Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata

Kousha Etessami[1], Thomas Wilke[2], and Rebecca A. Schuller[3]

[1] Bell Labs, Murray Hill, NJ
kousha@research.bell-labs.com
[2] Christian Albrecht University, 24098 Kiel, Germany
wilke@ti.informatik.uni-kiel.de
[3] Cornell University, Ithaca, NY
reba@math.cornell.edu

Abstract. We give efficient algorithms, beating or matching optimal known bounds, for computing a variety of simulation relations on the state space of a Büchi automaton. Our algorithms are derived via a unified and simple parity-game framework. This framework incorporates previously studied notions like *fair* and *direct* simulation, but our main motivation is state space reduction, and for this purpose we introduce a new natural notion of simulation, called *delayed* simulation. We show that, unlike fair simulation, delayed simulation preserves the automaton language upon quotienting, and that it allows substantially better state reduction than direct simulation. We use the parity-game approach, based on a recent algorithm by Jurdzinski, to efficiently compute all the above simulation relations. In particular, we obtain an $O(mn^3)$-time and $O(mn)$-space algorithm for computing both the delayed and fair simulation relations. The best prior algorithm for fair simulation requires time $O(n^6)$ ([HKR97]).
Our framework also allows one to compute bisimulations efficiently: we compute the fair bisimulation relation in $O(mn^3)$ time and $O(mn)$ space, whereas the best prior algorithm for fair bisimulation requires time $O(n^{10})$ ([HR00]).

1 Introduction

There are at least two distinct purposes for which it is useful to compute simulation relationships between the states of automata: (1) to efficiently establish language containment among nondeterministic automata; and (2) to reduce the state space of an automaton by obtaining its quotient with respect to the equivalence relation underlying the simulation preorder.

For state machines without acceptance conditions, there is a well-understood notion of simulation with a long history (see, e.g., [Mil89,HHK95]). For ω-automata, where acceptance (fairness) conditions are present, there are a variety of different simulation notions (see, e.g.,[HKR97],[GL94]). At a minimum, for such a simulation to be of use for purpose (1), it must have the following property:

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 694–707, 2001.

(*) whenever state q' "simulates" state q the language of the automaton with start state q' contains the language of the automaton with start state q.

This property alone however is not, as we will see in Sect. 5, sufficient to assure usefulness for purpose (2), which requires the following stronger property:

(**) the "simulation quotient" preserves the language of the automaton.

We will state precisely what is meant by a simulation quotient later.

In [HKR97] a number of the different simulation notions for ω-automata were studied using a game-theoretic framework. The authors also introduced a new natural notion of simulation, titled *fair* simulation. They showed how to compute fair simulations for both Büchi and, more generally, Streett automata. For Büchi automata, their algorithm requires $O(n^6)$ time to determine, for one pair of states (q, q'), whether q' fairly simulates q.[1] Their algorithm relies on an algorithm for tree automaton emptiness testing developed in [KV98]. In this paper, we present a new comparatively simple algorithm for Büchi automata. Our algorithm reduces the problem to a parity game computation, for which we use a recent elegant algorithm by Jurdzinski, [Jur00], along with some added enhancements to achieve our bounds. Our algorithm determines in time $O(mn^3)$ and space $O(mn)$ *all* such pairs (q, q') of states in an input automaton A where q' simulates q. Here m denotes the number of transitions and n the number of states of A. In other words, our algorithm computes the entire maximal fair simulation relation on the state space in the stated time and space bound.[2]

In [HKR97] the authors were interested in using fair simulation for purpose (1), and thus did not consider quotients with respect to fair simulation. The question arises whether fair simulation can be used for purpose (2), i. e., whether it satisfies property (**). We give a negative answer by showing that quotienting with respect to fair simulation fails badly to preserve the underlying language, under any reasonable definition of a quotient. On the other hand, there is an obvious and well known way to define simulation so that quotients do preserve the underlying language: *direct* simulation[3] ([Mil89,DHWT91]) simply accommodates acceptance into the standard definition of simulation by asserting that only an accept state can simulate another accept state. Direct simulation has already been used extensively (see, e.g., [EH00,SB00]) to reduce the state space of automata. Both [EH00] and [SB00] describe tools for optimized translations from linear temporal logic to automata, where one of the key optimizations is simulation reduction. However, as noted in [EH00], direct simulation alone is not able to reduce many obviously redundant state spaces. Recall that, in general, it is PSPACE-hard to find the minimum equivalent automaton for a given

[1] There is a typo in the original version of [HKR97] that indicates an $O(n^4)$ running time for their algorithm, but that typo has since been corrected.

[2] D. Bustan and O. Grumberg [BG00] have independently obtained an algorithm for computing fair simulation which, while it does not improve the $O(n^6)$ time complexity of [HKR97], improves the space complexity to $O(n^2)$. Thanks to Moshe Vardi for bringing their work to our attention, and thanks to Orna Grumberg for sending us a copy of their technical report.

[3] Direct simulation is called *strong* simulation in [EH00].

nondeterministic automaton. Thus, there is a need for efficient algorithms and heuristics that reduce the state space substantially.

We introduce a natural intermediate notion between direct and fair simulation, called *delayed* simulation, which satisfies property (**). We show that delayed simulation can yield substantially greater reduction, by an arbitrarily large factor, than direct simulation. We provide an algorithm for computing the entire delayed simulation relation which arises from precisely the same parity-game framework and has the same complexity as our algorithm for fair simulation.

