SIAM J. COMPUT.
Vol. 34, No. 5, pp. 1159–1175

FAIR SIMULATION RELATIONS, PARITY GAMES, AND STATE SPACE REDUCTION FOR BÜCHI AUTOMATA*

KOUSHA ETESSAMI†, THOMAS WILKE‡, AND REBECCA A. SCHULLER§

Abstract. We give efficient algorithms, improving 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 also a new natural notion of simulation called *delayed* simulation, which we introduce for the purpose of state space reduction. We show that delayed simulation—unlike fair simulation—preserves the automaton language upon quotienting and allows substantially better state space reduction than direct simulation.

Using our parity-game approach, which relies on an algorithm by Jurdziński, we give efficient algorithms for computing all of the above simulations. In particular, we obtain an $O(mn^3)$-time and $O(mn)$-space algorithm for computing both the delayed and the fair simulation relations. The best prior algorithm for fair simulation requires time and space $O(n^6)$. Our framework also allows one to compute bisimulations: 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 and space $O(n^{10})$.

Key words. fair simulation relations, parity games, state space reduction, Büchi automata

AMS subject classification. 68Q45

DOI. 10.1137/S0097539703420675

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., [20, 16]), mainly aimed at comparing the branching behavior of such machines (rather than just their sets of traces). For ω-automata, where acceptance (fairness) conditions are present, there are a variety of different simulation notions (see, e.g., [14, 11]). At a minimum, for such a simulation to be of use for purpose (1), it must have the following property:

(*) 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.

As we will see in section 5, however, this property alone is not 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 [14], a number of 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'

*Received by the editors January 6, 2003; accepted for publication (in revised form) August 18, 2004; published electronically June 3, 2005. This paper is based on the conference paper [9].
http://www.siam.org/journals/sicomp/34-5/42067.html
†University of Edinburgh, Edinburgh, Scotland EH8 9YL (kousha@inf.ed.ac.uk).
‡Christian-Albrechts-Universität, 24098 Kiel, Germany (wilke@ti.informatik.uni-kiel.de).
§Cornell University, Ithaca, NY 14853 (reba@math.cornell.edu).

fairly simulates q. Their algorithm relies on an algorithm for tree automaton emptiness testing developed in [19]. 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 Jurdziński [17], 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.

In [14], 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 (**). The answer is no: we show that quotienting with respect to fair simulation fails badly at preserving the underlying language, under any reasonable definition of a quotient. (Closely related observations were also made in [15].) 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[1] [6] simply accommodates acceptance into the standard definition of simulation [20] by asserting that only an accept state can simulate another accept state. Direct simulation has already been used extensively (see, e.g., [8, 22]) to reduce the state space of automata. See also [5], where simulation minimization for ordinary Kripke structures was studied. Both [8] and [22] describe tools for optimized translations from linear temporal logic to automata, where one of the key optimizations is simulation reduction. However, as noted in [8], 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 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.

Last, 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 space $O(mn)$. Fair bisimulation was studied in [15] for Büchi and Streett automata, where for Büchi automata they gave an $O(n^{10})$–time and space algorithm to compute whether one state is fair bisimilar to another.

This paper is based on the conference paper [9]. Several papers have since appeared that build on and/or use our work: [13, 10, 7, 4]. Independent of [9], in [3] Bustan and Grumberg obtained an algorithm for computing fair simulation which, while it did not improve the $O(n^6)$-time complexity of [14], improved the space complexity to $O(mn)$. The algorithms described here have been implemented in the TMP tool, available for download at http://www1.bell-labs.com/project/TMP/.

The paper is organized as follows: in section 2, we define all (bi)simulation notions used in the paper. In section 3 we show how for each simulation notion (and fair bisimulation), given a Büchi automaton, we can define a parity game that captures

[1] Direct simulation is called *strong* simulation in [8].

the (bi)simulation. In section 4, we use our variant of Jurdziński's algorithm for parity games to give efficient algorithms for computing several such (bi)simulation relations. In section 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 section 6.

For background on simulation and its versions incorporating acceptance, see, e.g., [20, 16] and [14], respectively. For background on Büchi automata and automata on infinite words in general, see [23, 24, 12].

We thank an anonymous referee for pointing out that an earlier version of the algorithm in Figure 2 was too complicated.

2. Simulation and bisimulation relations. We now define various notions of simulation, including fair and the new delayed simulation, in terms of appropriate games.

2.1. Büchi automata. 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. Unless the automaton is trivial (i.e., has an empty ω-language), 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. (Also, it is easy to check nontriviality while doing the same search. The running time of the search is linear in the size of the automaton.)

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 $L(A) = \{w_\pi \in \Sigma^\omega \mid \pi \text{ is an initial, accepting run of } A\}$. We may want to change the start state of A to a different state q; the revised automaton is denoted by $A[q]$.

2.2. Simulation relations. As in [14], we define simulation game-theoretically. 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 [14], 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.

We will consider four kinds of simulations: ordinary simulation, which ignores acceptance, as well as three variants which incorporate acceptance conditions of the given automaton, in particular, our new delayed simulation. All four simulations are based on the same game, which is described first. They differ only in the winning condition.

Let A be a Büchi automaton as above and q_0 and q_0' arbitrary states of A. The *basic game* $G_A(q_0, q_0')$ is played by two players, *Spoiler* and *Duplicator,* in rounds, where, at the beginning and at the end of each round, two pebbles, *Red* and *Blue*, are placed on two states (possibly the same). At the start, round 0, 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 choose 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, $\pi = q_0 a_0 q_1 a_1 q_2 \dots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$, built from the transitions visited by the two pebbles (and such that the same word is associated with them). The pair (π, π') is called an *outcome* of the game. Given such an outcome, the following rules are used to determine the winner.

