Solving Games Without Determinization*

Thomas A. Henzinger and Nir Piterman

EPFL, Switzerland

Abstract. The synthesis of reactive systems requires the solution of two-player games on graphs with ω-regular objectives. When the objective is specified by a linear temporal logic formula or nondeterministic Büchi automaton, then previous algorithms for solving the game require the construction of an equivalent *deterministic* automaton. However, determinization for automata on infinite words is extremely complicated, and current implementations fail to produce deterministic automata even for relatively small inputs. We show how to construct, from a given nondeterministic Büchi automaton, an equivalent *nondeterministic* parity automaton $\mathcal{P}$ that is *good for solving games* with objective $\mathcal{P}$. The main insight is that a nondeterministic automaton is good for solving games if it fairly simulates the equivalent deterministic automaton. In this way, we omit the determinization step in game solving and reactive synthesis. The fact that our automata are nondeterministic makes them surprisingly simple, amenable to symbolic implementation, and allows an incremental search for winning strategies.

1 Introduction

One of the most ambitious goals in formal methods is to automatically produce designs from specifications, a process called *synthesis*. We are interested in *reactive systems*, i.e., systems that continuously interact with other programs, users, or their environment (like operating systems or CPUs). The complexity of a reactive system does not arise from computing a complicated function but rather from the fact that it has to be able to react to all possible inputs and maintain its behavior forever. There are two (essentially equivalent) approaches to solving the synthesis problem. The first is by reducing it to the emptiness problem of tree automata [Rab72], and the second, by reducing it to solving infinite-duration two-player games [BL69]. We consider the second view. The two players in the game are the system and its environment. The environment tries to violate the specification and the system tries to satisfy it. The system wins the game if it has a strategy such that all infinite outcomes satisfy the specification. The winning strategy, the way in which the system updates its internal variables, is then translated into an implementation that satisfies the specification when interacting with any possible environment.

More formally, a *game* is a directed graph where the vertices are partitioned between player 0 (system) and player 1 (environment). A *play* proceeds by moving a token along the edges of the graph. If the token is on a vertex of player

* This research was supported in part by the Swiss National Science Foundation.

Z. Ésik (Ed.): CSL 2006, LNCS 4207, pp. 395–410, 2006.

0, she gets to choose to which successor to move the token. If the token is on a vertex of player 1, she chooses the successor. When they continue in this fashion ad infinitum, the token passes an infinite sequence of vertices. We determine who wins the play by looking at this infinite outcome. We define winning plays either by conditions (such as parity or Rabin conditions) on the locations that occur infinitely often along a play, or by recognizers (such as linear temporal logic formulas or Büchi automata) of infinite words over the alphabet of locations. In either case, we are interested in *solving* the game. That is, we wish to determine from which locations of the game, player 0 has a winning strategy, i.e., a way to resolve her decisions so that the resulting plays are winning. For example, when the winning condition is a parity condition [EJ91], the problem of solving the game is in NP∩co-NP [EJS93] and the current best complexity for solving such games is $O(t{\cdot}g^{\lfloor \frac{k}{2} \rfloor})$, where g, t, and k are the number of locations and transitions in the game and priorities in the parity condition, respectively [Jur00, JV00].

In the context of synthesis, we consider an interaction of the system and the environment as winning for the system if it satisfies the specification. Thus, it makes more sense to consider games where the winning condition is given as a *linear temporal logic* (LTL) formula or *nondeterministic Büchi word automaton* (NBW). The way to solve such games is by reducing the problem to the solution of simpler games such as parity or Rabin. As part of this reduction, before taking the product of the game with the winning condition, we have to construct a deterministic automaton for the winning condition. This is because every sequence of choices made in the game has to satisfy the specification.

The first problem we encounter when we come to determinize automata on infinite words is that the Büchi acceptance condition is not strong enough [Lan69]. We have to use stronger acceptance conditions like parity or Rabin. Indeed, Safra suggested a determinization construction that takes an NBW and constructs a deterministic Rabin automaton [Saf88]. Recently, Piterman suggested a variant of this construction with a smaller number of states that results in a deterministic parity automaton [Pit06]. Specifically, starting from an NBW with n states, he constructs a deterministic parity automaton with n^{2n+2} states and $2n$ priorities. When we combine the game with the deterministic automaton, we get a game with $g{\cdot}n^{2n+2}$ locations and $t{\cdot}n^{2n+2}$ transitions, where g and t are the number of locations and transitions in the original game. The overall complexity of solving this game, therefore, is $O(t{\cdot}n^{2n+2}{\cdot}(g{\cdot}n^{2n+2})^n)$. This theory is not applicable in practice, because determinization is extremely complex. Every state of the deterministic automaton is a tree of subsets of states of the original automaton. A transition moves states between different nodes of the tree, adds and removes nodes, and changes the names of the nodes. Only recently, 16 years after the publications of Safra's construction, it was finally implemented [THB95, KB05, ATW05]. These implementations are limited to determinize automata with approximately 10 states[1]. One possible

[1] Piterman's variant maintains the tree structure of Safra that proved hard to implement. It reduces the amount of information associated with every node, however, at the price of giving the nodes dynamic names, which may prove hard to implement.

solution is to consider restricted specifications that can be handled more efficiently (cf. [RW89, RA03, KPP05]). Another possible solution is to use nondeterministic specification automata, which make the approach sound but incomplete [HRS05, JGB05].

Here we do pursue complete solutions for general ω-regular specifications. While we cannot improve the worst-case complexity of synthesis, it is desirable to have an algorithm that performs well in many cases that occur in practice, even if they involve a large number of states. In particular, we wish to use two heuristics that have had great success in formal verification, but cannot be used when applying determinization. The first is to reason *symbolically* about sets of states, rather than explicitly about individual states [McM93]. Using a symbolic state representation in Safra's construction seems difficult. Second, we wish to be able to find a winning strategy in a game that uses a small amount of memory, if such a strategy exists. The memory used by a strategy corresponds to the number of states of a parity or Rabin specification automaton. Thus, small memory is not possible if we construct the deterministic automaton as the first step of the synthesis algorithm. Instead, we want to *incrementally* increase, as much as necessary, the memory provided to strategies.