Lastly, our parity game framework also easily accommodates computation of bisimulation relations (which are generally less coarse than simulation). In particular, we show that the fair bisimulation relation on Büchi automata can be computed in time $O(mn^3)$ and $O(mn)$ space. Fair bisimulation was studied by [HR00] for Büchi and Streett automata, who for Büchi automata gave an $O(n^{10})$ time algorithm to compute whether one state is fair bisimilar to another.

The paper is organized as follows: in Sect. 2, we define all (bi)simulation notions used in the paper. In Sect. 3 we show how for each simulation notion (and fair bisimulation), given a Büchi automaton, we can define a parity game that captures the (bi)simulation. In Sect. 4, we use our variant of Jurdzinski's algorithm for parity games to give efficient algorithms for computing several such (bi)simulation relations. In Sect. 5, we prove that the delayed simulation quotient can be used to reduce automaton size, and yields better reduction than direct simulation, but that the fair simulation quotient cannot be so used. We conclude in Sect. 6. Due to lack of space, most proofs must be omitted.

2 Delayed, Fair, and Other (Bi)Simulations

2.1 Simulation

We now define various notions of simulation, including fair and the new delayed simulation, in terms of appropriate games.[4] As usual, a Büchi automaton $A = \langle \Sigma, Q, q_I, \Delta, F \rangle$ has an alphabet Σ, a state set Q, an initial state $q_I \in Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set of final states $F \subseteq Q$. We will henceforth assume that the automaton has no *dead ends*, i. e., from each state of A there is a path of length at least 1 to *some* state in F. It is easy to make sure this property holds without changing the accepting runs from any state, using a simple search to eliminate unnecessary states and transitions.

Recall that a *run* of A is a sequence $\pi = q_0 a_0 q_1 a_1 q_2 \dots$ of states alternating with letters such that for all i, $(q_i, a_i, q_{i+1}) \in \Delta$. The ω-word associated with π is $w_\pi = a_0 a_1 a_2 \dots$ The run π is *initial* if it starts with q_I; it is *accepting* if there exist infinitely many i with $q_i \in F$. The language defined by A is $L(A) = \{w_\pi \mid \pi \text{ is an initial, accepting run of } A\}$. We may want to change the start state of A to a different state q; the revised automaton is denoted by $A[q]$.

[4] For background on simulation, and its versions incorporating acceptance, see, e. g., [Mil89,HHK95], and [HKR97], respectively.

As in [HKR97], we define simulation game-theoretically.[5] The next definition presents all the notions of simulation we will consider: ordinary simulation, which ignores acceptance, as well as three variants which incorporate acceptance conditions of the given automaton, in particular, our new *delayed simulation*.

Definition 1. Given a Büchi automaton A and $(q_0, q_0') \in Q^2$, we define:

1. the *ordinary simulation game,* denoted $\boldsymbol{G}_A^o(q_0, q_0')$,
2. the *direct (strong) simulation game,* denoted $\boldsymbol{G}_A^{di}(q_0, q_0')$,
3. the *delayed simulation game,* denoted $\boldsymbol{G}_A^{de}(q_0, q_0')$,
4. the *fair simulation game,* denoted $\boldsymbol{G}_A^f(q_0, q_0')$.

Each of the games is played by two players, *Spoiler* and *Duplicator,* in rounds as follows. At the start, round 0, two pebbles, *Red* and *Blue*, are placed on q_0 and q_0', respectively. Assume that, at the beginning of round i, *Red* is on state q_i and *Blue* is on q_i'. Then:

1. Spoiler chooses a transition $(q_i, a, q_{i+1}) \in \Delta$ and moves *Red* to q_{i+1}.
2. Duplicator, responding, must chose a transition $(q_i', a, q_{i+1}') \in \Delta$ and moves *Blue* to q_{i+1}'. If no a-transition starting from q_i' exists, then the game halts and Spoiler wins.

Either the game halts, in which case Spoiler wins, or the game produces two infinite runs: $\pi = q_0 a_0 q_1 a_1 q_2 \ldots$ and $\pi' = q_0' a_0 q_1' a_1 q_2' \ldots$, built from the transitions taken by the two pebbles. Given these infinite runs, the following rules are used to determine the winner.

1. *Ordinary simulation:* Duplicator wins in any case. (In other words, fairness conditions are ignored; Duplicator wins as long as the game does not halt).
2. *Direct simulation:* Duplicator wins iff, for all i, if $q_i \in F$, then also $q_i' \in F$.
3. *Delayed simulation:* Duplicator wins iff, for all i, if $q_i \in F$, then there exists $j \geq i$ such that $q_j' \in F$.
4. *Fair simulation:* Duplicator wins iff there are infinitely many j such that $q_j' \in F$ or there are only finitely many i such that $q_i \in F$ (in other words, if there are infinitely many i such that $q_i \in F$, then there are also infinitely many j such that $q_j' \in F$).