DEFINITION 1 (simulation games). *Let A be a Büchi automaton, let $(q_0, q'_0) \in Q^2$, and let (π, π') be an outcome of $G_A(q_0, q'_0)$ with $\pi = q_0 a_0 q_1 a_1 q_2 \dots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$*

1. *The* ordinary simulation game, *denoted by $G^o_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome (π, π') is winning for Duplicator (i.e., as long as the game does not halt, Duplicator wins).*

2. *The* direct (strong) simulation game, *denoted by $G^{di}_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome (π, π') is winning for Duplicator iff, for all i, if $q_i \in F$, then also $q'_i \in F$.*

3. *The* delayed simulation game, *denoted by $G^{de}_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome (π, π') is winning for Duplicator iff, for all i, if $q_i \in F$, then there exists $j \geq i$ such that $q'_j \in F$.*

4. *The* fair simulation game, *denoted by $G^f_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome (π, π') is winning for Duplicator 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 all other cases, Spoiler wins.*

In other words, in ordinary simulation games, fairness conditions are ignored; Duplicator wins as long as the game does not halt. And in fair simulation games, Duplicator's winning condition is as follows: if there are infinitely many i such that $q_i \in F$, then there are also infinitely many j such that $q'_j \in F$.

Remark 1. Let A be a Büchi automaton and $\star \in \{di, de, f\}$. If (π, π') is the outcome of a play of $G^\star_A(q, q')$ which Duplicator wins, then π' is accepting if π is.

Let $\star \in \{o, di, de, f\}$. A *strategy* for Duplicator in game $G^\star_A(q_0, q'_0)$ is a partial function $f\colon Q(Q\Sigma Q)^* \to Q$ which, given the history of the game up to a certain point, determines the next move of Duplicator. Formally, f is a strategy for Duplicator if $f(q_0) = q'_0$ and $(q'_i, a_i, q'_{i+1}) \in \Delta$ holds for every sequence $q_0 q'_0 a_0 q_1 q'_1 a_1 \dots a_i q_{i+1}$ with $(q_j, a_j, q_{j+1}) \in \Delta$ and $q'_j = f(q_0 q'_0 a_0 \dots a_j q_j)$ for $j \leq i$. Observe that the existence of a strategy implies that Duplicator has a way of playing such that the game does not halt. A strategy f for Duplicator is a *winning* strategy if, no matter how Spoiler plays, Duplicator always wins. Formally, a strategy f for Duplicator is winning if whenever $\pi = q_0 a_0 q_1 a_1 \dots$ is a run through A and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$ is the run defined by

$$q'_{i+1} = f(q_0 q'_0 a_0 q_1 q'_1 a_1 \dots q_{i+1}), \tag{1}$$

then (π, π') is winning for Duplicator (as specified in Definition 1). Observe that π' is well-defined.

DEFINITION 2 (simulation relations). *Let A be a Büchi automaton. A state q'* ordinary, direct, delayed, fair simulates *a state q if there is a winning strategy for Duplicator in $G^\star_A(q, q')$ where $\star = o$, di, de, or f, respectively. We denote such a relationship by $q \preceq_\star q'$ (where A is implicit).*

Our game definition of fair simulation deviates very slightly from that given in [14], but is equivalent since we consider only automata with no dead ends.

We first prove fundamental properties of the defined simulation relations.

PROPOSITION 3. *Let A be a Büchi automaton.*

1. *For $\star \in \{o, di, de, f\}$, $\preceq_\star$ is a reflexive, transitive relation (also called* 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 $L(A[q]) \subseteq L(A[q'])$.*

Proof. Reflexivity is obvious, as is part 2. To prove transitivity, suppose that $q_0 \preceq_\star q_0' \preceq_\star q_0''$ for some $\star \in \{o, di, de, f\}$. Then, by definition, Duplicator has winning strategies in both $G_A^\star(q_0, q_0')$ and $G_A^\star(q_0', q_0'')$, say f and f'. We combine these to get a winning strategy f'' for Duplicator in the game $G_A^\star(q_0, q_0'')$ as follows. If $f(q_0 q_0' a_0 q_1 q_1' a_1 \dots q_{i+1}) = q_{i+1}'$ and $f'(q_0' q_0'' a_0 q_1' q_1'' a_1 \dots q_{i+1}') = q_{i+1}''$, we set $f(q_0 q_0'' a_0 q_1 q_1'' a_1 \dots q_{i+1}) = q_{i+1}''$. It is easy to see that this defines a strategy for Duplicator. To see that f'' is in fact winning, let $\pi = q_0 a_0 q_1 a_1 \dots$ be a run through A and let $\pi'' = q_0'' a_0 q_1'' a_1 \dots$ be the run defined by

$$q_{i+1}'' = f''(q_0 q_0'' a_0 q_1 q_1'' a_1 \dots q_{i+1}). \tag{2}$$

We need to argue that (π, π'') is winning for Duplicator. By induction, one easily proves that if $\pi' = q_0' a_0 q_1' a_1 \dots$ is defined by (1), then

$$q_{i+1}'' = f'(q_0' q_0'' a_0 q_1' q_1'' a_1 \dots q_{i+1}'). \tag{3}$$

This means that (π, π') is winning for Duplicator in $G_A^\star(q_0, q_0')$ and (π', π'') is winning for Duplicator in $G_A^\star(q_0', q_0'')$. For instance, when $\star = de$, this implies the following: if $q_i \in F$, there exists $j \geq i$ such that $q_j' \in F$, which, in turn, means there exists $k \geq j$ such that $q_k'' \in F$. That is, (π, π'') is winning for Duplicator in $G_A^{de}(q_0, q_0'')$. Similar arguments apply in the other cases.

To prove part 3, assume $\star \in \{di, de, f\}$, $q_0 \preceq_\star q_0'$, and $w \in L(A[q_0])$ with $w = a_0 a_1 \dots$. Then there exists a winning strategy f for Duplicator in $G_A^\star(q_0, q_0')$ and an accepting run $\pi = q_0 a_0 q_1 a_1 \dots$ of A starting with q_0. Imagine Spoiler plays in $G_A^\star(q_0, q_0')$ just as π prescribes this and Duplicator replies according to f. Then a run $\pi' = q_0' a_0 q_1' a_1 \dots$ of A is built up according to (1). Since π is accepting and f is winning, π' will also be accepting; see Remark 1. □

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

2.3. Bisimulation relations. For all the mentioned simulations there are corresponding notions of bisimulation, defined via a modification of the game. 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 (see [15]) is as follows: "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 as follows: "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 condition 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."

