Delta T'), \Delta T)$. Recalling that a state is winning if its measure is smaller than ∞, it is clear that the latter game graph does not have more winning positions, and hence does not define a greater simulation relation.

A similar observation can be made for removing edges.

4 A Fair Minimization Algorithm

In this section we describe a method to minimize a Büchi automaton using the game graph. The proposed method uses the fair-simulation relation to find states that are candidates for merger and edges that are candidates for removal. By manipulating the game graph, the algorithm checks whether the proposed merger of two states or removal of an edge is correct, i.e., whether it results in a simulation-equivalent automaton. Because the simulation relation does not have to be recomputed every time from scratch, this method is efficient. Furthermore, it is more effective than known methods that can be applied statically, as discussed in Section 1.

The algorithm proceeds in two phases: First it tries to merge equivalent states, and then it tries to remove redundant edges.

The algorithm attempts to merge two fair-simulation equivalent states v and w by adding edges such that the successors of v become successors of w and vice-versa, and likewise for predecessors. Validation of the correctness of a modification is performed as described in Section 3.

In detail, we construct $\mathcal{G}_{\mathcal{A},\mathcal{A}}$ and compute the progress measure using Jurdziński's algorithm. Then, we pick a pair of states v, w that we wish to merge. We construct $\mathcal{G}' = \text{add}(\mathcal{G}_{\mathcal{A},\mathcal{A}}, \Delta T)$, where $\Delta T = (\{v, w\} \times T(\{v, w\})) \cup (T^{-1}(\{v, w\}) \times \{v, w\})$. We then update the progress measure, thereby computing the simulation relation between $\mathcal{A}$ and $\mathcal{A}'$, where $\mathcal{A}' = \text{add}(\mathcal{A}, \Delta T)$. If we find that the $\mathcal{A}$ still simulates $\mathcal{A}'$, then the merge is accepted, a new pair is proposed for merger, $\mathcal{G}_{\mathcal{A}'',\mathcal{A}}$ is computed, etc.

As discussed, pairs of simulation-equivalent states are picked as candidates for merger. Though [5] shows that fair-simulation equivalence is no guarantee for mergeability (and in fact the number of equivalent states that cannot be merged can be in the order of $|Q|^2$), the chances that two equivalent states are mergeable are quite high in practice. The number of rejected modifications is thus limited by the number of pairs of simulation-equivalent states that cannot be merged.

The second stage of the algorithm proceeds likewise to attempt to remove edges. The candidates for removal are edges (u, v) for which there is a state w that simulates v and an edge (u, w).

In Stage 1, if we find a pair of states (v, w) such that v and w are delayed-simulation equivalent, the merge is guaranteed to succeed. Similarly in Stage 2, if w direct-simulates v. Each stage of the algorithm leads to a graph that is optimal, in that no candidate for removal has to be checked again.

Backtracking can be implemented efficiently by using time stamps. Every assignment of a measure to a state receives a time stamp—initially 0. Before the measure is updated, the time stamp is increased. When $r(v)$ is changed, its time stamp is checked. If it is not the current timestamp, the value is saved. If one needs to backtrack, one looks for all the nodes such that $r(v)$ has the most recent timestamp. One replaces these values with the old values, and the old time stamp. Then, one decreases the current time stamp to the previous value. A list of nodes with new values is kept, so that the cost of undoing the changes is proportional to the extent of the changes, and not the size of the game graph.

Likewise, when an arc is added or removed from the game graph, a change record with the current time stamp is appended to the list. As pointed out in [7], another way to improve performance is to exploit the decomposition of the game graph into SCCs, processing them in reverse topological order.

5 Experiments

In this section we present preliminary experimental results for our algorithm. We have implemented the approach described in Section 4 in Wring [11], and compared it to other methods for the minimization of Büchi automata. As test cases we have used 1000 automata generated by translation of as many random LTL formulae distributed with Wring [11, Table 2]. In addition, we report results for 23 hard-to-minimize cases, partly derived from examples found in the literature [6, 11, 5].

In Wring, the sequence of optimization steps applied to a Büchi automaton starts with a pruning step (P) that removes states that cannot reach a fair cycle, and simplifies the acceptance conditions. This is followed by a pass of peep-hole minimization (M), which is of limited power, but is fast, and includes transformations that simulation-based methods cannot accomplish. After that, direct (D) and reverse (R) simulations are used to merge states and remove arcs. Finally, a second pruning step is applied. We refer to this standard sequence by the following abbreviation: PMDRP.

We compare this standard optimization sequence to others that use in addition or alternative to the other steps, fair simulation minimization (F), and delayed simulation minimization (d). Since neither of these two alternative methods can deal with generalized Büchi automata, they are applied only to the cases in which there is exactly one fairness condition. (For the 1000 automata of Table 1, this happens 465 times.) The notation F/D designates the application of fair simulation minimization to automata with one acceptance condition, and direct simulation to the other automata. Likewise for d/D.