Let $\star \in \{o, di, de, f\}$. A *strategy* for Duplicator in game $\boldsymbol{G}_A^\star(q_0, q_0')$ is a function $f\colon (\Sigma Q)^+ \to Q$ which, given the history of the game (actually, the choices of Spoiler) up to a certain point, determines the next move of Duplicator. Formally, f is a strategy for Duplicator if for every run $q_0 \rho a q$ of A, we have $(f(\rho), a, f(\rho a q)) \in \Delta$, where, by convention, $f(\epsilon) = q_0'$. A strategy f for Duplicator is a *winning* strategy if, no matter how Spoiler plays, Duplicator always wins.

[5] We will focus on simulations between distinct states of the same automaton ("autosimulations"), because we are primarily interested in state space reduction. Simulations between different automata can be treated by considering autosimulations between the states of the automaton consisting of their disjoint union. In [HKR97], the authors presented their work in terms of Kripke structures with fairness constraints. We use Büchi automata directly, where labels are on transitions instead of states. This difference is inconsequential for our results.

Formally, a strategy f for Duplicator is winning if whenever $\pi = q_0 a_0 q_1 a_1 \ldots$ is an infinite run through A and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \ldots$ is the run defined by $q'_{i+1} = f(a_0 q_1 a_1 q_2 \ldots q_{i+1})$, then Duplicator wins based on π and π'.

Definition 2. Let A be a Büchi automaton. A state q' *ordinary, direct, delayed, fair*[6] *simulates* a state q if there is a winning strategy for Duplicator in $\boldsymbol{G}^{\star}_{A}(q, q')$ where $\star = o$, di, de, or f, respectively. We denote such a relationship by $q \preceq_{\star} q'$.

Proposition 1. *Let A be a Büchi automaton.*

1. *For $\star \in \{o, di, de, f\}$, $\preceq_{\star}$ is a reflexive, transitive relation (aka,* preorder *or* quasi-order*) on the state set Q.*
2. *The relations are ordered by containment: $\preceq_{di} \subseteq \preceq_{de} \subseteq \preceq_{f} \subseteq \preceq_{o}$.*
3. *For $\star \in \{di, de, f\}$, if $q \preceq_{\star} q'$, then $L(A[q]) \subseteq L(A[q'])$.*

Thus, delayed simulation is a new notion of intermediate "coarseness" between direct and fair simulation. We will see in Sect. 5 why it is more useful for state space reduction.

2.2 Bisimulation

For all the mentioned simulations there are corresponding notions of bisimulation, defined via a modification of the game. For lack of space, we will not provide detailed definitions for bisimulation. Instead we describe intuitively the simple needed modifications. The bisimulation game differs from the simulation game in that *Spoiler* gets to choose in each round which of the two pebbles, *Red* or *Blue*, to move and *Duplicator* has to respond with a move of the other pebble. The winner of the game is determined very similarly: if the game comes to a halt, *Spoiler* wins. If not, the winning condition for *Fair* bisimulation ([HR00]) is: "if an accept state appears infinitely often on one of the two runs π and π', then an accept state must appear infinitely often on the other as well". The winning condition for *Delayed* bisimulation is: "if an accept state is seen at position i of either run, then an accept state must be seen thereafter at some position $j \geq i$ of the other run". The winning conditions for *Direct* bisimulation becomes "if an accept state is seen at position i of either run, it must be seen at position i of both runs". Winning strategies for the bisimulation games are defined similarly. Bisimulations define an equivalence relation $\approx^{bi}_{\star}$ (not a preorder) on the state space, and the following containments hold: $\approx^{bi}_{di} \subseteq \approx^{bi}_{de} \subseteq \approx^{bi}_{f} \subseteq \approx^{bi}_{o}$. Generally, bisimulation is less coarse than the equivalence derived from the simulation preorder, which we describe in Sect. 5, i.e., $\approx^{bi}_{\star} \subseteq \approx_{\star}$.

[6] Our game definition of fair simulation deviates very slightly from that given in [HKR97], but is equivalent since we consider only automata with no dead ends.

3 Reformulating (Bi)Simulations as Parity Games

3.1 Simulation

We now show how, given a Büchi automaton A and $\star \in \{o, di, de, f\}$, we can obtain in a straightforward way a parity game graph $\boldsymbol{G}_A^\star$ such that the winning vertices in $\boldsymbol{G}_A^\star$ for Zero (aka Player 0) in the parity game determine precisely the pairs of states (q, q') of A where q' $\star$-simulates q. Importantly, the size of these parity game graphs will be $O(|Q||\Delta|)$, and the nodes of the game graphs will be labeled by at most three distinct "priorities". In fact, only one priority will suffice for $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$, while $\boldsymbol{G}_A^{de}$ and $\boldsymbol{G}_A^f$ will use three priorities.

We briefly review here the basic formulation of a parity game. A parity game graph $\boldsymbol{G} = \langle V_0, V_1, E, p\rangle$ has two disjoint sets of vertices, V_0 and V_1, whose union is denoted V. There is an edge set $E \subseteq V \times V$, and $p\colon V \to \{0, \ldots, d-1\}$ is a mapping that assigns a *priority* to each vertex.

A parity game on $\boldsymbol{G}$, starting at vertex $v_0 \in V$, is denoted $\boldsymbol{P}(G, v_0)$, and is played by two players, *Zero* and *One*. The play starts by placing a pebble on vertex v_0. Thereafter, the pebble is moved according to the following rule: with the pebble currently on a vertex v_i, and $v_i \in V_0$ (V_1), Zero (One, respectively) plays and moves the pebble to a neighbor v_{i+1}, that is, such that $(v_i, v_{i+1}) \in E$.