Strategies and winning strategies for the bisimulation games are defined similarly. Note, however, that the definitions have to take into account that Spoiler may choose his pebble.

Bisimulations define an equivalence relation $\approx_\star^{bi}$ (not only a preorder) on the state space, and the following containments hold: $\approx_{di}^{bi} \subseteq \approx_{de}^{bi} \subseteq \approx_f^{bi} \subseteq \approx_o^{bi}$. Generally, bisimulation is less coarse than the equivalence derived from the simulation preorder, which we describe in section 5, i.e., $\approx_\star^{bi} \subseteq \approx_\star$.

3. Reformulating simulations and bisimulations 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 $G_A^\star$ such that the winning vertices in $G_A^\star$ for Even (a.k.a. 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 G_A^o and G_A^{di}, while G_A^{de} and G_A^f will use three priorities.

We briefly review here the basic formulation of a parity game. A parity-game graph $G = \langle V_0, V_1, E, p\rangle$ 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 G, starting at vertex $v_0 \in V$, is denoted $\mathrm{P}(G, v_0)$, and is played by two players, *Even* and *Odd*. 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), Even (Odd, 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 G, called a *play* of the game. The winner of the play is 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. Even wins if k_π is even, whereas Odd wins if k_π is odd.

We now show how to build the game graphs $G_A^\star$. All the game graphs are built following the same general pattern, with some minor alterations. We start with G_A^f. The game graph $G_A^f = \langle V_0^f, V_1^f, E_A^f, p_A^f\rangle$ will have 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 Even has a winning strategy from $v_{(q,q')}$ iff q' fair simulates q. Formally, G_A^f is defined by

$$V_0^f = \{v_{(q,q',a)} \mid q, q' \in Q \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{4}$$

$$V_1^f = \{v_{(q,q')} \mid q, q' \in Q\}, \tag{5}$$

$$\begin{aligned} E_A^f = {} & \{(v_{(q_1,q_1',a)}, v_{(q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta\} \\ & \cup \{(v_{(q_1,q_1')}, v_{(q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta\}, \end{aligned} \tag{6}$$

$$p_A^f(v) = \begin{cases} 0 & \text{if } v = v_{(q,q')} \text{ and } q' \in F, \\ 1 & \text{if } v = v_{(q,q')},\ q \in F, \text{ and } q' \notin F, \\ 2 & \text{otherwise.} \end{cases} \tag{7}$$

Let's first explain the underlying idea. The parity game mimics the simulation game. Even takes over the role of Duplicator and Odd takes over the role of Spoiler: when in the parity game the current position is node $v_{(q,q')}$, it denotes the situation in the simulation game when the red pebble is on q, the blue pebble is on q', and it is Spoiler's turn to move; $v_{(q,q',a)}$ denotes the situation where the red pebble is on q, the blue pebble is on q', it is Duplicator's turn to move, and the last transition taken by Spoiler was labeled by a. The priority function is defined in such a way that every time Duplicator visits a final state, the priority function returns 0. It returns only 1 if Spoiler visits a final state, but Duplicator does not. In all other cases, 2 is returned. That is, Spoiler wins iff he visits an accept state infinitely often but Duplicator does not. This is exactly what is needed.

We now describe how G_A^f can be modified to obtain G_A^o and G_A^{di}, both of which require only trivial modification to G_A^f. The parity-game graph G_A^o is exactly the same as G_A^f, except that all nodes will receive priority 0, i.e., $p_A^o(v) = 0$ for all v. This reflects the winning condition of the ordinary simulation game.

The parity-game graph G_A^{di} is just like G_A^o, meaning every vertex has priority 0, but some edges (the ones into and out of states of the form $v_{(q,q')}$ where $q \in F$ but $q' \notin F$) are eliminated in order to take care of the winning condition of the direct simulation game:

$$E_A^{di} = E_A^f \setminus (\{(v, v_{(q,q')}) \mid q \in F \wedge q' \notin F\} \cup \{(v_{(q,q')}, w) \mid q \in F \wedge q' \notin F\}). \tag{8}$$

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

$$V_0^{de} = \{v_{(b,q,q',a)} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{9}$$

$$V_1^{de} = \{v_{(b,q,q')} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge (q' \in F \to b = 0)\}. \tag{10}$$

The extra bit b encodes whether or not, thus far in the simulation game, the red pebble has witnessed an accept state without the blue pebble having witnessed one since then. The edges of G_A^{de} are as follows:

$$\begin{aligned} E_A^{de} = {} & \{(v_{(b,q_1,q_1',a)}, v_{(b,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \notin F\} \\ & \cup \{(v_{(b,q_1,q_1',a)}, v_{(0,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \in F\} \\ & \cup \{(v_{(b,q_1,q_1')}, v_{(b,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \notin F\} \\ & \cup \{(v_{(b,q_1,q_1')}, v_{(1,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \in F\}. \end{aligned} \tag{11}$$

Last, we describe the priority function of G_A^{de}:

$$p_A^{de}(v) = \begin{cases} b & \text{if } v = v_{(b,q,q')}, \\ 2 & \text{if } v \in V_0^{de}. \end{cases} \tag{12}$$

In other words, we will assign priority 1 to only those vertices in V_1 that signify that an "unmatched" accept has been visited by the red pebble.[2] The priority function

[2]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 4, which we need for our complexity bounds.

makes sure that the smallest number occurring infinitely often is 1 iff from some point onwards the bit in the first component is 1. Now observe that this bit is 1 iff a final state has been visited by Spoiler but not yet matched by Duplicator. In this way the winning condition of the delayed simulation game is transferred to the parity game.

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

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

1. *For $\star \in \{o, di, f\}$, Even has a winning strategy in $\mathrm{P}(G^\star_A, v_{(q_0,q_0')})$ iff 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 $\star \in \{o, di, de, f\}$, $|G^\star_A| \in O(|\Delta||Q|)$, i.e., the number of vertices and the number of edges is $O(|\Delta||Q|)$.*

3. *For $\star \in \{f, de\}$, $|\{v \in V^\star_A \mid p^\star_A(v) = 1\}| \in O(|Q|^2)$.*

Proof. Part 1 is obvious from the explanations given above.