For this purpose we propose a general solution that does not involve determinization. We define *good for games* automata (GFG, for short), which are the class of nondeterministic automata that can be used in the context of games. The main idea is that if an automaton can resolve its nondeterminism in a stepwise fashion, then it is good enough for reasoning about games. The formal definition of a GFG automaton considers a game played on the structure of the automaton in which the opponent chooses input letters, one at a time, and the automaton resolves its nondeterminism for each input letter. The automaton wins if whenever the infinite word chosen by the opponent is in the language of the automaton, then the run chosen by the automaton is accepting. The automaton is GFG if it has a winning strategy in this game. We show that a nondeterministic specification automaton with this property can indeed be used for solving games without prior determinization. That is, in the product of a game with a GFG automaton, the winning states correspond to the winning states of the original game. In order to check if an automaton is GFG, we give an alternative characterization: an automaton is GFG iff it fairly simulates [HKR97] a deterministic automaton for the same language.

Our main contribution is a construction that takes an NBW and produces a GFG automaton for the same language. Given an NBW with n states, we construct a nondeterministic parity automaton with $2^n \cdot n^{2n}$ states and $2n$ priorities. The resulting overall complexity is $O(t \cdot (2^n \cdot n^{2n})^2 \cdot (g \cdot 2^n \cdot n^{2n})^n)$ for synthesis. We generalize the $n!$ lower bound on determinization [Mic88] to the size of GFG automata, establishing that our construction is essentially optimal.

The most important feature of our nondeterministic GFG automaton is its simplicity. The automaton basically follows n different sets of subsets of the original automaton. This leads to a simple structure and even simpler transitions, which are amenable to symbolic implementations. Another attractive advantage

of this approach is that it offers a natural hierarchy of nondeterministic automata of increasing complexity that converge to the full GFG solution. That is, given a game and an NBW specification automaton, we can try first solving the game with a small automaton for the winning condition. If we succeed, we are done, having found a winning strategy with small memory for the particular game we are solving. If we fail, we increase the size of the automaton (and thus the memory size we consider), and try again. In the worst case, we get to the full GFG construction, whose memory suffices to win every game with that winning condition. If the GFG automaton fails, then we know that the original specification is not realizable. In Section 6, we give a family of game graphs and winning conditions for which this incremental approach indeed leads to considerable savings.

In addition, simple modifications of our construction lead to nondeterministic parity automata whose number of states ranges between n^{2n} and n^{3n}. Using the smallest possible automaton, the theoretical upper bound reduces to $O(t{\cdot}(n^{2n})^2{\cdot}(g{\cdot}n^{2n})^n)$, which almost matches the upper bound using the deterministic automaton. Recall that our automata are intended for symbolic implementation. Thus, it makes no sense to count the exact upper bound but rather to check which variant of the construction works best in practice. In addition, our hope for synthesis is that in many practical cases it would perform better than the worst-case theoretical upper bound. Using our construction it is possible to search for smaller strategies. Indeed, even for small values of n the time complexity $O(n^{2n^2})$ is impossible.

Recently, Kupferman and Vardi suggested another construction that avoids determinization in certain situations [KV05]. Their algorithm shows how to solve the emptiness problem of alternating parity tree automata through a reduction to the emptiness problem of nondeterministic Büchi tree automata. In order to use their construction to solve games, one has to be able to express the winning condition of the opponent by an NBW. Thus, their algorithm can be applied to synthesis for LTL specifications, because given an LTL winning condition, we negate the LTL formula to get the winning condition of the opponent. On the other hand, when the winning condition is given as an NBW, there is no easy way to complement it, and their algorithm cannot be applied. Furthermore, the worst-case complexity of their algorithm may be quadratically worse, and the size of the produced strategy may be exponentially larger than our algorithm.

2 Preliminaries

Nondeterministic Automata. A *nondeterministic automaton* is $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$, where Σ is a finite alphabet, S is a finite set of states, $\delta : S \times \Sigma \to 2^S$ is a transition function, $s_0 \in S$ is an initial state, and α is an acceptance condition to be defined below. A *run* of $\mathcal{N}$ on a word $w = w_0 w_1 \ldots$ is an infinite sequence of states $t_0 t_1 \ldots \in S^\omega$ such that $t_0 = s_0$ and for all $i \geq 0$, we have $t_{i+1} \in \delta(t_i, w_i)$. For a run $r = t_0 t_1 \ldots$, let $inf(r) = \{s \in S \mid s = t_i \text{ for infinitely many } i\text{'s}\}$ be the set of all states occurring infinitely often in the

run. We consider two acceptance conditions. A *parity* condition α is a partition $\{F_0, \ldots, F_k\}$ of S. We call k the *index* of the parity condition. The run r is *accepting* according to the parity condition α if for some even i we have $inf(r) \cap F_i \neq \emptyset$, and for all $j < i$, we have $inf(r) \cap F_j = \emptyset$. That is, the minimal set that is visited infinitely often is even. A *Büchi* condition is a set $F \subseteq S$ of states. The run r is *accepting* according to the Büchi condition F if $inf(r) \cap F \neq \emptyset$. That is, the run visits infinitely often states from F. A word w is *accepted* by $\mathcal{N}$ if there exists some accepting run of $\mathcal{N}$ over w. The *language* of $L(\mathcal{N})$ is the set of words accepted by $\mathcal{N}$. Two automata $\mathcal{N}_1$ and $\mathcal{N}_2$ are *equivalent* if they have the same language, i.e., $L(\mathcal{N}_1) = L(\mathcal{N}_2)$.

Given a set $S' \subseteq S$ of states and a letter $\sigma \in \Sigma$, we denote by $\delta(S', \sigma)$ the set $\bigcup_{s \in S'} \delta(s, \sigma)$. The automaton $\mathcal{N}$ is *deterministic* if for every state $s \in S$ and letter $\sigma \in \Sigma$, we have $|\delta(s, \sigma)| = 1$. In that case we write $\delta : S \times \Sigma \to S$.

We use the acronyms NBW, DPW, and NPW to denote automata. NBW stands for nondeterministic Büchi word automaton, DPW for deterministic parity word automaton, and NPW for nondeterministic parity word automaton.

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

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

A *strategy* for player 0 is a partial function $f : V^* \times V_0 \to V$ such that if $f(\pi \cdot v)$ is defined, then $(v, f(\pi \cdot v)) \in \rho$. A play $\pi = v_0 v_1 \ldots$ is *f-conform* if whenever $v_i \in V_0$, we have $v_{i+1} = f(v_0 \ldots v_i)$. The strategy f is *winning from* a location $v \in V$ if every f-conform play that starts in v is winning for player 0. We say that player 0 *wins from* v if she has a winning strategy from v. The *winning region* of player 0 is the set of locations from which player 0 wins. We denote the winning region of player 0 by W_0. Strategies, winning strategies, winning, and winning regions are defined dually for player 1. We *solve* a game by computing the winning regions W_0 and W_1. For the kind of games considered in this paper, W_0 and W_1 form a partition of V [Mar75].