If ever the above rule cannot be applied, i. e., someone can't move because there are no outgoing edges, the game ends, and the player who cannot move loses. Otherwise, the game goes on forever, and defines a path $\pi = v_0 v_1 v_2 \ldots$ in $\boldsymbol{G}$, called a *play* of the game. The winner of the game is then determined as follows. Let k_π be the minimum priority that occurs infinitely often in the play π, i. e., so that for infinitely many i, $p(v_i) = k_\pi$ and k_π is the least number with this property; Zero wins if k_π is even, whereas One wins if k_π is odd.

We now show how to build the game graphs $\boldsymbol{G}_A^\star$. All the game graphs are built following the same general pattern, with some minor alterations. We start with $\boldsymbol{G}_A^f$. The game graph $\boldsymbol{G}_A^f = \langle V_0^f, V_1^f, E_A^f, p_A^f\rangle$ will have only three priorities (i. e., the range of p_A^f will be $\{0, 1, 2\}$). For each pair of states $(q, q') \in Q^2$, there will be a vertex $v_{(q,q')} \in V_0^f$ such that Zero has a winning strategy from $v_{(q,q')}$ if and only if q' fair simulates q. Formally, $\boldsymbol{G}_A^f$ is defined by

$$V_0^f = \{v_{(q,q',a)} \mid q, q' \in Q \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \tag{1}$$

$$V_1^f = \{v_{(q,q')} \mid q, q' \in Q\} \ , \tag{2}$$

$$E_A^f = \{(v_{(q_1,q_1',a)}, v_{(q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta\} \cup \{(v_{(q_1,q_1')}, v_{(q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta\} \ , \tag{3}$$

$$p_A^f(v) = \begin{cases} 0, & \text{if } (v = v_{(q,q',a)} \text{ or } v = v_{(q,q')}) \text{ and } q' \in F, \\ 1, & \text{if } v = v_{(q,q')},\ q \in F, \text{ and } q' \notin F, \\ 2, & \text{otherwise.} \end{cases} \tag{4}$$

We now describe how $\boldsymbol{G}_A^f$ can be modified to obtain $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$, both of which require only trivial modification to $\boldsymbol{G}_A^f$. The parity game graph $\boldsymbol{G}_A^o$ is exactly

the same as $\boldsymbol{G}_A^f$, except that all nodes will receive priority 0, i.e., $p_A^o(v) = 0$ for all v. The parity game graph $\boldsymbol{G}_A^{di}$ is just like $\boldsymbol{G}_A^o$, meaning every vertex has priority 0, but some edges are eliminated:

$$E_A^{di} = E_A^f \setminus (\{(v, v_{(q_1,q_1')}) \mid q_1 \in F \wedge q_1' \notin F\} \cup \{(v_{(q_1,q_1')}, w) \mid q_1 \notin F \wedge q_1' \in F\}) \tag{5}$$

Finally, to define $\boldsymbol{G}_A^{de}$ we need to modify the game graph somewhat more. For each vertex of $\boldsymbol{G}_A^f$ there will be at most two corresponding vertices in $\boldsymbol{G}_A^{de}$:

$$V_0^{de} = \{v_{(b,q,q',a)} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \tag{6}$$

$$V_1^{de} = \{v_{(b,q,q')} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge (q' \in F \to b = 0)\} \ . \tag{7}$$

The extra bit b encodes whether or not, thus far in the simulation game, the *Red* pebble has witnessed an accept state without *Blue* having witnessed one since then. The edges of $\boldsymbol{G}_A^{de}$ are as follows:

$$\begin{aligned} E_A^{de} = \{&(v_{(b,q_1,q_1',a)}, v_{(b,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \notin F\} \\ &\cup \{(v_{(b,q_1,q_1',a)}, v_{(0,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \in F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(b,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \notin F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(1,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \in F\} \ . \end{aligned} \tag{8}$$

Lastly, we describe the priority function of $\boldsymbol{G}_A^{de}$:

$$p_A^{de}(v) = \begin{cases} b, & \text{if } v = v_{(b,q,q')}, \\ 2, & \text{if } v \in V_0. \end{cases} \tag{9}$$

In other words, we will assign priority 1 to only those vertices in V_1 that signify that an "unmatched" accept has been encountered by *Red*.[7]

The following lemma gathers a collection of facts we will need.

Lemma 1. *Let A be a Büchi automaton.*

1. *For $\star \in \{o, di, f\}$, Zero has a winning strategy in $\boldsymbol{P}(\boldsymbol{G}_A^\star, v_{(q_0,q_0')})$ if and only if q_0' $\star$-simulates q_0 in A. For $\star = de$, this statement holds if $v_{(q_0,q_0')}$ is replaced by $v_{(b,q_0,q_0')}$, letting $b = 1$ if $q_0 \in F$ and $q_0' \notin F$, and $b = 0$ otherwise.*
2. *For $\star \in \{o, di, de, f\}$, $|\boldsymbol{G}_A^\star| \in O(|\Delta||Q|)$.*
3. *For $\star \in \{f, de\}$, $|\{v \in V_A^\star \mid p_A^\star(v) = 1\}| \in O(|Q|^2)$.*

Since vertices of $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$ only get assigned a single priority, we can dispense with algorithms for computing ordinary and direct simulation right away, matching the best known upper bounds: with one priority the winning set for Zero can be determined by a variant of AND/OR graph accessibility, computable in linear time (see, e. g., [And94]). Thus:

Corollary 1. *([HHK95,BP95]) Given a Büchi automaton A, with n states and m transitions, both $\preceq_o$ and $\preceq_{di}$ can be computed in time $O(mn)$.*

[7] Note that it is possible to use only two priorities in p_A^{de}, by assigning a vertex v the priority b, where b is the indicator bit of v. However, it turns out that using two priorities is a disadvantage over three because the encoding would not have property (3) of Lemma 1, which we need for our complexity bounds.