To prove part 2, first consider the case where $\star \in \{f, o\}$. In this case, $V^\star_1$ contains exactly $|Q|^2$ vertices, and since by assumption every state of A has a transition leaving it, $|Q|^2 \leq |\Delta||Q|$. Similarly, $V^\star_0$ has, for every q', a state $v_{(q,q',a)}$ iff there is a transition to q labeled by a. Thus $|V_0| \leq |\Delta||Q|$.

As far as $|E^f_A|$, for every transition $(q, a, q') \in \Delta$, and every $q'' \in Q$, there is an edge $(v_{(q,q'')}, v_{(q',q'',a)}) \in E^\star$. There are $|\Delta||Q|$ such edges. Likewise, there are $\leq |\Delta||Q|$ edges from $V^\star_0$ to $V^\star_1$. So $|E^f_A| \in O(|\Delta||Q|)$. Thus, the size of $G^\star_A$ is $O(|\Delta||Q|)$. Now observe that if $\star = o$, the vertices do not change and the edge set is a subset, and if $\star = de$, the number of vertices and edges is larger by at most a factor of 2.

Last, since the vertices labeled by priority 1 in both G^f and G^{de} are a subset of V_0, clearly $|p^{-1}(1)| \in O(|Q|^2)$. □

Since vertices of G^o_A and G^{di}_A 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:

PROPOSITION 5 (see [16, 2]). *Given a Büchi automaton A, with n states and m transitions, both $\preceq_o$ and $\preceq_{di}$ can be computed in time and space $O(mn)$.*

Proof. $G^\star_A$ here has size $O(|\Delta||Q|)$ and only one priority. For such game graphs, we can compute the winning set for Even using a variant of AND/OR graph accessibility, which can be computed in linear time (see, e.g., [1]). The only vertices in the game graph that have no outgoing edges are in V_0. These are losing positions for Even, as are any other vertices from which these are accessible in the and/or sense (vertices from V_0 are considered "and nodes" and vertices from V_1 are considered "or nodes"). All the remaining vertices are winning positions for Even. □

Algorithms for computing the other simulation relations will be presented in section 4.

3.2. Bisimulation. For $\star \in \{o, di, de, f\}$, $\star$-bisimulations can also be reformulated as parity games. For improving the complexity, such a reformulation helps only for fair bisimulation. Ordinary and direct bisimulation have known $O(m \log n)$-time algorithms (see [21]), 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^{fbi}_A as follows. The

vertex sets of G_A^{fbi} are

$$V_0^{fbi} = \{v_{(q,q',a,b_1,b_2)} \mid q, q' \in Q \wedge b_1, b_2 \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{13}$$

$$V_1^{fbi} = \{v_{(q_0,q_1,b_2)} \mid q, q' \in Q \wedge b_2 \in \{0,1\}\}. \tag{14}$$

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 pebbles was latest to visit an accept state (prior to this round and with precedence for the red pebble, with 0 encoding the red pebble). For $q_0, q_1 \in Q$ and $b_2 \in \{0,1\}$, let

$$\text{new}(q_0, q_1, b_2) = \begin{cases} 0 & \text{if } q_0 \in F, \\ 1 & \text{if } q_0 \notin F \text{ and } q_1 \in F, \\ b_2 & \text{otherwise.} \end{cases} \tag{15}$$

The edge set E_A^{fbi} is the union of

$$\{(v_{(q_0,q_1,b_2)}, v_{(q_0',q_1',a,b_1,\text{new}(q_0,q_1,b_2))}) \mid (q_{b_1}, a, q'_{b_1}) \in \Delta \wedge q_{1-b_1} = q'_{1-b_1}\} \tag{16}$$

and

$$\{(v_{(q_0,q_1,a,b_1,b_2)}, v_{(q_0',q_1',b_2)}) \mid (q_{1-b_1}, a, q'_{1-b_1}) \in \Delta \wedge q_{b_1} = q'_{b_1}\}. \tag{17}$$

The priority of a vertex is determined using the following function. For q_0, q_1, b let

$$\text{pr}(q_0, q_1, b) = \begin{cases} 0 & \text{if } q_{1-b} \in F, \\ 1 & \text{if } q_{1-b} \notin F \text{ and } q_b \in F, \\ 2 & \text{otherwise.} \end{cases} \tag{18}$$

For $v \in V_0$, $p_A^{fbi}(v) = 2$, and for $v_{(q_0,q_1,b_2)} \in V_1$,

$$p_A^{fbi}(v_{(q_0,q_1,b_2)}) = \text{pr}(q_0, q_1, b_2). \tag{19}$$

The correspondence of this parity game and fair bisimulation is as follows, similar to Lemma 4.

LEMMA 6. *Let A be a Büchi automaton.*

1. *Even has a winning strategy in $\text{P}(G_A^{fbi}, v_{(q_0,q_1,0)})$ iff q_0 and q_1 are fair-bisimilar in A.*
2. *$|G_A^{fbi}| \in O(|\Delta||Q|)$ and $|\{v \in V_A^{fbi} \mid p_A^{fbi}(v) = 1\}| \in O(|Q|^2)$.*

Proof. It is clear that the parity game models the bisimulation game in a straightforward way as far as the sequence of the visited positions is concerned. We show that the winning condition is also taken care of.

Assume $\pi = q_0 a_0 q_1 a_1 \dots$ and $\pi' = q'_0 a_0 q'_1 a_1 \dots$ are two runs. Let $b_0 b_1 \dots$ be the sequence of bits defined by $b_0 = 0$ and $b_{i+1} = \text{new}(q_i, q'_i, b_i)$. Finally, let p_i be defined by $p_i = \text{pr}(q_i, q'_i, b_i)$. This describes exactly what happens in the game. We proceed by a case distinction.