We consider parity winning conditions. A parity condition α is a partition $\{F_0, \ldots, F_k\}$ of V. The parity condition α defines the set W of infinite plays in G such that the minimal set that is visited infinitely often is even, i.e., $\pi \in W$ if there exists an even i such that $inf(\pi) \cap F_i \neq \emptyset$, and for all $j < i$, we have $inf(\pi) \cap F_j = \emptyset$. The complexity of solving parity games is as follows:

Theorem 1. [Jur00] *Given a parity game G with g locations, t transitions, and index k, we can solve G in time $O(t \cdot g^{\lfloor \frac{k}{2} \rfloor})$.*

We are also interested in more general winning conditions. We define W using an NBW over the alphabet V (or some function of V). Consider a game $G = \langle V, V_0, V_1, \rho, W \rangle$ and an NBW $\mathcal{N}$ over the alphabet V such that $W = L(\mathcal{N})$. We abuse notation and write $G = \langle V, V_0, V_1, \rho, \mathcal{N} \rangle$, or just $G = \langle V, \rho, \mathcal{N} \rangle$.

The common approach to solving such games is by reducing them to parity games. Consider a game $G = \langle V, V_0, V_1, \rho, W\rangle$ and a deterministic automaton $\mathcal{D} = \langle V, S, \delta, s_0, \alpha\rangle$, whose alphabet is V such that $L(\mathcal{D}) = W$. We define the *product* of G and $\mathcal{D}$ to be the game $G \times \mathcal{D} = \langle V \times S, V_0 \times S, V_1 \times S, \rho', W'\rangle$ where $((v, s), (v', s')) \in \rho'$ iff $(v, v') \in \rho$ and $s' = \delta(s, v)$ and W' contains all plays whose projections onto the second component are accepting according to α. A *monitor for* G is a deterministic automaton $\mathcal{D}$ such that for all locations v of G, player 0 wins from v in G iff player 0 wins from (v, s_0) in $G \times \mathcal{D}$.

The common way to solve a game $G = \langle V, \rho, \mathcal{N}\rangle$ where $\mathcal{N}$ is an NBW, is by constructing an equivalent DPW $\mathcal{D}$ [Pit06] and solving the product game $G \times \mathcal{D}$. Unfortunately, determinization has defied implementation until recently, and it cannot be implemented symbolically [THB95, ATW05, KB05]. This means that theoretically we know very well how to solve such games, however, practically we find it very difficult to do so. Formally, we have the following.

Theorem 2. *Consider a game G whose winning condition $\mathcal{N}$ is an NBW with n states. We can construct a DPW $\mathcal{D}$ equivalent to $\mathcal{N}$ with n^{2n+2} states and index $2n$. The parity game $G \times \mathcal{D}$ can be solved in time $O(t{\cdot}n^{2n+2}{\cdot}(g{\cdot}n^{2n+2})^n)$, where g and t are the number of locations and transitions of G.*

It is common wisdom that nondeterministic automata cannot be used for game monitoring. In this paper we show that this claim is false. We define nondeterministic automata that can be used for game monitoring; we call such automata *good for games* (GFG). Our main result is a construction that takes an NBW and produces a GFG NPW. Though our NPW may be slightly bigger than the equivalent DPW (see Section 5), it is much simpler, amenable to symbolic implementation, and suggests a natural hierarchy of automata of increasing complexity that lead to the full solution.

3 Good for Games Automata

In this section we define when an automaton can be used as a monitor for games. In order to define GFG automata, we consider the following game. Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha\rangle$ be an automaton. The *monitor game* for $\mathcal{N}$ is a game played on the set S of states. The game proceeds in rounds in which player 1 chooses a letter, and player 0 answers with a successor state reading that letter. Formally, a *play* is a maximal sequence $\pi = t_0\sigma_0 t_1\sigma_1 \dots$ such that $t_0 = s_0$ and for all $i \geq 0$, we have $t_{i+1} \in \delta(t_i, \sigma_i)$. That is, an infinite play produces an infinite word $w(\pi) = \sigma_0\sigma_1 \dots$, and a run $r(\pi) = t_0 t_1 \dots$ of $\mathcal{N}$ on $w(\pi)$. A play π is *winning* for player 0 if π is infinite and, in addition, either $w(\pi)$ is not in $L(\mathcal{N})$ or $r(\pi)$ is an accepting run of $\mathcal{N}$ on $w(\pi)$. Otherwise, player 1 wins. That is, player 0 wins if she never gets stuck and, in addition, either the resulting word constructed by player 1 is not in the language or (the word is in the language and) the resulting run of $\mathcal{N}$ is accepting.

A *strategy* for player 0 is a partial function $f : (S{\times}\Sigma)^+{\rightarrow}S$ such that if $f(\pi{\cdot}(s, \sigma))$ is defined, then $f(\pi{\cdot}(s, \sigma)) \in \delta(s, \sigma)$. A play $\pi = t_0\sigma_0 t_1\sigma_1 \dots$ is *f-conform* if for all $i \geq 0$, we have $t_{i+1} = f(t_0 \dots \sigma_i)$. The strategy f is *winning*

from a state $s \in S$ if every f-conform play that starts in s is winning for player 0. We say that *player 0 wins from s* if she has a winning strategy from s. The automaton $\mathcal{N}$ is good for games (GFG) if player 0 wins from the initial state s_0.

We show how to use GFG automata for game monitoring. Consider a GFG automaton $\mathcal{N} = \langle V, S, \delta, s_0, \alpha \rangle$ and a game $G = \langle V, V_0, V_1, E, \mathcal{N} \rangle$. We construct the following extended product game. Let $G \otimes \mathcal{M} = \langle V', V_0', V_1', E', W' \rangle$, where the components of $G \otimes \mathcal{N}$ are as follows:

- $V' = V \times S \times \{0, 1\}$, where $V_0' = (V \times S \times \{0\}) \cup (V_0 \times S \times \{1\})$ and $V_1' = V_1 \times S \times \{1\}$.
  Given a location $(v, s, i) \in V'$, let $v \Downarrow_S = s$ be the projection onto the state in S. We extend $\Downarrow_S$ to sequences of locations in the natural way.
- $E' = \{((v, s, 0), (v, s', 1)) \mid s' \in \delta(s, v)\} \cup \{((v, s, 1), (v', s, 0)) \mid (v, v') \in E\}$.
- $W' = \{\pi \in V'^{\omega} \mid \pi \Downarrow_S$ is accepting according to $\alpha\}$.