3.2 Bisimulation

$\star$-bisimulations can also be reformulated as parity games. For improving the complexity, such a reformulation only helps for fair bisimulation. Ordinary and direct bisimulation have known $O(m \log n)$ time algorithms ([PT87]), and we will see that delayed bisimulation corresponds to direct bisimulation after some linear time preprocessing on accept states of the Büchi automaton. We formulate fair bisimulation with a parity game graph G_A^{fbi} as follows. The vertices of G_A^{fbi} are

$$V_0^{fbi} = \{v_{(q,q',a,b_1,b_2)} \mid q, q' \in Q \wedge b_1, b_2 \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \quad (10)$$

$$V_1^{fbi} = \{v_{(q_0,q_1,b_2)} \mid q, q' \in Q \wedge b_2 \in \{0,1\}\} \ . \quad (11)$$

The two bits b_1 and b_2 will encode (1) which pebble was moved by *Spoiler* in this round, and (2) which of the two runs was latest to see (prior to this round) an accept state, respectively. For $q_0, q_1 \in Q$ and $b_2 \in \{0,1\}$, let

$$new(q_0, q_1, b_2) = \begin{cases} 0 & \text{if } q_0 \in F, \\ 1 & \text{if } q_0 \notin F \text{ and } q_1 \in F, \\ b_2 & \text{otherwise.} \end{cases}$$

The edges E_A^{fbi} are

$$\{(v_{(q_0,q_1,b_2)}, v_{(q_0',q_1',a,b_1,b_2')}) \mid (q_{b_1}, a, q_{b_1}') \in \Delta \wedge q_{(1-b_1)} = q_{(1-b_1)}' \wedge b_2' = new(q_0, q_1, b_2)\}$$
$$\cup \{(v_{(q_0,q_1,a,b_1,b_2)}, v_{(q_0',q_1',b_2)}) \mid (q_{(1-b_1)}, a, q_{(1-b_1)}') \in \Delta \wedge q_{b_1} = q_{b_1}'\} \quad (12)$$

The priority function is as follows. For $v \in V_0$, $p_A^{fbi}(v) = 2$, and for $v_{(q_0,q_1,b_2)} \in V_1$,

$$p_A^{fbi}(v_{(q_0,q_1,b_2)}) = \begin{cases} 0 & \text{if } q_{(1-b_2)} \in F, \\ 1 & \text{if } q_{(1-b_2)} \notin F \text{ and } q_{b_2} \in F, \\ 2 & \text{otherwise.} \end{cases} \quad (13)$$

The correspondence of this parity game and fair bisimulation is as follows.

Lemma 2. *Zero has a winning strategy in* $\boldsymbol{P}(\boldsymbol{G}_A^{fbi}, v_{(q_0,q_1,0)})$ *if and only if* q_0 *and* q_1 *are fair-bisimilar in* A*. Furthermore,* $|\boldsymbol{G}_A^{fbi}| \in O(|\Delta||Q|)$ *and* $|\{v \in V_A^{fbi} \mid p_A^{fbi}(v) = 1\}| \in O(|Q|^2)$*.*

To compute delayed bisimulation efficiently, we show that the delayed bisimulation relation corresponds to the direct bisimulation relation after some linear time preprocessing on the accept states of the Büchi automaton. Consider the following closure operation on accept states. Let $cl(A)$ be the Büchi automaton obtained from A by repeating the following until a fixed point is reached: while there is a state q such that all of its successors are in F, put q in F. Clearly, $cl(A)$ can be computed in linear time and $L(A) = L(cl(A))$.

Proposition 2. $q_1 \approx_{de}^{bi} q_2$ *in* A *if and only if* $q_1 \approx_{di}^{bi} q_2$ *in* $\mathrm{cl}(A)$*.*

Thus, $\approx_{de}^{bi}$ can also be computed in time $O(m \log n)$ ([PT87]).

4 Fast Algorithm for Computing Fair (Bi)Simulations and Delayed Simulations

We now use $\boldsymbol{G}_A^f$, $\boldsymbol{G}_A^{fbi}$, and $\boldsymbol{G}_A^{de}$ to give a fast algorithm for computing the relations $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$. Henceforth, we assume all parity game graphs have neither self loops nor dead ends. We can always obtain an "equivalent" such graph in linear time. To obtain the desired complexity bounds for solving parity games, we describe an efficient implementation of an algorithm by Jurdzinski [Jur00]. Jurdzinski uses progress measures (see also [Kla94,Wal96]) to compute the set of vertices in a parity game from which Zero has a winning strategy.