Clearly, if π and π' are not accepting, then $p_j = 2$ for all j large enough and Even wins, which is required. Next, assume π is accepting, and π' is not. Then there exists i such that $q'_j \notin F$ for $j \geq i$ and $q_j \in F$ for infinitely many j, say $i_0 < i_1 < i_2 < \cdots$ are such that $q_{i_j} \in F$ for every j. Assume $i_k > i$. According to the definition of *new*, $b_{i_j} = 0$ for $j > k$. Thus, by definition of *pr*, for every $j > i_k$, $p_j = 1$ if $j = i_l$ for some $l > k$ and $p_j = 2$ otherwise—Even loses. The same argument applies if π is not

accepting but π' is. (The precedence for red does not play any role here.) Finally, assume π and π' are accepting. Let $i_0 < i_1 < \cdots$ be an infinite sequence such that $q_{i_j} \in F$ for all j. For j, let i'_j be the least $k > i_j$ such that $q'_k \in F$. We will have $b_k = 0$ for $i_j < k \leq i'_j$, which means $p_{i'_j} = 0$ for every j—Even wins.

The claim about the size of G_A^{fbi} and the number of its vertices of priority 1 can be proved along the same lines as Lemma 4. □

This enables us to give an efficient algorithm for computing fair bisimulation in section 4.

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 the set of accept states. Let $\mathrm{cl}(A)$ 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. Call the revised set of accept states F'. Clearly, $\mathrm{cl}(A)$ can be computed in linear time and $L(A) = L(\mathrm{cl}(A))$.

PROPOSITION 7. *Let A be a Büchi automaton. For any two states q and q', $q \approx^{bi}_{de} q'$ in A iff $q \approx^{bi}_{di} q'$ in $\mathrm{cl}(A)$.*

Proof. We show that for every pair (q, q') of states, a winning strategy for Duplicator in the delayed bisimulation game on (q, q') in A is a winning strategy for Duplicator in the direct bisimulation game on (q, q') in $\mathrm{cl}(A)$, and vice versa. By definition of the bisimulation relations, this proves the proposition.

First, let f be a winning strategy for Duplicator in the delayed bisimulation game on (q_0, q'_0) in A. Suppose that with Duplicator playing according to strategy f the direct bisimulation game reaches (q_i, q'_i) after i rounds. We have to show that $q_i \in F'$ iff $q'_i \in F'$. Suppose, for contradiction, that $q_i \notin F'$, while $q'_i \in F'$ (the other situation is symmetric). We will show how Spoiler can win the delayed bisimulation game. Since $q_i \notin F'$, there is an infinite path leaving q_i such that no state on this path is an accepting state. Spoiler's strategy is to play this path. Since $q'_i \in F'$, there is no such path (without an accept state on it) starting at q'_i. Therefore, regardless of how Duplicator plays, when $(q_0 a_0 q_1 a_1 \ldots, q'_0 a_0 q'_1 a_1 \ldots)$ is the outcome of the play, then $q'_i \in F'$, but $q_j \notin F'$ for all $j \geq i$—Spoiler wins the delayed bisimulation game.

Second, let f be a winning strategy for Duplicator in the direct bisimulation game on (q, q') in $\mathrm{cl}(A)$ and suppose Duplicator plays according to f in the delayed bisimulation game. Let $(q_0 a_0 q_1 a_1 \ldots, q'_0 a_0 q'_1 a_1 \ldots)$ be any outcome of such a play. We have to show that it satisfies Duplicator's winning condition. So let i be any index such that $q_i \in F$. Then, by definition of F', $q_i \in F'$. But since f is a winning strategy in the direct bisimulation game, this implies $q'_i \in F'$. As every infinite path out of q'_i contains an accept state, there must be a $j \geq i$ such that $q'_j \in F$. Symmetrically, if $q'_i \in F$, then there exists $j \geq i$ such that $q_j \in F$. □

Taking into account that direct bisimulation can be computed in time $O(m \log n)$ (see [21]), we conclude with the following result.

COROLLARY 8. *Delayed bisimulation can be computed in time $O(m \log n)$.*

4. Fast parity-game algorithm to compute simulations (and bisimulations) efficiently. Using the parity-game graphs G_A^f, G_A^{fbi}, and G_A^{de}, we give fast algorithms for computing the relations $\preceq_f$, $\approx^{bi}_f$, and $\preceq_{de}$. To this effect, we describe an efficient implementation of an algorithm for solving parity games presented by Jurdziński in [17]. This algorithm uses progress measures (see also [18, 25]) to compute the set of vertices in a parity game from which Even has a winning strategy.

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.) We start with some terminology, closely following the notation of [17]. Let G be a parity-game graph as before, n' its number of vertices, m' its number of edges. For computing simulations, we only need assume there are only three priorities, that is, $p\colon V \to \{0,1,2\}$. However, we will present the algorithm in its full generality, i.e., $p\colon V \to \{0,1,\ldots,d-1\}$, since the algorithm is of much broader interest.

Let $[n] = \{0,\ldots,n-1\}$, and let $n_i = |p^{-1}(i)|$. The algorithm assigns to each vertex a "progress measure" from $M_G^\infty = M_G \cup \{\infty\}$, where

$$M_G = \begin{cases} [1]\times[n_1+1]\times[1]\times[n_3+1]\times\cdots[1]\times[n_{d-1}+1] & \text{if } d \text{ is even,}\\ [1]\times[n_1+1]\times[1]\times[n_3+1]\times\cdots[1]\times[n_{d-2}+1] & \text{if } d \text{ is odd.}\end{cases} \tag{20}$$

In other words, a progress measures is either ∞ or a length d vector which at even index positions is 0, and at odd index positions i ranges over $\{0,\ldots,n_i\}$.

Initially, every vertex is assigned 0, the all-zero vector. The measures are repeatedly "incremented" in a certain fashion until a fixed point is reached.

We first explain the increment operation, which is at the heart of Jurdziński's algorithm. For $i < d$ and $x \in M_G^\infty$ we define $\langle x\rangle_i$ as follows. For $x = (l_0,\ldots,l_{d-1})$, $\langle x\rangle_i = (l_0,\ldots,l_i,0,0,\ldots,0)$. In other words, we set positions indexed $> i$ to 0. Moreover, $\langle\infty\rangle_i = \infty$. We define a lexicographic total order on M_G^∞, denoted $>$. Here, index 0 is the most significant position, and ∞ is greater than all other vectors. In addition, for d-vectors x and y, we write $x >_i y$ iff $\langle x\rangle_i > \langle y\rangle_i$ according to the above ordering. For example, $(0,3,0,1) >_1 (0,2,0,3)$. Note that $x > y$ iff $x >_{d-1} y$. Now, we can say what it means to "increment" a progress measure. For each $i \in [d]$, let

$$\mathrm{incr}_i(x) = \begin{cases} \langle x\rangle_i & \text{if } i \text{ is even } x \neq \infty,\\ \min\{y \in M_G^\infty \mid y >_i x\} & \text{if } i \text{ is odd, } x \neq \infty,\\ \infty & \text{if } x = \infty.\end{cases} \tag{21}$$

Observe that, for a fixed i, the operation $\mathrm{incr}_i(\cdot)$ is monotone with respect to the ordering $<$; that is, if $x \le x'$, then $\mathrm{incr}_i(x) \le \mathrm{incr}_i(x')$.