W.l.o.g., we assume that the acceptance condition of $\mathcal{N}$ is closed under finite stuttering (which is true for Büchi and parity). When $\mathcal{N}$ is GFG, we can use $G \otimes \mathcal{N}$ to solve G.

Theorem 3. *Player 0 wins from location v in the game G iff she wins from location $(v, s_0, 0)$ in the game $G \otimes \mathcal{N}$, where s_0 is the initial state of $\mathcal{N}$.*

A win in $G \otimes \mathcal{N}$ is easily translated into a win in G by forgetting the $\mathcal{N}$ component. In the other direction, a winning strategy in G is combined with a winning strategy in the monitor game for $\mathcal{N}$ to produce a strategy in $G \otimes \mathcal{N}$. As the strategy in G is winning, the projection of a resulting play onto the locations of G is a word accepted by $\mathcal{N}$. As the strategy in the monitor game is winning, the projection of the play onto the states of $\mathcal{N}$ is an accepting run of $\mathcal{N}$.

4 Checking the GFG Property

In this section we suggest one possible way of establishing that an automaton is GFG. We prove that an automaton is GFG by showing that it fairly simulates another GFG automaton for the same language. By definition, every deterministic automaton is GFG. This follows from the fact that player 0 does not have any choices in the monitor component of the game. Hence, if an automaton fairly simulates the deterministic automaton for the same language, it is GFG.

4.1 Fair Simulation

We define *fair simulation* [HKR97]. Consider two automata $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ and $\mathcal{R} = \langle \Sigma, T, \eta, t_0, \beta \rangle$ with the same alphabet. In order to define fair simulation, we define the *fair-simulation game*. Let $G_{\mathcal{N},\mathcal{R}} = \langle V, V_0, V_1, \rho, W \rangle$ be the game with the following components:

- $V = (S \times T) \cup (S \times T \times \Sigma)$, where $V_0 = S \times T \times \Sigma$ and $V_1 = S \times T$.
- $\rho = \{((s, t), (s', t, \sigma)) \mid s' \in \delta(s, \sigma)\} \cup \{((s, t, \sigma), (s, t')) \mid t' \in \eta(t, \sigma)\}$.

Given an infinite play π of $G_{\mathcal{N},\mathcal{R}}$ we define π_1 to be the projection of π onto the states in S and π_2 the projection of π onto the states in T. Player 0 *wins* a play π if π is infinite and whenever π_1 is an accepting run of $\mathcal{N}$, then π_2 is an accepting run of $\mathcal{R}$ (w.l.o.g., the acceptance conditions α and β are closed under finite stuttering). If player 0 wins the fair-simulation game from a location (s,t), then the state t of $\mathcal{R}$ *fairly simulates* the state s of $\mathcal{N}$, denoted by $s \leq_f t$. If $s_0 \leq_f t_0$ for the initial states s_0 and t_0, then $\mathcal{R}$ *fairly simulates* $\mathcal{N}$, denoted $\mathcal{N} \leq_f \mathcal{R}$.

4.2 Proving an Automaton GFG

Here we show that if an automaton $\mathcal{N}$ fairly simulates a GFG automaton $\mathcal{D}$ for the same language, then $\mathcal{N}$ is a GFG automaton as well.

Theorem 4. *Let $\mathcal{N}$ be a nondeterministic automaton and $\mathcal{D}$ a GFG automaton equivalent to $\mathcal{N}$. Then $\mathcal{D} \leq_f \mathcal{N}$ implies $\mathcal{N}$ is GFG.*

Proof. Let $\mathcal{N}=\langle \Sigma, N, \delta, n_0, \alpha\rangle$ and $\mathcal{D}=\langle \Sigma, D, \eta, d_0, \beta\rangle$. Assume that $\mathcal{D} \leq_f \mathcal{N}$. Let $G_{\mathcal{D},\mathcal{N}} = \langle V, V_0, V_1, \rho, W\rangle$ be the fair-simulation game between $\mathcal{D}$ and $\mathcal{N}$. Let $f : V^* \times V_0 \to V$ be a winning strategy for player 0 in $G_{\mathcal{D},\mathcal{N}}$. We denote the monitor game for $\mathcal{D}$ by G_1, and the monitor game for $\mathcal{N}$ by G_2. Let $h : (D \times \Sigma)^+ \to D$ be the winning strategy of player 0 in G_1. We compose f and h to resolve the choices of player 0 in G_2: we use the choices of player 1 in G_2 to simulate choices of player 1 in G_1, and then h instructs us how to simulate player 1 in $G_{\mathcal{D},\mathcal{N}}$, and the choice of f in $G_{\mathcal{D},\mathcal{N}}$ translates to the choice of player 0 in G_2. Accordingly, we construct plays in the three games that adhere to the following invariants:

- The plays in $G_{\mathcal{D},\mathcal{N}}$ and G_1 are f-conform and h-conform, respectively.
- The projection of the play in G_2 onto Σ is the projection onto Σ of the plays in G_1 and $G_{\mathcal{D},\mathcal{N}}$.
- The projection of the play in G_1 onto the states of $\mathcal{D}$ is the projection of the play in $G_{\mathcal{D},\mathcal{N}}$ onto the states of $\mathcal{D}$.
- The projection of the play in $G_{\mathcal{D},\mathcal{N}}$ onto the states of $\mathcal{N}$ is the projection of the play in G_2 onto the states of $\mathcal{N}$.

We call such plays *matching*. Consider the following plays of length one. The initial position in G_1 is d_0, The initial position in G_2 is n_0, and the initial position in $G_{\mathcal{D},\mathcal{N}}$ is (d_0, n_0). Obviously, these are matching plays.