We start with some terminology. Let $\boldsymbol{G}$ be a parity game graph as before, n' its number of vertices, m' its number of edges, and assume there are only three priorities, that is, $p\colon V \to \{0,1,2\}$. Let $n_1 = |p^{-1}(1)|$. The algorithm assigns to each vertex a "progress measure" in the range $D = \{0,\ldots,n_1\} \cup \{\infty\}$. Initially, every vertex is assigned 0. The measures are repeatedly "incremented" in a certain fashion until a "fixed point" is reached. We assume D is totally ordered in the natural way. For $i < 3$ and $x \in D$, we define $\langle x \rangle_i$ as follows. First, $\langle x \rangle_0 = 0$ if $x < \infty$ and $\langle \infty \rangle_0 = \infty$. Second, $\langle x \rangle_1 = \langle x \rangle_2 = x$ for every $x \in D$. In the same spirit, $\mathit{incr}_0(x) = \mathit{incr}_2(x) = x$ for every x, and $\mathit{incr}_1(x) = x + 1$ where, by convention, $n_1 + 1 = \infty$ and $\infty + 1 = \infty$. For simplicity in notation, if $v \in V$, we write $\langle x \rangle_v$ and $\mathit{incr}_v(x)$ for $\langle x \rangle_{p(v)}$ and $\mathit{incr}_{p(v)}(x)$, respectively. For every function $\rho\colon V \to D$, called a *measure*, and $v \in V$, let

$$\mathit{val}(\rho, v) = \begin{cases} \langle \min(\{\rho(w) \mid (v,w) \in E\}) \rangle_v, & \text{if } v \in V_0, \\ \langle \max(\{\rho(w) \mid (v,w) \in E\}) \rangle_v, & \text{if } v \in V_1. \end{cases} \tag{14}$$

Jurdzinski defines a "lifting" operator, which, given a measure ρ and $v \in V$, gives a new measure. In order to define it, we first need to define how an individual vertex's measure is updated with respect to that of its neighbors:

$$\mathit{update}(\rho, v) = \mathit{incr}_v(\mathit{val}(\rho, v)) \ . \tag{15}$$

The "lifted" measure, $\mathit{lift}(\rho, v)\colon V \to D$, is then defined as follows:

$$\mathit{lift}(\rho, v)(u) = \begin{cases} \mathit{update}(\rho, v), & \text{if } u = v, \\ \rho(u), & \text{otherwise.} \end{cases} \tag{16}$$

Jurdzinski's algorithm is depicted in Figure 1.

```
1  foreach v ∈ V do ρ(v) := 0
2   while there exists a v such that  update(ρ, v) ≠ ρ(v) do
3      ρ := lift(ρ, v)
4   endwhile
```

Fig. 1. Jurdzinski's lifting algorithm

The outcome determines the winning set of vertices for each player as follows:

Theorem 1. *([Jur00]) Let $\boldsymbol{G}$ be a parity game with $p\colon V \to \{0,1,2\}$. Zero has a winning strategy from precisely the vertices v such that, after the lifting algorithm depicted in Fig. 1 halts, $\rho(v) < \infty$.*

Jurdzinski's algorithm needs at most $n'(n_1+1)$ iterations of the while loop. More precisely, Jurdzinski argues as follows. Each vertex can only be lifted n_1+1 times. A lifting operation at v can be performed in time $O(|\mathrm{Sucs}(v)|)$ where $\mathrm{Sucs}(v)$ denotes the set of successors of v. So, overall, he concludes, the running time is $O(m'n_1)$. In this analysis, it is implicitly assumed that one can, in constant time, decide if there is a vertex v such that $\mathit{update}(\rho, v) \neq \rho(v)$, and find such a vertex. We provide an implementation of Jurdzinski's algorithm that achieves this when the number of priorities d is bounded by a constant.

Our algorithm, depicted in Figure 2, maintains a set L of "pending" vertices v whose measure needs to be considered for lifting, because a successor has recently been updated resulting in a requirement to update $\rho(v)$. Further, we maintain arrays B and C that store, for each vertex v, the value $\mathit{val}(\rho, v)$ and the number of successors u of v with $\langle \rho(u) \rangle_{p(v)} = \mathit{val}(\rho, v)$, denoted $\mathit{cnt}(\rho, v)$.

```
 1  foreach v ∈ V do
 2    B(v) := 0; C(v) := |{w | (v,w) ∈ E}|; ρ(v) := 0;
 3  L := {v ∈ V | p(v) is odd};
 4  while L ≠ ∅ do
 5    let v ∈ L; L := L \ {v};
 6    t := ρ(v);
 7    B(v) := val(ρ,v); C(v) := cnt(ρ,v); ρ(v) := incr_v(B(v));
 8    P := {w ∈ V | (w,v) ∈ E};
 9    foreach w ∈ P such that w ∉ L do
10      if w ∈ V_0 and t = B(w) and C(w) > 1 then C(w) := C(w) − 1;
11      if w ∈ V_0 and t = B(w) and C(w) = 1 then L := L ∪ {w};
12      if w ∈ V_1 and ρ(v) = B(w) then C(w) := C(w) + 1;
13      if w ∈ V_1 and ρ(v) > B(w) then L := L ∪ {w};
14    endfor
15  endwhile
```

