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 N that is* good for solving games *with objective N. 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 their 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 [27], and the second, by reducing it to solving infinite-duration two-player games [3]. 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 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 states 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 states. In either case, we are interested in *solving* the game. That is, we wish to determine from which states 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 Rabin condition [26], the problem of solving the game is NP-complete [4] and the current best complexity for solving such games is $O(t \cdot g^{2k} \cdot k!)$, where t is the number of transitions in the game, g is the number of states, and k is the number of pairs in the Rabin condition [16, 10]. Parity games [5] are known to be in NP $\cap$ co-NP [6] and the best complexity for solving them is $O(t \cdot g^{\lfloor \frac{k}{2} \rfloor})$, where k is the number of priorities in the parity condition [13].

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 [18]. We have to use stronger acceptance conditions like parity or Rabin. Indeed, Safra suggests a determinization construction that takes a NBW and constructs a deterministic Rabin automaton [29]. Specifically, starting from an NBW with n states, he constructs a deterministic Rabin automaton with $12^n \cdot n^{2n}$ states and n Rabin pairs [29]. When we combine the game with the deterministic automaton, we get a game with $g \cdot 12^n \cdot n^{2n}$ states and $t \cdot 12^n \cdot n^{2n}$ transitions, where g and t are the number of states and transitions in the original game. The overall complexity of solving this game, therefore, is $O(t \cdot 12^n \cdot n^{2n} (g \cdot 12^n \cdot n^{2n})^{2n} \cdot n!)$.[1] This theory is not applicable in practice, because Safra's determinization is extremely complex. Every state of the deterministic automaton is a tree of colored subsets of states of the original automaton. A transition moves states between different nodes of the tree, adds and removes nodes, and colors them. Only recently, 16 years after the publications of Safra's construction, it was finally implemented [14, 1]. These implementations are limited to determinize automata with approximately 10 states. One possible solution is to consider restricted specifications that can be handled more efficiently (cf. [28, 2, 25, 15]). Another possible solution is to use nondeterministic specification automata [11, 12], which make the approach sound but incomplete.

Here we do pursue complete solutions for most general 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 [20]. Using a symbolic state representation in Safra's construction seems impossible. 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 exist. The memory used by a strategy corresponds to the number of states of a parity or Rabin specification automaton. Thus, considering 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 step-wise 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 [9] a deterministic automaton for the same language. We further show how, given an NBW with n states, we can construct a GFG automaton for the same language. Our construction yields a nondeterministic parity automaton with $2^n \cdot n^{2n}$ states and index $2n$, giving an overall complexity of $O(t \cdot 2^n \cdot n^{2n} (g \cdot 2^n \cdot n^{2n})^n)$ for synthesis. We also generalize the $n!$ lower bound on the size of the deterministic automaton 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 an amazingly 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

---

[1] An improvement of Safra's determinization yields a deterministic parity automaton with n^{2n+2} states and $2n$ priorities [24]. The overall complexity then reduces to $O(t \cdot n^{2n+2} (g \cdot n^{2n+2})^n)$.

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 graphs and winning conditions for which this incremental approach indeed leads to considerable savings.

Recently, Kupferman and Vardi suggested another construction that avoids determinization in certain situations [17]. Their algorithm shows how to solve the emptiness problem of alternating parity tree automata through a reduction to the emptiness problem of non-deterministic 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 and the size of the produced strategy may be quadratically worse than the algorithm presented here.

2 Preliminaries

2.1 Nondeterministic Automata

A *nondeterministic automaton* is $N = \langle \Sigma, S, \delta, s_0, \alpha \rangle$, 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 N on a word $w = w_0 w_1 \cdots$ is an infinite sequence of states $t_0 t_1 \ldots \in S^\omega$ such that $t_0 = s_0$ and forall $i \geq 0$ we have $t_{i+1} \in \delta(t_i, w_i)$. For a run $r = s_0 s_1 \ldots$, let $inf(r) = \{s \in S \mid s = s_i$ for infinitely many i's$\}$ be the set of all states occurring infinitely often in the run. We consider three acceptance conditions. A *Rabin* condition α is a set of pairs $\{\langle L_1, U_1 \rangle, \ldots, \langle L_k, U_k \rangle\}$ where forall i we have $L_i, U_i \subseteq S$. A run is *accepting* according to the Rabin condition α if for some i we have $inf(r) \cap L_i \neq \emptyset$ and $inf(r) \cap U_i = \emptyset$. That is, for some pair $\langle L_i, U_i \rangle$ the run visits L_i infinitely often and U_i finitely often. A *parity* condition α is a partition $\{F_0, \ldots, F_k\}$ of S. We call k the *index* of the parity condition. A run is *accepting* according to the parity condition α if for some even

i we have $inf(r) \cap F_i \neq \emptyset$ and forall $i' < i$ we have $inf(r) \cap F_{i'} = \emptyset$. That is, the minimal set to be visited infinitely often is even. A *Büchi* condition is $F \subseteq S$. A run 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 N if there exists some accepting run of N over w. The *language* of N is the set of words accepted by N. Formally, $L(N) = \{w \mid w$ is accepted by $N\}$. Two automata are *equivalent* if they accept the same language.