Let $\pi_2 = n_0\sigma_0\ n_1\sigma_1\ \ldots\ n_i$ be a play in G_2, let $\pi_1 = d_0\sigma_0\ d_1\sigma_1\ \ldots\ d_i$ be a play in G_1, and let $\pi_s = (d_0,n_0)\ (d_1,n_0,\sigma_0)\ (d_1,n_1)\ \ldots\ (d_i,n_i)$ be a play in $G_{\mathcal{D},\mathcal{N}}$. Assume that π_1, π_2, and π_s are matching. Let σ_i be the choice of player 1 in G_2. We set d_{i+1} to $h(\pi_1\sigma_i)$, and set $\pi'_1 = \pi_1\sigma_i d_{i+1}$. Let (d_{i+1}, n_{i+1}) be $f(\pi_s(d_{i+1}, n_i, \sigma_i))$, and set $\pi'_s = \pi_s(d_{i+1}, n_i, \sigma_i)(d_{i+1}, n_{i+1})$. Finally, we play n_{i+1} in G_2. By definition of $G_{\mathcal{D},\mathcal{N}}$, it follows that $n_{i+1} \in \delta(n_i, \sigma_i)$. The plays π'_1, π'_s, and π'_2 are matching. Clearly, we can extend the plays according to this strategy to infinite plays.

Let π_1, π_s, and π_2 be some infinite plays constructed according to the above strategy. Let w be the projection of π_2 onto Σ. If $w \notin L(\mathcal{N})$, then player 0 wins in G_2. Assume that $w \in L(\mathcal{N})$. As h is a winning strategy in G_1, we conclude that the projection of π_1 onto D is an accepting run of $\mathcal{D}$. As f is a winning strategy in $G_{\mathcal{D},\mathcal{N}}$, we conclude that the projection of π_s onto N is also accepting. As the projections of π_2 and π_s onto N are equivalent, we are done.

The above condition is not only sufficient, but also necessary. Given two equivalent GFG automata, we can use the strategies in the respective monitor games to construct a winning strategy in the fair-simulation game. In fact, all GFG automata that recognize the same language fairly simulate each other.

5 Constructing GFG Automata

In this section we describe our main contribution, a construction of a GFG automaton for a given language. We start with an NBW and end up with a GFG NPW. In order to prove that our NPW is indeed a GFG, we prove that it fairly simulates the DPW for the same language.

5.1 From NBW to NPW

The idea behind the construction of the NPW is to mimic the determinization construction [Pit06]. We replace the tree structure by nondeterminism. We simply follow the sets maintained by the Safra trees without maintaining the tree structure. In addition we have to treat acceptance. This is similar to the conversion of alternating Büchi word automata to NBW [MH84]: a subset is marked accepting when *all* the paths it follows visit the acceptance set at least once; when this happens we start again. The result is a simple GFG NPW.

Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ be an NBW such that $|S| = n$. We construct a GFG NPW $\mathcal{P} = \langle \Sigma, Q, \eta, q_0, \alpha' \rangle$ with the following components.

- The set Q of states is an n-tuple of annotated subsets of S.
  Every state in a subset is annotated by 0 or 1. The annotation 1 signifies that this state is reachable along a path that visited the acceptance set α of $\mathcal{N}$ recently. When a state s is annotated 1, we say that it is *marked*, and when it is annotated 0, we say that it is *unmarked*. Such an annotated subset can be represented by an element $C \in \{0,1,2\}^S$, where $C(s) = 0$ means that s is not in the set, $C(s) = 1$ means that s is in the set and is unmarked, and $C(s) = 2$ means that s is in the set and is marked. For simplicity of notation, we represent such an annotated set C by a pair $(A, B) \in 2^S \times 2^S$ of sets with $B \subseteq A$, such that $s \in B$ means $C(s) = 2$, $s \in A - B$ means $C(s) = 1$, and $s \notin A$ means $C(s) = 0$. We abuse notation and write $(A, B) \in \{0,1,2\}^S$. We write $(A, B) \subseteq (C, D)$ to denote $A \subseteq C$ and $B \subseteq D$. We sometimes refer to an annotated set as a set.

  In addition, we demand that a set is contained in the B part of some previous set and disjoint from all sets between the two. If some set is empty, then all sets after it are empty as well. Formally,

$$Q = \left\{ \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle \;\middle|\; \begin{array}{l} \forall i \,.\, (A_i, B_i) \in \{0,1,2\}^S, \\ \forall i \,.\, A_i = \emptyset \text{ implies } A_{i+1} = \emptyset, \\ \forall i < j \,.\, \left[\begin{array}{l} \text{either } A_i \cap A_j = \emptyset \\ \text{or } A_j \subseteq B_i \end{array} \right] \end{array} \right\}.$$

- $q_0 = \langle (\{s_0\}, \{s_0\} \cap \alpha), (\emptyset, \emptyset), \ldots, (\emptyset, \emptyset) \rangle$.
  That is , the first set is initialized to the set that contains the initial state of $\mathcal{N}$. All other sets are initialized to the empty set.
- In order to define the transition function η we need a few definitions.
  For $(A, B) \in \{0,1,2\}^S$, $\sigma \in \Sigma$, and $i \in \{0,1\}$, let $succ((A, B), \sigma, i)$ denote the set defined as follows:

$$succ((A,B),\sigma,i) = \begin{cases} \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \delta(A,\sigma) \text{ and} \\ B' \subseteq (\delta(B,\sigma) \cap A') \cup (A' \cap \alpha) \end{array} \right\} & \text{If } B \neq A \text{ and } i=0 \\ \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \delta(A,\sigma) \text{ and} \\ B' \subseteq A' \cap \alpha \end{array} \right\} & \text{If } B=A \text{ and } i=0 \\ \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \delta(A,\sigma) \text{ and} \\ B' \subseteq A' \end{array} \right\} & \text{If } i=1 \end{cases}$$

  That is, given a set $(A, B) \subseteq \{0,1,2\}^S$, the possible successors (A', B') are subsets of the states reachable from (A, B). We add to the marked states all visits to α, and if all states are marked (that is, if $A=B$), then we unmark them.[2] In the case that $i = 1$, we are completely free in the choice of B'.

  For $(A, B), (C, D) \in \{0,1,2\}^S$ and $\sigma \in \Sigma$, let $trans((A, B), \sigma, (C, D))$ be:

$$trans((A,B),\sigma,(C,D)) = \begin{cases} succ((A,B),\sigma,0) & \text{If } A \neq \emptyset \\ succ((C,D),\sigma,1) & \text{If } A = \emptyset \end{cases}$$

  That is, we may choose a successor of either (A, B) or (C, D). We may use (C, D) only if (A, B) is empty. In this case, we may choose to initialize the set of markings as we wish. As $succ((A, B), \sigma, i)$ includes every subset of $\rho(A, \sigma)$, it is always possible to choose the empty set, and in the next step, to choose a subset of (the successors of) (C, D).