For simplicity in notation, if $v \in V$, we write $\langle x\rangle_v$ and $\mathrm{incr}_v(x)$ for $\langle x\rangle_{p(v)}$ and $\mathrm{incr}_{p(v)}(x)$, respectively. For every assignment $\rho\colon V \to M_G^\infty$ of progress measures to the vertices of a game graph, which we call an *assignment* for short, and for $v \in V$, let

$$\text{best-nghb-ms}(\rho,v) = \begin{cases} \langle\min(\{\rho(w) \mid (v,w)\in E\})\rangle_v & \text{if } v \in V_0,\\ \langle\max(\{\rho(w) \mid (v,w)\in E\})\rangle_v & \text{if } v \in V_1.\end{cases} \tag{22}$$

Here, best-nghb-ms(ρ,v) stands for the set of *best neighbors* of v with respect to the *measure* we have defined. Jurdziński defines a "lifting" operator, which, given an assignment ρ and $v \in V$, gives a new assignment. In order to define it, he first defines how an individual vertex's measure is "updated" with respect to those of its neighbors:

$$\mathrm{update}(\rho,v) = \mathrm{incr}_v(\text{best-nghb-ms}(\rho,v)). \tag{23}$$

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

$$\mathrm{lift}(\rho,v)(u) = \begin{cases} \mathrm{update}(\rho,v) & \text{if } u = v,\\ \rho(u) & \text{otherwise.}\end{cases} \tag{24}$$

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

FIG. 1. *Jurdziński's lifting algorithm.*

Observe that for every v, the operator $\mathrm{lift}(\cdot, v)$ is a monotone operator with respect to the complete partial ordering where $\rho \leq \rho'$ if $\rho(v) \leq \rho'(v)$ for all $v \in V$.

Jurdziński's algorithm is depicted in Figure 1. The outcome determines the winning set of vertices for each player as follows.

THEOREM 9 (see [17]). *Let G be a parity game. Even has a winning strategy from precisely the vertices v such that, after the lifting algorithm depicted in Figure* 1 *halts, $\rho(v) < \infty$.*

Jurdziński's algorithm needs at most $n'N_G$ iterations of the while loop where

$$N_G = |M_G^\infty| = 1 + \prod_{i:\ 0<2i+1\leq d-1} n_{2i+1}. \tag{25}$$

More precisely, Jurdziński argues as follows. Each vertex can only be lifted N_G 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 $O(m'N_G d)$. In this analysis, it is implicitly assumed that one can, in constant time, decide if there is a vertex v such that $\mathrm{update}(\rho, v) \neq \rho(v)$, and find such a vertex. We provide an implementation of Jurdziński's algorithm that achieves this.

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 $\rho(v)$. This set L is implemented as a list together with a bit array; extracting an element, adding an element, and membership test can then be carried out in constant time. Further, we maintain arrays B and C that store, for each vertex v, the value $\text{best-nghb-ms}(\rho, v)$ and the number of successors u of v with $\langle \rho(u) \rangle_v = \text{best-nghb-ms}(\rho, v)$, denoted $\mathrm{cnt}(\rho, v)$.

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." This is only necessary for $w \in V_0$, because if this is the case we need to be able to realize when all neighbors with the current minimum value have "died out," while for $w \in V_1$ we look at the maximum of all neighbors.

LEMMA 10. *The lifting algorithm depicted in Figure* 2 *computes the function ρ from Jurdziński's algorithm in time $O(m'N_G d)$ and space $O(dm')$.*

Proof. The correctness follows from the above explanation. The running time follows because each vertex can enter L at most $n_1 + 1$ times, and the time taken by the body of the while loop is proportional to the number of edges incident on the vertex.

The bound on the running time can be explained as follows. The initialization (lines 1–4) takes time $O(m'd)$. If a vertex enters L in the body of the while loop, then its ρ-value will be incremented next time the vertex is removed from L. That means every vertex enters L at most N_G times. The time it takes to process a vertex v taken

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

FIG. 2. *Efficient implementation of the lifting algorithm.*

from the loop is $O(\#$ vertices incident on $v)$. This means that lines 5–17 take time $O(m'N_G d)$. This proves the claim. ∎

We can now state one of our main theorems.

THEOREM 11. *For a Büchi automaton A, the relations $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$ can all be computed in time $O(|\Delta||Q|^3)$ and space $O(|Q||\Delta|)$.*

Proof. The theorem follows from Lemmas 4 and 10. Observe that in the (bi)simulation games involved we have $N_G = n_1 + 1 = O(|Q|^2)$. ∎

As mentioned, in prior work $O(|Q|^6)$–time and space [14], and $O(|Q|^{10})$–time and space [15] 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 that (1) quotienting with respect to delayed simulation preserves the recognized language; (2) 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" $\mathrm{cl}(A)$ (unlike what we saw with delayed bisimulation). We first define quotients.

DEFINITION 12. 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

$$A/{\approx} = \langle \Sigma, Q/{\approx}, \Delta_{\approx}, [q_I], F/{\approx} \rangle, \tag{26}$$

where

$$\Delta_{\approx} = \{([q], a, [q']) \mid \exists\ q_0 \in [q],\ q_0' \in [q'] \text{ such that } (q_0, a, q_0') \in \Delta\}. \tag{27}$$

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 needed 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. But first we show that this is true for $\approx_{de}$.

We start with a lemma.