Given a set of states $S' \subseteq S$ and a letter $\sigma \in \Sigma$, we denote by $\delta(S', \sigma)$ the set $\bigcup_{s \in S'} \delta(s, \sigma)$.

An automaton 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 denote automata by acronyms in $\{N, D\} \times \{R, P, B\} \times \{T, W\}$. The first symbol stands for the branching mode of the automaton: N for nondeterministic and D for deterministic. The second symbol stands for the acceptance condition of the automaton: R for Rabin, P for parity, and B for Büchi. The last symbol stands for the object the automaton is reading: T for trees and W for words. For example, a DPW is a deterministic parity word automaton and an NBT is a nondeterministic Büchi tree automaton.

2.2 Games

A *game* is a tuple $G = \langle V, V_0, V_1, \rho, W \rangle$ where V is the set of *locations* or *states* of the game, V_0 and V_1 are a partition of V to locations of player 0 and player 1 respectively, $\rho \subseteq V \times V$ is the *transition relation* or *edges*, and $W \subseteq V^\omega$ is the *winning set* of G.

A *play* in G is a maximal sequence of locations $\pi = v_0 v_1 \cdots$ such that forall $i \geq 0$ we have $(v_i, v_{i+1}) \in \rho$. A play π is *winning* for player 0 if $\pi \in W$ or π is finite $\pi = \pi' v$ and $v \in V_1$ (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 whenever $f(\pi v)$ is defined $(v, f(\pi v)) \in \rho$. We say that a play $\pi = v_0 v_1 \cdots$ is *f-conform* if whenever $v_i \in V_0$ we have $v_{i+1} = f(v_0 \cdots v_i)$. The strategy f is *winning from v* if every f-conform play that starts in v is winning for player 0. We say that *player 0 wins* from v if she has a winning strategy. The *winning region* of player 0, is the set of states from which player 0 wins. We denote the winning region of player 0 by W_0. A strategy, winning strategy, win, and winning region are defined dually for player 1. We *solve* a game by computing the winning regions W_0 and W_1. For the kind of games handled by this paper W_0 and W_1 form a partition of V [8].

Also here, we consider parity and Rabin winning

conditions. These are defined just like for automata over the locations of the game. The following Theorem summarizes the complexity of solving games.

Theorem 2.1 [13, 16] *Given a game G with g states and t transitions we can solve G in time*

- $O(t \cdot g^{\lfloor \frac{k}{2} \rfloor})$ *where G is a parity game of index k.*
- $O(t \cdot g^{2k} \cdot k!)$ *where G is a Rabin game with k pairs.[2]*

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 N over the alphabet V such that $W = L(N)$. We abuse notations and write $G = \langle V, V_0, V_1, \rho, N \rangle$ or just $G = \langle V, \rho, N \rangle$. The common approach to solving such games is by reducing them to either Rabin or parity games. Consider a game $G = \langle V, V_0, V_1, \rho, W \rangle$ and a deterministic automaton $\mathcal{M} = \langle V, M, \eta, v_0, \alpha \rangle$ whose alphabet is V such that $L(\mathcal{M}) = W$. We define the *product* of G and $\mathcal{M}$ to be the game $G \times \mathcal{M} = \langle V \times M, V_0 \times M, V_1 \times M, \rho', W' \rangle$ where $((v, s), (v', s')) \in \rho'$ iff $(v, v') \in \rho$ and $s' = \eta(s, v)$ and W' contains all the plays whose projection on the second component is winning according to α. A *monitor for G* is a deterministic automaton $\mathcal{M}$ such that v is winning for player 0 in G iff (v, m_0) is winning for player 0 in $G \times \mathcal{M}$. The common way to solve a game $G = \langle V, \rho, \mathcal{N} \rangle$ where $\mathcal{N}$ is an NBW is constructing an equivalent DRW $\mathcal{D}$ [29] and considering the product $G \times \mathcal{D}$. Unfortunately, determinization has defied implementation until recently and it cannot be implemented symbolically [31, 1, 14]. 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 2.2 *Consider a game $G = \langle V, \rho, N \rangle$ where N is an NBW with n states. We can construct a DRW D equivalent to N with $12^n \cdot n^{2n}$ states and n pairs. The Rabin game that is the product of G and D can be solved in time $O(t \cdot 12^n \cdot n^{2n} (g \cdot 12^n \cdot n^{2n})^{2n} n!)$ where g is the number of states in V and t is the size of ρ.[3]*

It is a 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 term such automata *good for games* (GFG). We show one possible way of deciding when an automaton is GFG and give a construction that takes an NBW

and produces a GFG NPW. Our NPW is much simpler than the DRW constructed by [29], has fewer states, it is amenable to symbolic implementation, and allows 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. We term such automata as *good for games* (GFG for short). We show that our definition is strong enough, namely, we can indeed use such automata for game monitoring.