  The transition function η is defined for every state $q \in Q$ and letter $\sigma \in \Sigma$ as follows. Let $q = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$. Then:

$$\eta(q,\sigma) = Q \cap \prod_{i=1}^{n} trans((A_i, B_i), \sigma, (A_1, B_1))$$

  Intuitively, (A_1, B_1) holds the set of states that are reachable from the initial state. The other sets correspond to guesses as to which states from (A_1, B_1) to follow in order to ignore the non-accepting runs. Whenever one of the

[2] The decision to allow the set B to decrease nondeterministically may seem counter-intuitive. This is equivalent to 'forgetting' that some of the followed paths visited α. This is more convenient and allows more freedom. It also simplifies proofs.

sets becomes empty, it can be *loaded* by a set of successors of (A_1, B_1). It follows that in order to change a guess, the automaton has to empty the respective set and in the next move load a new set. Notice that emptying a set forces the automaton to empty all the sets after it and load them again from (A_1, B_1).

– Consider a state $q = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$. We define $ind_E(q)$ to be the minimal value k in $[2..n]$ such that $A_k = \emptyset$, or $n+1$ if no such value exists. Formally, $ind_E(q) = min\{k, n+1 \mid 1 < k \leq n \text{ and } A_k = \emptyset\}$. Similarly, $ind_F(q)$ is the minimal value k in $[2..n]$ such that $A_k = B_k$ and $A_k \neq \emptyset$, or $n+1$ if no such value exists. Formally, $ind_F(q) = min\{k, n+1 \mid 1 < k \leq n \text{ and } A_k = B_k \neq \emptyset\}$.

  The parity condition α' is $\langle F_0, \ldots, F_{2n-1} \rangle$, where

  - $F_0 = \{q \in Q \mid A_1 = B_1 \text{ and } A_1 \neq \emptyset\}$;
  - $F_{2i+1} = \{q \in Q \mid ind_E(q) = i+2 \text{ and } ind_F(q) \geq i+2\}$;
  - $F_{2i+2} = \{q \in Q \mid ind_F(q) = i+2 \text{ and } ind_E(q) > i+2\}$.

  As all sets greater than $ind_E(q)$ are empty, the odd sets require that for all sets $A_i \neq B_i$ or $A_i = \emptyset$. In these cases $ind_F(q) = n+1$. Notice that we do not consider the case that (A_1, B_1) is empty. This is a rejecting sink state.

This completes the definition of $\mathcal{P}$.

We first show that $\mathcal{N}$ and $\mathcal{P}$ are equivalent. We show that $L(\mathcal{P})$ contains $L(\mathcal{N})$ by tracing the runs of $\mathcal{N}$. For each run r of $\mathcal{N}$, we use the first set in a state of $\mathcal{P}$ to follow singletons from r. The proof that $L(\mathcal{P})$ is contained in $L(\mathcal{N})$ is similar to the proof that the DPW constructed by Piterman is contained in the language of $\mathcal{N}$ [Pit06].

Lemma 1. *$L(\mathcal{P}) = L(\mathcal{N})$.*

Let $\mathcal{D}$ be the DPW constructed by Piterman [Pit06]. We show that $\mathcal{P}$ fairly simulates $\mathcal{D}$. The proof proceeds by showing how to choose a state of $\mathcal{P}$ that maintains the same sets as labels of the nodes in Safra trees, but without maintaining the parenthood function. In fact, $\mathcal{D}$ also fairly simulates $\mathcal{P}$. This follows immediately from the equivalence of the two and $\mathcal{D}$ being deterministic [HKR97].

Lemma 2. *$\mathcal{D} \leq_f \mathcal{P}$.*

5.2 Complexity Analysis

We count the number of states of the GFG automaton $\mathcal{P}$ and analyze the complexity of using it for solving games.

Theorem 5. *Given an NBW $\mathcal{N}$ with n states, we can construct an equivalent GFG NPW $\mathcal{P}$ with $2^n \cdot n^{2n}$ states and index $2n$.*

Proof. We represent a state of $\mathcal{P}$ as a tree of subsets (or sometimes a forest). The pair (A_i, B_i) is a son of the pair (A_j, B_j) such that $A_i \subseteq B_j$. This tree structure

is represented by the parenthood function $p : [n] \to [n]$ (here $[n]$ is $\{1, \ldots, n\}$). We map every state of $\mathcal{N}$ to the minimal node in the tree (according to the parenthood function) to which it belongs. Thus, the partition into $A_1, \ldots, A_n$ is represented by a function $l : S \to [n]$. Every state of $\mathcal{N}$ that appears in a pair (A_j, B_j) and also in some son (A_i, B_i) belongs to B_j. In addition, we have to remember all states of $\mathcal{N}$ that appear in some set A_i, in no descendant of A_i, and also appear in B_i. It suffices to remember the subset of all these states.

To summarize, there are at most n^n parenthood functions, n^n state labelings, and 2^n subsets of S. This gives a total of $2^n \cdot n^{2n}$ states for $\mathcal{P}$.

We note that the GFG NPW is larger than the DPW constructed in [Pit06] by a factor of 2^n. However, the NPW is much simpler than the DPW. Although Piterman's variant is slightly simpler than Safra's construction, it still maintains the tree structure that proved hard to implement. Existing implementations of Safra's construction [ATW05, KB05] enumerate the states. We believe that this would be the case also with Piterman's variant. The structure of the NPW above is much simpler and amenable to symbolic methods. In order to represent a set of NBW states, we associate a Boolean variable with every state of the NBW. A BDD over n variables can represent a set of sets of states. In order to represent tuples of n sets, we need a BDD over n^2 variables.

We note that very simple modifications can be made to the NPW without harming its GFG structure. We could remove the restrictions on the containment order between the labels in the sets, or tighten them to be closer to the restrictions imposed on the trees in the DPW. This would result in increasing or reducing the number of states between n^{2n} and n^{3n}. The best structure may depend not on the final number of states, but rather on which structure is most efficiently represented symbolically. It may be the case that looser structures may have a better symbolic representation and work better in practice.