The results for the automata derived from LTL formulae are summarized in Table 1. For each method, we give the total number of states, transitions,

Table 1. Experimental results for 1000 automata derived from LTL formulae

method	states	trans	fair	init	weak	term	time
PMDRP	5620	9973	487	1584	400	523	125.5
PMDRFP	5581	9827	487	1560	396	529	158.0
PMDRdP	5618	9980	488	1584	400	523	160.1
PMF/DRP	5587	9869	488	1556	395	529	162.5
PMd/DRP	5618	9980	488	1584	400	523	159.9
PDP	5704	10722	489	1587	396	523	114.7
PF/DP	5688	10625	489	1561	392	529	153.5
Pd/DP	5910	11522	488	1626	383	520	155.0

fairness conditions, initial states, and we report how many automata were weak or terminal [1]. Finally, we include the total CPU time. In comparing the numbers it should be kept in mind that the results are affected by a small noise component, since they depend on the order in which operations are attempted, and this order is affected by the addresses in memory of the data structure.

The result for PMDRFP shows that our algorithm can still improve automata that have undergone extensive optimization. The CPU times increase w.r.t. PMDRP, but remain quite acceptable. In spite of having to check each modification of the automata, fair simulation minimization is about as fast as delayed simulation.

There are several reasons for this. First, the time to build the game graph dominates the time to find the winning positions, and delayed simulation produces larger game graphs (up to twice the size, and about 10% larger on average) in which each state has four components instead of three. Second, most modification attempted by the fair simulation algorithm do not change the language of the given automaton (78% in our experiments); hence, as discussed in Section 3, their cost is low.

Finally, Jurdziński's algorithm converges faster for the fair simulation game when the delayed simulation relation is a proper subset of the fair simulation relation.

The shorter optimization sequences are meant to compare fair simulation minimization to delayed simulation minimization without too much interference from the other techniques. In particular, one can see from comparing PF/DP and Pd/DP that removal of transitions, as opposed to merging of simulation equivalent states, does play a significant role in reducing the automata. Indeed, direct simulation, which can be safely used for that purpose, does better than delayed simulation.

Finally, Table 2 summarizes the results for the hard-to-minimize automata.

Table 2. Experimental results for 23 hard-to-minimize automata

method	states	trans	fair	init	weak	term	time
PMDRP	131	219	21	29	3	2	0.49
PMDRFP	106	165	21	25	4	2	1.05
PMDRdP	128	212	21	29	3	2	1.22
PMF/DRP	106	167	21	25	4	2	1.38
PMd/DRP	138	229	21	29	3	2	1.40
PDP	133	222	22	30	3	2	0.47
PF/DP	106	168	21	25	4	2	2.60
Pd/DP	130	217	21	30	3	2	3.42

6 Conclusions

We have presented an algorithm for the minimization of Büchi automata based on fair simulation. We have shown that existing approaches are limited in their optimization power, and that our new algorithms can remove more redundancies than the other approaches based on simulation relations. We have presented preliminary experimental results showing that fair simulation minimization improves results even when applied after an extensive battery of optimization techniques like the one implemented in Wring [11]. Our implementation is still experimental, and we expect greater efficiency as it matures, but the CPU times are already quite reasonable.

The approach of checking the validity of moves by updating the solution of a game incrementally can be applied to other notions of simulation that do not allow safe collapsing of states or removal of edges. In particular, we plan to apply it to a relaxed versions of reverse simulation. We also plan to address the open issue of extending our approach to generalized Büchi automata, that is, to automata with multiple acceptance conditions.

Acknowledgment

We thank Kavita Ravi for many insightful observations on simulation minimization.

References

[1] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 222–235. Springer-Verlag, Berlin, 1999. LNCS 1633. 621

[2] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In K. G. Larsen and A. Skou, editors, *Third Workshop on Computer Aided Verification (CAV'91)*, pages 255–265. Springer, Berlin, July 1991. LNCS 575. 610

[3] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, October 1991. 613

[4] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *Proc. 11th International Conference on Concurrency Theory (CONCUR2000)*, pages 153–167. Springer, 2000. LNCS 1877. 610

[5] K. Etessami, T. Wilke, and A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming: 28th International Colloquium*, pages 694–707, Crete, Greece, July 2001. Springer. LNCS 2076. 610, 611, 613, 614, 617, 619, 620

[6] T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'97)*, pages 273–287. Springer-Verlag, 1997. LNCS 1243. 610, 611, 613, 614, 620

[7] M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 290–301, Lille, France, February 2000. Springer. LNCS 1770. 613, 617, 620

[8] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994. 610

[9] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985. 610

[10] R. Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, NJ, 1989. 610

[11] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In E. A. Emerson and A. P. Sistla, editors, *Twelfth Conference on Computer Aided Verification (CAV'00)*, pages 248–263. Springer-Verlag, Berlin, July 2000. LNCS 1855. 610, 611, 612, 613, 620, 622

[12] W. Thomas. On the synthesis of strategies in infinite games. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science*, pages 1–13. Springer-Verlag, 1995. LNCS 900. 613

[13] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1983. 610