In order to define GFG automata we consider the following game. Let $\mathcal{M} = \langle \Sigma, M, \eta, m_0, \alpha \rangle$ be an automaton. The *monitor game* is a game played on the set of states M. The game proceeds in rounds in which player 1 chooses a letter and player 0 answers with a successor state of previous location reading that letter. Formally, a *play* is a maximal sequence $\pi = m_0 \sigma_0 m_1 \sigma_1 \cdots$ such that forall $i \geq 0$ we have $m_{i+1} \in \eta(m_i, \sigma_i)$. That is, a play produces an infinite word $w(\pi) = \sigma_0 \sigma_1 \cdots$ and a run $r(\pi) = m_0 m_1 \cdots$ of $\mathcal{M}$ on $w(\pi)$. A play π is winning for player 0 if π is infinite and in addition either $w(\pi)$ is not in $L(\mathcal{M})$ or $r(\pi)$ is an accepting run 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 $\mathcal{M}$ is accepting.

A strategy for player 0 is a partial function $f : (M \cdot \Sigma)^+ \rightarrow M$ such that whenever $f(\pi m \sigma)$ is defined we have $f(\pi m \sigma) \in \eta(m, \sigma)$. We say that a play $\pi = m_0 \sigma_0 m_1 \sigma_1 \cdots$ is *f-conform* if forall $i \geq 0$ we have $m_{i+1} = f(m_0 \cdots \sigma_i)$. The strategy f is *winning from m* if every f-conform play that starts in m is winning for player 0. We say that *player 0 wins* from m if she has a winning strategy from m. We say that $\mathcal{M}$ is GFG automaton if player 0 wins from m_0.

We show how to use GFG automata for game monitoring. Let $\mathcal{M} = \langle V, M, \eta, m_0, \alpha \rangle$ be a GFG automaton and consider a game $G = \langle V, V_0, V_1, E, \mathcal{M} \rangle$. We construct the following game. Let $G \times \mathcal{M} = \langle V', V_0', V_1', E', W' \rangle$ where $V' = V \times M \times \{0, 1\}$, $V_0' = (V \times M \times \{0\}) \cup (V_0 \times M \times \{1\})$, $V_1' = (V_1 \times M \times \{1\})$, $E' = \{((v, m, 0), (v, m', 1)) \mid m' \in \eta(m, v)\} \cup \{((v, m, 1), (v', m, 0)) \mid (v, v') \in E\}$, and $W' = \{\pi \in V'^\omega \mid \pi \Downarrow_M$ satisfies $\alpha\}$. Wlog, we assume that the acceptance condition of $\mathcal{M}$ is closed under finite stuttering (which is true for Büchi, parity, and Rabin).

When $\mathcal{M}$ is a GFG we can use $G \times \mathcal{M}$ to solve G.

Theorem 3.1 *Player 0 wins from location v in G iff she wins from location $(v, m_0, 0)$ in $G \times \mathcal{M}$.*

---

[2]We note that by reducing Rabin games to parity games we can solve Rabin games in time $O(t(g \cdot k!)^k)$.

[3]Using a recent improvement of Safra's determinization, we can construct a DPW with n^{2n+2} states and index $2n$ and the overall complexity reduces to $O(t \cdot n^{2n+2}(g \cdot n^{2n+2})^n)$ [24].

A win in $G \times \mathcal{M}$ is easily translated to a win over G by forgetting the $\mathcal{M}$ component. In the other direction, a winning strategy in G is combined with a winning strategy in the monitor game over $\mathcal{M}$ to produce a combined strategy in $G \times \mathcal{M}$. As the strategy used in G is winning, the resulting play is accepted by $\mathcal{M}$. As the strategy in the monitor game is winning it follows that the projection of the play on the states of $\mathcal{M}$ is an accepting run. The full proof is given in Appendix A.

We established that GFG automata are useful for solving games. We show how to check whether an automaton is GFG and how to construct GFG automata.