We compare the usage of our automata in the context of game solving to other methods. Consider a game $G = \langle V, V_0, V_1, \rho, W \rangle$, where W is given by an NBW $\mathcal{N} = \langle V, S, \delta, s_0, \alpha \rangle$. Let $|S| = n$, and let g and t be the number of locations and transitions of G, respectively. Using Piterman's construction, we construct a DPW $\mathcal{P}$ with n^{2n+2} states and index $2n$. According to Theorem 1, we can solve the resulting parity game in time $O(t \cdot n^{2n+2} \cdot (g \cdot n^{2n+2})^n)$. When we combine our GFG NPW with the game G, the resulting structure may have $t \cdot (2^n \cdot n^{2n})^2$ transitions and $g \cdot 2^n \cdot n^{2n}$ states. That is, we can solve the resulting parity game in time $O(t \cdot 2^{2n} \cdot n^{4n} \cdot (g \cdot 2^n \cdot n^{2n})^n)$. Note also that the construction of Kupferman and Vardi cannot be applied directly [KV05]. This is because Kupferman and Vardi's construction requires an NBW for the complement of the winning condition. On the other hand, in the context of LTL games (i.e., games with LTL winning conditions), Kupferman and Vardi's construction can be applied. Their construction leads to a time complexity of $O(t \cdot n^{2n+2} \cdot (g \cdot n^{2n+2})^{2n})$, with $2n$ in the exponent instead of n. The memory used by the extracted strategy is bounded by $2^{O(n^2)}$ while it is bounded by $2^n \cdot n^{2n}$ in our case.

We note that for checking the emptiness of alternating parity tree automata, our GFG construction cannot be applied. The reason is similar to the reason

why Kupferman and Vardi's method cannot be used for solving games with NBW winning conditions. In this case, we have to construct a GFG NPW for the complement language, which we do not know how to do.

We can use the lower bound on the memory needed for winning strategies to show that our construction is in some sense optimal. For this purpose, we generalize Michel's lower bound on the size of determinization [Mic88, Löd98]. That is, we construct a game with an NBW acceptance condition whose winning strategies require $n!$ memory. Given that our GFG automaton can be used as the memory of a winning strategy and that the resulting game is a parity game that requires no additional memory, this proves that every GFG automaton for the given language has at least $n!$ states.

6 Incremental Construction

Our automata have a natural incremental structure. We simply choose how many sets of states to follow in a state of the GFG automaton. Consider a game $G = \langle V, \rho, \mathcal{N} \rangle$, where $\mathcal{N}$ is an NBW. Let $\mathcal{N}'$ be a nondeterministic automaton such that $L(\mathcal{N}) = L(\mathcal{N}')$ and let s_0 be the initial state of $\mathcal{N}'$. It is simple to see that if player 0 wins from $(v, s_0, 0)$ in $G \otimes \mathcal{N}'$, then player 0 wins from v in G. Indeed, this is the basis of the incomplete approaches described in [HRS05, JGB05]. Using this fact, we suggest the following incremental approach to solving games with ω-regular winning conditions.

Let n be the number of states of $\mathcal{N}$. We apply the construction from Section 5 on $\mathcal{N}$ but use only 2 sets (i.e., restrict the sets $3, \ldots, n$ to the empty set), let $\mathcal{P}_1$ denote this automaton. We then solve the game $G \otimes \mathcal{P}_1$. It is simple to see that $\mathcal{P}_1$ is equivalent to $\mathcal{N}$. Thus, if $(v, q_0, 0)$ is winning for player 0 in $G \otimes \mathcal{P}_1$ (where q_0 is the initial state of $\mathcal{P}_1$), then v is winning for player 0 in G. It follows, that by solving $G \otimes \mathcal{P}_1$ we recognize a subset $W_0' \subseteq W_0$ of the winning region of player 0 in G. Sometimes, we are not interested in the full partition of G to W_0 and W_1, we may be interested in a winning strategy from some set of initial locations in G. If this set of locations is contained in W_0', then we can stop here. Otherwise, we try a less restricted automaton with 3 sets, then 4 sets, etc. For every number of sets used, the resulting automaton may not be GFG, but it recognizes the language of $\mathcal{N}$. A strategy winning in the combination of G and such an automaton is winning also in the original game G (with winning condition $\mathcal{N}$). If we increase the number of sets to n, and still find that the states that interest us are losing, then we conclude that the game is indeed lost. The result is a series of games of increasing complexity. The first automaton has $2^n \cdot 2^{n+2}$ states and index four, resulting in complexity $O(t \cdot (2^n \cdot n^{n+2})^2 \cdot (g \cdot n^2 \cdot n^{n+2})^2)$, where g and t are the number of locations and transitions in G, respectively. In general, the ith automaton has $2^n \cdot i^{n+i}$ states and index $2i$, resulting in complexity $O(t \cdot (2^n \cdot i^{n+i})^2 \cdot (g \cdot 2^n \cdot i^{n+i})^i)$.

We give a family of games and automata that require almost the full power of our construction. Furthermore, we identify several sets of edges in each game

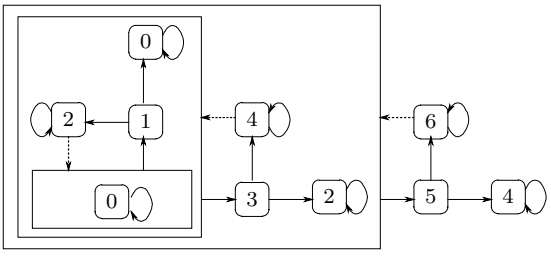

Fig. 1. The game G_3

such that removing one set of edges allows to remove one set from the GFG automaton and identify the winning regions correctly.

We give a recursive definition of the game G_i. Let G_0 be $\langle V^0, \emptyset, V^0, \rho^0, \mathcal{N}^0 \rangle$, where $V^0 = S^0 = \{s_0^0\}$ and $\rho^0 = \{(s_0^0, s_0^0)\}$. The acceptance condition is given with respect to a labeling of the states of the game, to be defined below. The game G_i is $\langle V^i, \emptyset, V^i, \rho^i, \mathcal{N}^i \rangle$, where $V^i = V^{i-1} \cup S^i$, and $S^i = \{s_1^i, s_2^i, s_3^i\}$, and $\rho^i = \rho^{i-1} \cup T^i \cup R^i$, and $T^i = \{(s_1^i, s_2^i), (s_1^i, s_3^i), (s_2^i, s_2^i), (s_3^i, s_3^i)\} \cup ((\bigcup_{j<i} S^j) \times \{s_1^i\})$, and $R^i = \{s_3^i\} \times (\bigcup_{j<i} S^j)$. The labeling on the states of the game is defined as follows. We set $L(s_0^0) = 0$ and for all $i \geq 1$, we set $L(s_1^i) = 2i-1$, $L(s_2^i) = 2i-2$, and $L(s_3^i) = 2i$. The graph depicted in Figure 1 is G_3. An edge from a rectangle to a state s is a shorthand for edges from all states in the rectangle to s, and similarly for edges from states to rectangles. Note that G_{i-1} is contained in G_i.