Fig. 2. Efficient implementation of the lifting algorithm

Lemma 3. *The lifting algorithm depicted in Fig. 2 computes the same function ρ as Jurdzinski's algorithm, in time $O(m'n_1)$ and space $O(m')$.*

Proof (sketch). Whether a vertex w needs to be placed on L is determined in constant time by maintaining, for each vertex w, the current "best measure" $B(w)$ of any of its successors, as well as the count $C(w)$ of how many such neighbors there are with the "best measure". The running time follows because each vertex can enter L at most n_1+1 times, and the time taken by the while loop body is proportional to the number of edges incident on the vertex. □

In the general case, where $p\colon V \to \{0,\ldots,d-1\}$, Jurdzinski's algorithm uses a more elaborate measure, where the elements of D are vectors of length $\lfloor d/2 \rfloor$. Our implementation of Jurdzinski's algorithm extends directly to the general case, yielding an additional factor d in comparison with the time bound claimed by Jurdzinski. We do not know how Jurdzinski's claimed bounds can be achieved in the general case without the extra factor d. Finally, using Lemmas 1, 2, and 3, we may conclude:

Theorem 2. *For a Büchi automaton A, $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$ can all be computed in time $O(|\Delta||Q|^3)$ and space $O(|Q||\Delta|)$.*

As mentioned, in prior work $O(|Q|^6)$-time ([HKR97]), and $O(|Q|^{10})$-time ([HR00]), algorithms were given for deciding whether $q \preceq_f q'$, and respectively $q \approx_f^{bi} q'$, hold for a *single* pair of states (q, q').

5 Reducing State Spaces by Quotienting: Delayed Simulation Is Better

In this section, we show: (1) quotienting with respect to delayed simulation preserves the recognized language, (2) that this is not true with fair simulation, and (3) quotients with respect to delayed simulation can indeed be substantially smaller than quotients with respect to direct simulation, even when the latter is computed on the "accept closure" $cl(A)$ (unlike what we saw with delayed bisimulation). We first define quotients.

Definition 3. For a Büchi automaton A, and an equivalence relation $\approx$ on the states of A, let $[q]$ denote the equivalence class of $q \in Q$ with respect to $\approx$. The *quotient* of A with respect to $\approx$ is the automaton $A/{\approx} = \langle \Sigma, Q/{\approx}, \Delta_{\approx}, [q_I], F/{\approx}\rangle$ where $\Delta_{\approx} = \{([q], a, [q']) \mid \exists\ q_0 \in [q],\ q_0' \in [q'],$ such that $(q_0, a, q_0') \in \Delta\}$.

In order to apply our simulation relations, we define, corresponding to each simulation preorder, an equivalence relation $\approx_o$, $\approx_{di}$, $\approx_{de}$, and $\approx_f$, where: $q \approx_\star q'$ iff $q \preceq_\star q'$ and $q' \preceq_\star q$. Note that both $\approx_\star$ and $A/{\approx_\star}$ can be computed from $\preceq_\star$ requiring no more time (asymptotically) than that to compute $\preceq_\star$ on A. The quotient with respect to $\approx_{di}$ preserves the language of any automaton, while this is obviously not true for $\approx_o$. We will later see that this is not true for $\approx_f$ either. We show that this is true for $\approx_{de}$.

Lemma 4. *Let A be a Büchi automaton.*

1. *If $q_0 \preceq_{de} q_0'$ and $(q_0, a, q_1) \in \Delta$, then there exists q_1' with $q_1 \preceq_{de} q_1'$ and $(q_0', a, q_1') \in \Delta$.*
2. *If $q_0 \preceq_{de} q_0'$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \ldots$ is a finite or infinite run of $A/{\approx_{de}}$, then there exists a run $q_0' a_0 q_1' a_1 \ldots$ of A of the same length such that $q_i \preceq_{de} q_i'$ for every i.*
3. *If $q_0 \preceq_{de} q_0''$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \ldots$ is an infinite run of $A/{\approx_{de}}$ with $q_0 \in F$, then there exists a finite run $q_0'' a_0 \ldots a_{r-1} q_r''$ of A such that $q_j \preceq_{de} q_j''$ for $j \leq r$ and $q_r'' \in F$.*

Theorem 3. *For any Büchi automaton A, $L(A) = L(A/{\approx_{de}})$.*

We can thus use $A/{\approx_{de}}$ to reduce the size of A, just as with direct simulation. In fact, $A/{\approx_{de}}$ can be smaller than $A/{\approx_{di}}$ (as well as $cl(A)/{\approx_{di}}$) by an arbitrarily large factor:

Proposition 3. *For $n \geq 2$, there is a Büchi automaton A_n with $n+1$ states such that $A_n/{\approx_{de}}$ has 2 states but $A_n/{\approx_{di}}$ has $n+1$ states (and $A_n = \mathrm{cl}(A_n)$).*

Proof. Consider automaton A_n in Figure 3. It is not hard to establish that in A_n each outer state delayed simulates each other outer state. Thus $A_n/{\approx_{de}}$ has 2 states. On the other hand, $A_n = cl(A_n)$, and no state of A_n direct simulates any other state of A_n. Thus $A_n/{\approx_{di}} = A_n$ and has $n+1$ states. □