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 for the same language. By definition every deterministic automaton is GFG. This follows from the fact that player 0 does not have a choice in the monitor component. Hence, if an automaton fairly simulates the deterministic automaton for the same language, it is GFG. We define fair simulation [9] and show that fair simulation establishes the GFG property.

4.1 Fair Simulation

We define *fair simulation* [9]. Consider two automata $N = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ and $R = \langle \Sigma, T, \rho, t_0, \beta \rangle$. In order to define fair simulation we define the *fair-simulation game*. Given N and R, let $G_{N,R} = \langle V, V_0, V_1, \rho, W \rangle$ be the game with the following components.

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

Given an infinite play $\pi = v_0 v_1 \cdots$ we define π_1 to be the projection of π on the states in S and π_2 to be the projection of π on the states in T. Player 0 wins a play π if π is infinite and whenever π_1 is an accepting run of N then π_2 is an accepting run of R (wlog, the acceptance condition is closed under finite stuttering). If player 0 wins the fair-simulation game from state (s,t) then t fairly simulates s, denoted by $s \leq_f t$. If both $s \leq_f t$ and $t \leq_f s$ then s and t are *fair-simulation equivalent*, denoted $s =_f t$. If $s_0 \leq_f t_0$ we say that R *fairly simulates* N, denoted $N \leq_f R$.

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 4.1 *Let $\mathcal{N}$ be a nondeterministic automaton and $\mathcal{D}$ a GFG automaton such that $L(\mathcal{N}) = L(\mathcal{D})$. Then $\mathcal{D} \leq_f \mathcal{N}$ implies $\mathcal{N}$ is GFG.*

Proof: Let $\mathcal{N} = \langle \Sigma, N, \rho, n_0, \alpha \rangle$ and $\mathcal{D} = \langle \Sigma, D, \eta, d_0, \beta \rangle$. Let $G_{D,N} = \langle V, V_0, V_1, \rho, W \rangle$ be the fair-simulation game between $\mathcal{D}$ and $\mathcal{N}$ and suppose $\mathcal{D} \leq_f \mathcal{N}$. Let $f : V^* \times V_0 \to V$ be a winning strategy for player 0 in $G_{D,N}$. We denote the monitor game over $\mathcal{D}$ by G_1 and the monitor game over $\mathcal{N}$ 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. Then h instructs us how to simulate player 1 in $G_{D,N}$ and the choice of f in $G_{D,N}$ 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 $G_{D,N}$ and G_1 are f-conform and h-conform respectively.
- The projection of the play in G_2 on Σ is the projection on Σ of the plays in G_1 and $G_{D,N}$.
- The projection of the play in G_1 on the states of $\mathcal{D}$ is the projection of the play in $G_{D,N}$ on the states of $\mathcal{D}$.
- The projection of the play in $G_{D,N}$ on the states of $\mathcal{N}$ is the projection of the play in G_2 on the states of $\mathcal{N}$.

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

Let $\pi_2 = n_0 \sigma_0 \, n_1 \sigma_1 \cdots n_i$ be a play in G_2, let $\pi_1 = d_0 \sigma_0 \, m_1 \sigma_1 \cdots m_i$ be a play in G_1, and let $\pi_s = (d_0, n_0) (d_1, n_0, \sigma_0) (d_1, n_1) \cdots (d_i, n_i)$ be a play in $G_{D,N}$. 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 $G_{D,N}$ it follows that $n_{i+1} \in \rho(n_i, \sigma_i)$. 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 on 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 on D is an accepting run of $\mathcal{D}$. As f is a winning strategy in $G_{D,N}$, we conclude that the projection of π_s on N is also accepting. As the projections of π_2 and π_s on N are equivalent we are done. $\square$

The above condition is not only sufficient it is 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 are fair-simulation equivalent.

5 Constructing GFG Automata

In this section we describe 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 DRW for the same language [29].

5.1 From NBW to NPW

The idea behind the construction of the NPW is to mimic the determinization construction [29]. Safra constructs a tree of subset constructions. 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 [21]: 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 very simple GFG NPW.

Let $\mathcal{N} = \langle \Sigma, S, \rho, s_0, \alpha \rangle$ 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 of states Q 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 s is not in the set, $C(s) = 1$ means that s is in the set and is unmarked, and $C(s) = 2$ means s is in the set and is marked. For simplicity of notation we represent such an annotated set C by a pair of sets $(A, B) \in 2^S \times 2^S$ where $B \subseteq A$ 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 notations and write $(A, B) \in \{0, 1, 2\}^S$. We write $(A, B) \subseteq (C, D)$ to denote $A \subseteq C$ and $B \subseteq D$.

  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. A formal definition of Q is given in Figure 1.

- In order to define the transition η we need a few definitions.