The winning condition is $\mathcal{N}^i = \langle [2i+2], [2i+2], \eta, 2i+2, [2i+2]^{\text{even}} \rangle$ where $[n]$ is $\{1, \ldots, n\}$, and $[n]^{\text{even}} = \{i \in [n] \mid i \text{ is even}\}$, and η is as follows:

$$\eta(2k, j) = \begin{cases} \emptyset & j{>}2k \\ \{2k\} & j{=}2k \\ \{j, j{+}1, j{+}3, \ldots, 2k{-}1\} & j{<}2k \text{ even} \\ \{j, j{+}2, \ldots, 2k-1\} & j{<}2k \text{ odd} \end{cases}$$

$$\eta(2k+1, j) = \begin{cases} \emptyset & j{>}2k{+}2 \\ \{2k{+}2\} & j{=}2k{+}2 \\ \{j, j{+}1, j{+}3, \ldots, 2k{+}1\} & j{<}2k{+}2 \text{ is even} \\ \{j, j{+}2, \ldots, 2k{+}1\} & j{<}2k{+}2 \text{ is odd} \end{cases}$$

It is also the case that $\mathcal{N}_{i-1}$ is contained in $\mathcal{N}_i$.

We show that for all $i \geq 0$, player 0 wins from every state in G_i. Furthermore, in order to use our GFG construction from Section 5, we have to use $i{+}1$ sets. That is, if we take the product of the graph G_i with the GFG that uses $i{+}1$ sets (denoted $\mathcal{P}_i$), then player 0 wins form every state in the resulting parity game. We further show that this does not hold for the GFG with i sets. That is, player 1 wins from some of the states in the product of G_i and $\mathcal{P}_{i-1}$. Finally, the edges in G_i are $\rho_0 \cup \bigcup_{j \leq i} T^j \cup R^j$. Consider a set of edges R^j for $j < i$. We show that if we remove R^j from G_i, then we can remove one set from the GFG. If we now

remove R^k for $k < i$, then we can remove another set from the GFG, and so on. This is summarized in the following lemmata.

Lemma 3. *For all $i \geq 0$, the following are true:*
- *player 0 wins from every location in G_i,*
- *player 0 wins the parity game $G_i \otimes \mathcal{P}_i$, and*
- *if $i \geq 1$, then player 1 wins from $(s_0^0, q_0, 0)$ in $G_i \otimes \mathcal{P}_{i-1}$.*

Consider some set $I \subseteq [i]$ such that $i \in I$. Let G_i^I denote the game with locations S^i and transitions $(\bigcup_{i' \leq i} T^i) \cup (\bigcup_{i' \in I} R^{i'})$. That is, G_i^I includes only the transitions in $R^{i'}$ for $i' \in I$.

Lemma 4. *For all $I \subseteq [i]$ such that $i \in I$ and $|I| = j$ the following are true:*

- *player 0 wins the parity game $G_i^I \otimes \mathcal{P}_j$, and*
- *player 1 wins from $(s_0^0, q_0, 0)$ in $G_i^I \times \mathcal{P}_{j-1}$.*

7 Conclusion and Future Work

We introduced a definition of nondeterministic automata that can be used for game monitoring. Our main contribution is a construction that takes an NBW and constructs a GFG NPW with $2^n \cdot n^{2n}$ states. In comparison, the DPW constructed by Piterman has n^{2n+2} states. However, the structure of the NPW is much simpler, and we suggest that it be implemented symbolically.

We also suggest an incremental approach to solving games. The algorithm of Kupferman and Vardi also shares this property [KV05] (though it cannot be used directly for games with NBW winning conditions). In addition, their algorithm allows to reuse the work done in the earlier stages of the incremental search. We believe that the symmetric structure of our automata will allow similar savings. Another interesting problem is to find a property of game graphs that determines the number of sets required in the GFG construction.

Starting from a Rabin or a parity automaton, it is easy to construct an equivalent Büchi automaton. This suggests that we can apply our construction to Rabin and parity automata as well. Recently, it has been shown that tailored determinization constructions for these types of automata can lead to great savings in the number of states. A similar question is open for GFG automata, as well as for Streett automata.

Finally, we mention that our GFG automaton cannot be used for applications like emptiness of alternating tree automata. The reason is that emptiness of alternating tree automata requires co-determinization, i.e., producing a deterministic automaton for the complement of the original language. We are searching for ways to construct a GFG automaton for the complement language.

Acknowledgment

We thank M.Y. Vardi for pointing out the disadvantages of the construction in [KV05].

References

[ATW05] C.S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. In *CIAA*, LNCS. Springer, 2005.

[BL69] J.R. Büchi and L.H.G. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

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

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model checking for fragments of μ-calculus. In *CAV*, LNCS 697, pp. 385–396, Springer, 1993.

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

[HRS05] A. Harding, M. Ryan, and P.Y. Schobbens. A new algorithm for strategy synthesis in LTL games. In *TACAS*, LNCS 3440, pp. 477–492. 2005.

[JGB05] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *CAV*, LNCS 3576, pp. 226–238, Springer, 2005.

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

[JV00] M. Jurdzinski J. Voge. A discrete strategy improvement algorithm for solving parity games. In *CAV*, LNCS 1855, pp. 202–215, Springer, 2000.

[KB05] J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. In *CIAA*, LNCS. Springer, 2005.

[KPP05] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. *I&C*, 200:35–61, 2005.

[KV05] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *FOCS*, IEEE, 2005.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *MST*, 3:376–384, 1969.

[Löd98] C. Löding. Methods for the transformation of ω-automata: Complexity and connection to second-order logic. Master Thesis, University of Kiel, 1998.

[Mar75] D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.

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

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, IEEE, 2006. To appear.

[RA03] P. Madhusudan R. Alur, and S. La Torre. Playing games with boxes and diamonds. In *CONCUR*, LNCS 2761, pp. 127–141, Springer, 2003.

[Rab72] M.O. Rabin. Automata on infinite objects and Church's problem. *American Mathematical Society*, 1972.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Trans. Control Theory*, 77:81–98, 1989.

[Saf88] S. Safra. On the complexity of ω-automata. In *FOCS*, IEEE, 1988.

[THB95] S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic ω-automata. In *CHARME*, LNCS 987, 1995.