LEMMA 13. *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 $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \dots$ is a finite or infinite run of $A/\approx_{de}$, then there exists a run $q_0' a_0 q_1' a_1 \dots$ 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 $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \dots$ is an infinite run of $A/\approx_{de}$ with $q_0 \in F$, then there exists a finite run $q_0'' a_0 \dots a_{r-1} q_r''$ of A such that $q_j \preceq_{de} q_j''$ for $j \leq r$ and $q_r'' \in F$.*

Proof. For the first part, recall that by definition of $\preceq_{de}$, we know Duplicator wins $G_A^{de}(q_0, q_0')$. Let f be a winning strategy for him, and let $q_1' = f(q_0 q_0' a q_1)$. Then, by definition, $(q_0', a, q_1') \in \delta$. Also, it is easy to see that f' defined by $f'(\rho) = f(q_0 q_0' a \rho)$ is a winning strategy for Duplicator in $G_A^{de}(q_1, q_1')$. Therefore, the claim holds.

For the second part, observe that by definition of $A/\approx_{de}$ there exist $\hat{q}_0$ and $\hat{q}_1$ such that (i) $q_0 \approx_{de} \hat{q}_0$, (ii) $q_1 \approx_{de} \hat{q}_1$, and (iii) $(\hat{q}_0, a_0, \hat{q}_1) \in \Delta$. From (i), we obtain $\hat{q}_0 \preceq_{de} q_0'$ by transitivity of $\preceq_{de}$. So, from (iii) and the first part of the lemma, we can conclude there exists $(q_0', a, q_1') \in \Delta$ such that $\hat{q}_1 \preceq_{de} q_1'$. From (ii), we obtain $q_1 \preceq_{de} q_1'$. Hence, we have constructed the first transition of the desired run and are in a completely analogous situation. The rest follows by induction.

For the third part, first set $q_0' = q_0$. Let $q_0' a_0 q_1' a_1 \dots$ be the infinite run which we know exists by the second part. Next, let f be a winning strategy of Duplicator in $G_A^{de}(q_0', q_0'')$. Consider $q_0'' a_0 q_1'' a_1 \dots$ defined by $q_{i+1}'' = f(q_0' q_0'' a_0 \dots q_i')$. Just as in the proof of the first part, it can be argued that $q_j \preceq_{de} q_j' \preceq_{de} q_j''$ holds for every j. Because of $q_0' = q_0 \in F$, we conclude there exists r such that $q_r'' \in F$, which completes the proof. ∎

THEOREM 14. *For any Büchi automaton A, $L(A) = L(A/\approx_{de})$.*

Proof. To see that $L(A) \subseteq L(A/\approx_{de})$, consider any accepting run $\pi = q_0 a_0 q_1 a_1 \dots$ of A. By definition of $A/\approx_{de}$, $[q_0] a_0 [q_1] a_1 \dots$ is an accepting run of $A/\approx_{de}$. This holds for any of our simulation notions.

To show $L(A/\approx_{de}) \subseteq L(A)$, consider an accepting run $[q_0] a_0 [q_1] a_1 \dots$ of $A/\approx_{de}$. Although we cannot guarantee that $q_0 a_0 q_1 a_1 \dots$ is a run of A, we can construct another accepting run over the same word.

We may assume that $q_0 = q_I$ and that there are infinitely many i such that $q_i \in F$. We construct a sequence ρ_0, ρ_1, ... of finite runs of A on prefixes of $a_0 a_1 \dots$ where ρ_{l+1} strictly extends ρ_l and contains at least $l+1$ elements from F. So the limit of the ρ_i's will be the run we are looking for.

We start with $\rho_0 = q_0$. Assume $\rho_l = q_0' a_0 \dots q_i'$ has already been constructed in such a way that $q_i \preceq_{de} q_i'$. There exists $j > i$ such that $q_j \in F$. So, by the second part of the previous lemma, we know there exists a run $q_i' a_i \dots q_j'$ such that $q_j \preceq_{de} q_j'$. By the third part of the lemma, we know there exists $k \geq j$ and a run $q_j' a_j \dots q_k'$ such that $q_k \preceq_{de} q_k'$ and $q_k' \in F$. We set $\rho_{l+1} = \rho a_i q_{i+1}' \dots q_k'$. ∎

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 $\mathrm{cl}(A)/\approx_{di}$) by an arbitrarily large factor.

PROPOSITION 15. *For $n \geq 2$, there is a Büchi automaton A_n with $n+1$ states*

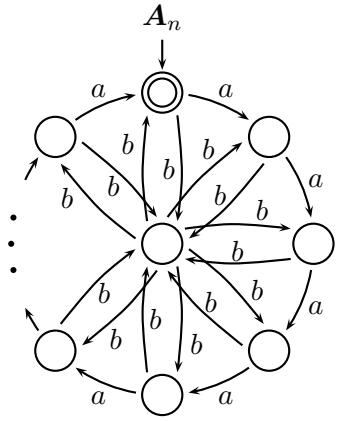

FIG. 3. *Family of automata* A_n.

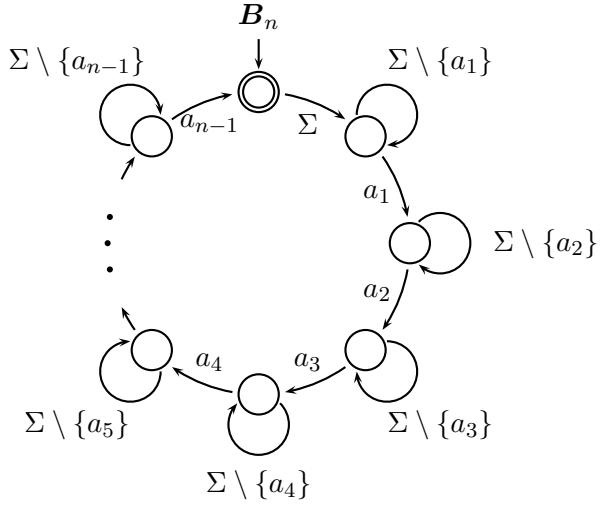

FIG. 4. *Family of automata* B_n.

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, $A_n = \mathrm{cl}(A_n)$, 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. □

Next we see that Theorem 14 fails badly for fair simulation and bisimulation; that is, fair (bi)simulation cannot be used for state space reduction via quotienting under any reasonable definition of quotient. [15] already makes a very closely related observation, showing an automaton whose fair bisimulation quotient is not fair bisimilar to itself.