  For a set $(A, B) \in \{0, 1, 2\}^S$, a letter $\sigma \in \Sigma$, and $i \in \{0, 1\}$ let $succ((A, B), \sigma, i)$ denote the set defined in Figure 1.

  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 then we unmark them [4]. In the case that $i = 1$ we are completely free in the choice of B'.

  Given sets $(A, B), (C, D) \in \{0, 1, 2\}^S$ and letter $\sigma \in \Sigma$, let $trans((A, B), \sigma, (C, D))$ be as follows.

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

  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 $succ((A, B), \sigma)$ 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 η is defined for every state $q \in Q$ and letter $\sigma \in \Sigma$ as follows. Let $q = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$.

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

  Intuitively, (A_1, B_1) hold 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 sets gets 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 to load a new set.

  Notice, that emptying a set forces the automaton to empty all the sets above it and load them again from (A_1, B_1).

- $q_0 = \langle (\{s_0\}, \{s_0\} \cap \alpha), (\emptyset, \emptyset), \ldots, (\emptyset, \emptyset) \rangle$

  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.

---

[4] 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. In particular, it simplifies the proofs below.

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

$$succ((A, B), \sigma, i) = \left\{ \begin{array}{ll} \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \rho(A, \sigma) \text{ and} \\ B' \subseteq (\rho(B, \sigma) \cap A') \cup (A' \cap \alpha) \end{array} \right\} & B \neq A \text{ and } i = 0 \\[2em] \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \rho(A, \sigma) \text{ and} \\ B' \subseteq A' \cap \alpha \end{array} \right\} & B = A \text{ and } i = 0 \\[2em] \left\{ (A', B') \;\middle|\; \begin{array}{l} A' \subseteq \rho(A, \sigma) \text{ and} \\ B' \subseteq A' \end{array} \right\} & i = 1 \end{array} \right.$$

Figure 1. The set of states Q and the function $succ$.

- 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
  - $F_0 = \{q \mid A_1 = B_1 \text{ and } A_1 \neq \emptyset\}$.
  - $F_{2i+1} = \{q \mid ind_E(q) = i + 2 \text{ and } ind_F(q) \geq i + 2\}$.
  - $F_{2i+2} = \{q \mid ind_F(q) = i + 2 \text{ and } ind_E(q) > i + 2\}$.
  As all sets greater than $ind_E(q)$ are empty, the odd sets require that forall 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.

We first show that $\mathcal{N}$ and $\mathcal{P}$ are equivalent. We show that $\mathcal{P}$ contains $\mathcal{N}$ by using the run of $\mathcal{N}$. We use the first set in a state of $\mathcal{P}$ to follow singletons from the run of $\mathcal{N}$. The proof that $\mathcal{P}$ is contained in $\mathcal{N}$ is similar to the proof that the DRW constructed by Safra is contained in the language of $\mathcal{N}$ [29]. The full proof is given in Appendix B.

Lemma 5.1 $L(\mathcal{P}) = L(\mathcal{N})$.

Let $\mathcal{D}$ be the DRW constructed by Safra [29]. We show that $\mathcal{P}$ fairly simulates $\mathcal{D}$. In fact $\mathcal{D}$ also fairly simulates $\mathcal{P}$. This follows immediately from the two having the same language and $\mathcal{D}$ being deterministic [9]. Thus, $\mathcal{D}$ and $\mathcal{P}$ are fair-simulation equivalent.

Lemma 5.2 $\mathcal{D} \leq_f \mathcal{P}$.

The proof proceeds by showing how to choose a state of $\mathcal{P}$ that maintains the same sets as the tree state of the deterministic automaton. Part of the problem is in linearizing the nodes in the tree. This is done by

maintaining a permutation π that minimizes the nodes that are oldest in the tree (over all nodes). We use this permutation π to associate the ith set in the state of $\mathcal{P}$ with node $\pi(i)$ in the tree state of $\mathcal{D}$.

5.2 Complexity Analysis

We analyze the complexity of the automaton presented above. We count the number of states of the automaton and analyze the complexity of using it for solving games.

Theorem 5.3 *Given an NBW $\mathcal{N}$ with n states, we can construct a GFG NPW $\mathcal{P}$ with $2^n n^{2n}$ states and index $2n$.*

Proof: We represent a state of $\mathcal{P}$ as a tree of subsets. The pair (A_i, B_i) is a son of the pair $(A_{i'}, B_{i'})$ such that $A_i \subseteq B_{i'}$. This tree structure is represented by a function $p : [n] \to [n]$.

We map every state of $\mathcal{N}$ to the minimal node in the tree (according to the parenthood relation) to which it belongs. Thus, the partition to $A_1, \ldots, A_n$ is represented by a function $l : S \to [n]$.