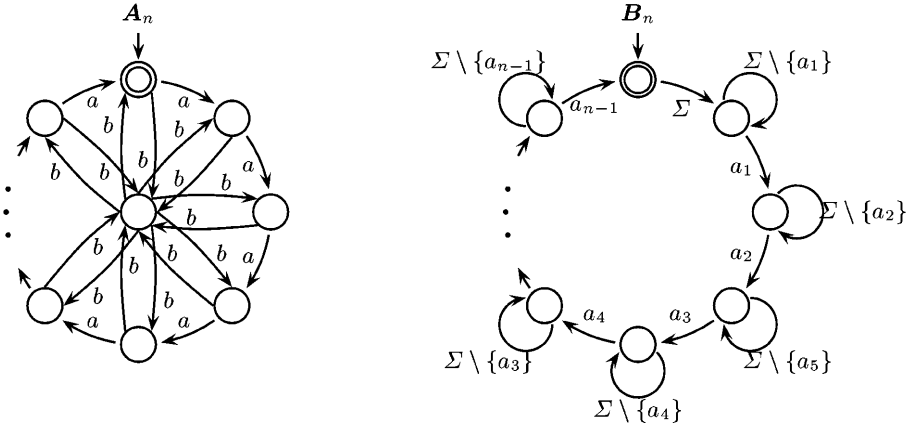

Fig. 3. Automata A_n and B_n

Next we see that Theorem 3 fails badly for fair simulation and bisimulation, that is, fair (bi)simulation cannot be used for state space reduction via quotienting. ([HR00] makes some related observations.)

Proposition 4. *For $n \geq 2$, there is a Büchi automaton B_n with n states, each of which fairly (bi)simulates every other state, but such that no Büchi automaton with fewer than n states accepts $L(B_n)$. In particular, $L(B_n) \neq L(B_n/{\approx_f^{bi}})$.*[8]

Proof. Consider the automaton B_n shown in Figure 3. It has n states, and an alphabet $\Sigma = \{a_1, \ldots, a_{n-1}\}$. To see that every state of B_n fair simulates (and fair bisimulates) every other state, first note that because the automaton is

[8] Note that this inequality holds under any "reasonable" definition of a quotient with respect to $\approx_f^{bi}$ or $\approx_f$, e.g., as long as states of the quotient are equivalence classes.

deterministic Duplicator has no choice in her strategy. A run (played by Spoiler) goes through the accept state infinitely often iff each a_i is encountered infinitely often. But this statement holds no matter which state the run begins from. Thus Duplicator's unique strategy from the initial state pair will be a winning strategy. The language $L(B_n)$ contains precisely those ω-words where each a_i occurs infinitely often. It is not hard to show that there are no Büchi automata recognizing $L(B_n)$ with fewer than n states. □

6 Conclusions

We have presented a unified parity game framework in which to understand optimal known algorithms for a variety of simulation notions for Büchi automata. In particular, we have improved upon the best bounds for fair simulation (and fair bisimulation), matched the best bound for ordinary simulation, and have presented an algorithm for the new notion of delayed simulation. Our algorithms employ a relatively simple fixed point computation, an enhancement of an algorithm by Jurdzinski for parity games, and should perform well in practice.

Our own main aim in using simulations is efficient state space reduction, as in [EH00]. We introduced delayed simulation and showed that, unlike fair simulation, delayed simulation quotients can be used for state space reduction, and allow greater reduction than direct (strong) simulation, which has been used in the past. Optimization of property automata prior to model checking is an important ingredient in making explicit state model checkers such as SPIN more efficient.

References

[And94] H. Andersen. Model checking and boolean graphs. *TCS*, 126(1):3–30, 1994.

[BG00] D. Bustan and O. Grumberg. Checking for fair simulation in models with Büchi fairness constraints, Dec. 2000. Tech. Rep. TR-CS-2000-13, Technion.

[BP95] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, 1995.

[DHWT91] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *Proceedings of CAV'91*, pages 329–341, 1991.

[EH00] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Proc. of 11th Int. Conf on Concurrency Theory (CONCUR)*, pages 153–167, 2000.

[GL94] O. Grumberg and D. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[HHK95] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of 36th IEEE Symp. on Foundations of Comp. Sci. (FOCS'95)*, pages 453–462, 1995.

[HKR97] T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. of 9th Int. Conf. on Concurrency Theory (CONCUR'97)*, number 1243 in LNCS, pages 273–287, 1997.

[HR00] T. Henzinger and S. Rajamani. Fair bisimulation. In *TACAS*, 2000.

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000, 17th Symp. on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, 2000.

[Kla94] N. Klarlund. Progress measures,immediate determinacy, & a subset construction for tree automata. *Ann. Pure & Applied Logic*, 69:243–268, 1994.

[KV98] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *Proc. 30th ACM Symp. on Theory of Computing*, 1998.

[Mil89] R. Milner. *Communicatation and Concurrency*. Prentice-Hall, 1989.

[PT87] R. Paige and R. E. Tarjan. Three partition-refinement algorithms. *SIAM J. of Computing*, 16(6):973–989, 1987.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of 12th Int. Conf. on Computer Aided Verification*, 2000.

[Wal96] I. Walukiewicz. Pushdown processes: games and model checking. In *Computer Aided Verification*, LNCS, pages 62–75. springer-verlag, 1996.