PROPOSITION 16. *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,* $L(B_n) \neq L(B_n/\approx_f^{bi})$*.*

Proof. Consider the automaton B_n shown in Figure 4. 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 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 presented an algorithm for the new notion of delayed simulation. Our algorithms employ a relatively simple fixed point computation, an implementation of an algorithm by Jurdziński for parity games, and should perform well in practice.

Our own main aim in using simulations is efficient state space reduction, as in [8]. 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 ingredient in making explicit state model checkers such as SPIN more efficient. Preliminary results indicate that in practice delayed simulation does outperform direct simulation on many inputs; further studies need to be carried out to get a clearer picture of the relative advantages of delayed simulation.

REFERENCES

[1] H. R. Andersen, *Model checking and Boolean graphs*, Theoret. Comput. Sci., 126 (1994), pp. 3–30.

[2] B. Bloom and R. Paige, *Transformational design and implementation of a new efficient solution to the ready simulation problem*, Sci. Comput. Programming, 24 (1995), pp. 189–220.

[3] D. Bustan and O. Grumberg, *Checking for Fair Simulation in Models with Büchi Fairness Constraints*, Tech. report TR-CS-2000-13, Technion, Haifa, Israel, 2000.

[4] D. Bustan and O. Grumberg, *Applicability of fair simulation*, Inform. and Comput., 194 (2004), pp. 1–18.

[5] D. Bustan and O. Grumberg, *Simulation-based minimization*, ACM Trans. Comput. Logic, 4 (2003), pp. 181–206.

[6] D. L. Dill, A. J. Hu, and H. Wong-Toi, *Checking for language inclusion using simulation preorders*, in Proceedings of the 3rd International Workshop on Computer Aided Verification, CAV '91, Aalborg, Denmark, 1991, Lecture Notes in Comput. Sci. 575, K. G. Larsen and A. Skou, eds., Springer-Verlag, Berlin, 1992, pp. 255–265.

[7] K. Etessami, *A hierarchy of polynomial-time computable simulations for automata*, in Proceedings of the 13th International Conference of Concurrency Theory, CONCUR 2002, Brno, Czech Republic, 2002, Lecture Notes in Comput. Sci. 2421, L. Brim, P. Jancar, M. Kretínský, and A. Kucera, eds., Springer-Verlag, Berlin, 2002, pp. 131–144.

[8] K. Etessami and G. J. Holzmann, *Optimizing Büchi automata*, in Proceedings of the 11th International Conference of Concurrency Theory, CONCUR 2000, University Park, PA, 2000, Lecture Notes in Comput. Sci. 1877, C. Palamidessi, ed., Springer-Verlag, Berlin, 2000, pp. 153–167.

[9] K. Etessami, R. Schuller, and Th. Wilke, *Fair simulation relations, parity games, and state space reduction for Büchi automata*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete, Greece, 2001, Lecture Notes in Comput. Sci. 2076, F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, Berlin, 2001, pp. 694–707.

[10] C. Fritz and Th. Wilke, *State space reductions for alternating Büchi automata: Quotienting by simulation equivalences*, in Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science, FST TCS 2002, Kanpur, India, 2002, Lecture Notes in Comput. Sci. 2556, M. Agrawal and A. Seth, eds., Springer-Verlag, Berlin, pp. 157–169.

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

[12] E. Grädel, W. Thomas, and Th. Wilke, eds., *Automata, Logics, and Infinite Games: A Guide to Current Research*, Lecture Notes in Comput. Sci. 2500, Springer-Verlag, New York, 2002.

[13] S. Gurumurthy, R. Bloem, and F. Somenzi, *Fair simulation minimization*, in Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002, Copenhagen, Denmark, 2002, Lecture Notes in Comput. Sci. 2404, E. Brinksma and K. Guldstrand Larsen, eds., Springer-Verlag, Berlin, 2002, pp. 610–624.

[14] T. A. Henzinger, O. Kupferman, and S. Rajamani, *Fair simulation*, Inform. and Comput., 173 (2002), pp. 64–81.

[15] T. A. Henzinger and S. Rajamani, *Fair bisimulation*, in Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS 2000, Berlin, Germany, 2000, Lecture Notes in Comput. Sci. 1785, S. Graf and M. I. Schwartzbach, eds., Springer-Verlag, Berlin, 2000, pp. 299–314.

[16] M. Henzinger Rauch, T. A. Henzinger, and P. W. Kopke, *Computing simulations on finite and infinite graphs*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, IEEE, pp. 453–462.

[17] M. Jurdziński, *Small progress measures for solving parity games*, in Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science STACS 2000, Lille, France, 2000, Lecture Notes in Comput. Sci. 1770, H. Reichel and S. Tison, eds., Springer-Verlag, Berlin, 2000, pp. 290–301.

[18] N. Klarlund, *Progress measures, immediate determinacy, and a subset construction for tree automata*, Ann. Pure Appl. Logic, 69 (1994), pp. 243–268.

[19] O. Kupferman and M. Y. Vardi, *Weak alternating automata and tree automata emptiness*, in Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, ACM, New York, pp. 224–233.

[20] R. Milner, *Communication and Concurrency*, Prentice-Hall, New York, 1989.

[21] R. Paige and R. E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput., 16 (1987), pp. 973–989.

[22] F. Somenzi and R. Bloem, *Efficient Büchi automata from LTL formulae*, in Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000, Chicago, IL, 2000, Lecture Notes in Comput. Sci. 1855, E. A. Emerson and A. Prasad Sistla, eds., Springer-Verlag, Berlin, 2000, pp. 248–263.

[23] W. Thomas, *Automata on infinite objects*, in Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 134–191.

[24] W. Thomas, *Languages, automata and logic*, in Handbook of Formal Languages, Vol. 3: Beyond Words, A. Salomaa and G. Rozenberg, eds., Springer-Verlag, Berlin, 1997, pp. 389–455.

[25] I. Walukiewicz, *Completeness of Kozen's axiomatisation of the propositional μ-calculus*, Inform. and Comput., 157 (2000), pp. 142–182.