Every state of $\mathcal{N}$ that appears in a pair (A_i, B_i) and also in some son $(A_{i'}, B_{i'})$ belongs to B_i. In addition we have to remember all the states s of $\mathcal{N}$ that appear in some set A_i, in no descendant of A_i and in addition 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. $\square$

The above stated bound improves on the size of the DRW constructed by Safra [29] by a factor of 6^n. In addition, the NPW is much simpler than the DRW. As mentioned, Safra's construction proved very hard

to implement. Existing constructions [1, 14] enumerate the states. The structure of the NPW above is much simpler and amenable to symbolic methods.[5] 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 DRW. This would result in augmenting and 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 best 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 $\mathcal{N} = \langle V, S, \rho, s_0, \alpha \rangle$. Let $|S| = n$, and let g and t be the number of states and transitions of G respectively. Using Safra's construction, we construct a DRW $\mathcal{R}$ with $12^n \cdot n^{2n}$ states and n pairs. According to Theorem 2.1, we can solve the resulting Rabin game in time $O(t \cdot 12^n \cdot n^{2n}(g \cdot 12^n \cdot n^{2n})^{2n} \cdot n!)$. If we use our GFG NPW, we replace $12^n \cdot n^{2n}$ above by $2^n \cdot n^{2n}$. In addition, the exponent reduces from $2n$ to n and the $n!$ multiplier disappears. That is, we can solve the resulting parity game in time $O(t \cdot n^{2n+2}(g \cdot n^{2n+2})^n)$. Notice, that in this case the construction of Kupferman and Vardi cannot be applied directly [17]. In order to apply their construction, Kupferman and Vardi need an NBW for the complement of the winning condition.

In the context of LTL games (i.e., games with LTL winning conditions) Kupferman and Vardi's construction can be applied. Their construction can be applied in time $O(t \cdot 12^n \cdot n^{2n} \cdot n!(g \cdot 12^n \cdot n^{2n} \cdot n!)^{2n})$. We note that even if we use the symbolic algorithm for solving parity games [7] our upper bound increases to $O(t \cdot n^{2n+2}(g \cdot n^{2n+2})^{2n})$, which is still significantly better than previous approaches.

We note that in the context of emptiness of alternating parity tree automata our GFG construction cannot be applied. This is similar to the reason why Kupferman and Vardi's method cannot be used for games with NBW winning condition. In this case, we have to construct a GFG NPW for the complement language, which we do not know how to do.

5.3 Lower Bound

We use the lower bound on memory needed for winning strategies to show that our construction is in some sense optimal. We generalize Michel's lower bound on the size of determinization [22, 19]. That is, we construct a game with an NBW acceptance condition that requires $n!$ memory. Given that our GFG automaton can be used as the memory, this proves that every GFG automaton for the given language has at least $n!$ states.

We start by defining the winning condition. The winning condition is defined by the NBW $N_n = \langle \Sigma_n, S_n, \rho_n, S_0^n, \{0\} \rangle$ where $\Sigma_n = \{1, \ldots, n, \#\}$, $S_n = \{0, \ldots, n\}$, $S_0^n = \{1, \ldots, n\}$, and the transition ρ is defined for every state $i \in \{0, \ldots, n\}$ and letter $\sigma \in \Sigma_n$ as follows.

$$\rho(i, \sigma) = \begin{cases} \{\sigma\} & i = 0 \text{ and } \sigma \neq \# \\ \{0, i\} & i \neq 0 \text{ and } i = \sigma \\ \{i\} & \text{Otherwise} \end{cases}$$

The following lemma characterizes the language of N.

Lemma 5.4 [22, 19] *The following statements are equivalent. (a) $w \in L(N_n)$. (b) There exist letters $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that the sequences $i_1 i_2, \ldots, i_{k-1} i_k, i_k i_1$ appear infinitely often in w.*

It follows that the only words not in $L(N_n)$ are the words w for which there exists some permutation $i_1 \cdots i_n$ over $1..n$ such that eventually w contains only subsequences of $i_1 \cdots i_n$ enclosed by $\#$.

We now define the game G_n. Intuitively, the game allows player 1 to choose a permutation π and then player 0 chooses a permutation π'. Player 1 then chooses two values in $[n]$ and reverses the order in which they appear in π'. It follows that if player 0 chooses the same permutation as player 1 then player 1 has no choice but to 'close a cycle'. If player 0 chooses a different permutation, then player 1 can choose a pair whose order in π' is in accordance with π. Formally, we have the following. Let Π denote the set of permutations over $[n]$. The game G_n is $\langle V, V_0, V_1, E, N_n \rangle$ with the following components.

- $V = \{0, 1\} \cup [n]^2 \cup \bigcup_{\pi \in \Pi} V_\pi$ where

  $V_\pi = \{\pi, \pi_{1,2}, \ldots, \pi_{1,n}, \pi_{2,1}, \ldots, \pi_{2,n-1}\}$.
  The states 0, 1, and $\pi \in \Pi$ are labeled by $\#$. A state $(i, j) \in [n]^2$ is labeled by j, and where $\pi = i_1 \cdots i_n$ we have $\pi_{j,l}$ is labeled by i_l.
- $V_0 = \{1\}$ and $V_1 = V - V_0$.
- The transition is $E = E_0 \cup E_1 \cup E_2 \cup \bigcup_{\pi \in \Pi} E_\pi$ where

  - $E_0 = \{(0, (1, j)), ((n, j), 1) \mid j \in [n]\}$.
  - $E_1 = \{((i, j), (i+1, j')) \mid i, j, j' \in [n] \text{ and } i < n\}$.

$- E_2 = \{(1, \pi), (\pi, \pi_{1,j}), (\pi_{2,j}, 0) \mid \pi \in \Pi \text{ and } j \in [n]\}.$

$- E_\pi = \{(\pi_{1,j}, \pi_{2,l}) \mid j > l\}.$

A play proceeds by rounds that start in state 0. The round starts by player 1 choosing a sequence of n labels (or rather a permutation). Then player 0 chooses a permutation $\pi \in \Pi$ which is followed by player 1 choosing a pair ordered according to the inverse of this permutation. Finally, the play returns to state 0. We show that player 0 wins this game however she cannot do that with less than $n!$ memory.

It follows that every GFG P_n such that $L(P_n) = L(N_n)$ has at least $n!$ states. A full proof is given in Appendix C. Formally, we have the following.

Theorem 5.5 *There exists a family of NBW N_n such that N_n has n states and the minimal GFG automaton equivalent to N_n 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 $G = \langle V, E, N \rangle$, where N is an NBW with n states. We can apply the construction from Section 5 but use only 2 sets (i.e., restrict the sets $3, \ldots, n$ to the empty set). We then combine the restricted automaton with G and try to solve the resulting game. If we find that the states that interest us in the game are winning we stop. Otherwise, we try a less restricted automaton with 3 sets, then 4 sets, etc. 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 four priorities, resulting in complexity $O(t \cdot 2^n \cdot n^{n+2}(g \cdot n^2 \cdot n^{n+2})^2)$, where g and t are the number of states and transitions in G respectively. In general, the $i{-}1$th automaton has $2^n \cdot i^{n+i}$ states and $2i$ priorities, resulting in complexity $O(t \cdot 2^n \cdot i^{n+i}(g \cdot 2^n \cdot i^{n+i})^i)$. In this section we show that this incremental approach is indeed useful. 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 such that removing one set of edges allows us to remove one set from the GFG automaton and still identify the winning regions correctly.

We give a recursive definition of the game G_i. The game G_0 is $\langle V^0, \emptyset, V^0, \rho^0, \mathcal{N}^0 \rangle$ 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 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 $V^i = V^{i-1} \cup S^i$, $S^i = \{s_1^i, s_2^i, s_3^i\}$, $\rho^i = \rho^{i-1} \cup T^i \cup R^i$, and $T^i = \{(s_1^i, s_2^i), (s_1^i, s_3^i), (s_2^i, s_2^i), s_3^i, s_3^i)\} \cup$
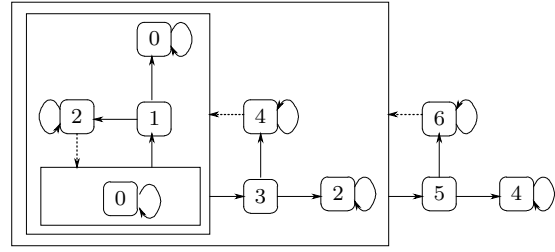
Figure 2. The game G_3.

$((\bigcup_{i' < i} S_{i'}) \times \{s_1^i\})$ and $R^i = \{s_3^i\} \times (\bigcup_{i' < i} S_{i'})$. The labeling on the states of the game is defined as follows. We set $L(s_0^0) = 0$ and forall $i \geq 1$ 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 2 is G_3. Notice that G_{i-1} is contained in G_i.

The winning condition is the automaton $\mathcal{N}^i = \langle [2i + 2], [2i + 2], \eta, 2i + 2, [2i + 2]^{\text{even}} \rangle$ where $[2i]^{\text{even}}$ is the set of even values in $[2i]$ and η is as follows.

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

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

We show that forall i we have 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 ($\mathcal{P}_i$), 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 $\bigcup_{i' \leq i} T^{i'} \cup R^{i'}$. Consider a set of edges $R^{i'}$ for $i' < i$. We show that if we remove $R^{i'}$ from G_i, then we can remove one set from the GFG. If we now remove $R^{i''}$ for $i'' < i$ we can remove another set from the GFG and so on. Full proofs are given in Appendix D.

7 Conclusion and Future Work

We introduced a definition of nondeterministic automata that can be used for game monitoring. We accompanied our definition with a construction that takes an NBW and constructs a GFG NPW with $2^n n^{2n}$

states. In comparison, the DRW constructed by Safra has $12^n \cdot n^{2n}$ states. In addition, the structure of the NPW is much simpler and we suggested that it be implemented symbolically.

We also suggest an incremental approach to solving games. The algorithm of Kupferman and Vardi also shares this property [17]. 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 a similar saving. 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 type 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 mentioned 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 currently searching for ways to construct a GFG automaton for the complement language.

References

[1] C. S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. In *10th CIAA*, LNCS. Springer-Verlag, 2005.

[2] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *SSC*, pp 469–474, 1998.

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

[4] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *29th FOCS*, pp 328–337, 1988.

[5] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *32nd FOCS*, pp 368–377, 1991.

[6] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *5th CAV*, LNCS 697, pp 385–396, 1993. Springer-Verlag.

[7] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *1st LICS*, pp 267–278, 1986.

[8] Y. Gurevich and L. Harrington. Trees, automata, and games. In *14th STOC*, pp 60–65. ACM, 1982.

[9] T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *8th Concur*, LNCS 1243, pp 273–287, 1997. Springer-Verlag.

[10] F. Horn. Streett games on finite graphs. In *2nd GDV*, 2005.

[11] A. Harding, M. Ryan, and P.Y. Schobbens. A new algorithm for strategy synthesis in ltl games. In *11th TACAS*, LNCS 3440, pp 477–492. Springer, 2005.

[12] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *17th CAV*, LNCS 3576, pp 226–238. Springer-Verlag, 2005.

[13] M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pp 290–301. Springer-Verlag, 2000.

[14] J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. In *10th CIAA*, LNCS. Springer-Verlag, 2005.

[15] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. *IC*, 200(1):35–61, 2005.

[16] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *30th STOC*, pp 224–233, 1998.

[17] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *46th FOCS*, 2005.

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

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

[20] K.L. McMillan. *Symbolic Model Checking*. 1993.

[21] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *TCS*, 32:321–330, 1984.

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

[23] R. Milner. An algebraic definition of simulation between programs. In *2nd IJCAI*, pp 481–489. 1971.

[24] N. Piterman. From nondeterminstic Büchi and Streett automata to deterministic parity automata. submitted.

[25] P. Madhusudan R. Alur, S. La Torre. Playing games with boxes and diamonds. In *14th Concur*, LNCS 2761, pp 127–141, 2003. Springer-Verlag.

[26] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *AMS*, 141:1–35, 1969.

[27] M.O. Rabin. Automata on infinite objects and Church's problem. *AMS*, 1972.

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

[29] S. Safra. On the complexity of ω-automata. In *29th FOCS*, pp 319–327, 1988.

[30] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *24th STOC*, 1992.

[31] S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic omega-automata. In *8th CHARME*, LNCS 987, pp 261–277, 1995. Springer-Verlag.

[32] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *JCSS*, 32(2):182–221, 1986.

A   Proofs from Section 3

We prove Theorem 3.1.

Proof: One direction is simple. If player 0 wins from $(v, m_0, 0)$ it is simple enough to show that she wins from v. We take the projection of her strategy on the states of G.

In the other direction we have to show that whenever player 0 wins from v in G she wins from $(v, m_0, 0)$ in $G \times \mathcal{M}$. We do this by combining the winning strategy in G with the winning strategy in the monitor game over $\mathcal{M}$. Let $f : V^* \cdot V_0 \to V$ be a winning strategy for player 0 in G. Let $g : (M \cdot \Sigma)^+ \to M$ be a winning strategy for player 0 in the monitor game over $\mathcal{M}$. Consider the following strategy $h : V'^* \cdot V_0' \to V'$ for player 0 in $G \times \mathcal{M}$. Let $\pi = (v_0, m_0, 0), (v_0, m_1, 1),$ $(v_1, m_1, 0), (v_1, m_2, 1), \ldots, (v_i, m_i, \beta)$ be a play. When $\beta = 0$ we have $|\pi|$ is odd and we denote by π_0 the sequence $m_0 v_0 m_1 v_1 \cdots m_i v_i$. Notice that π_0 is a prefix of a play in the monitor game. When $\beta = 1$ we have $|\pi|$ is even and we denote by π_1 the sequence $v_0 v_1 \cdots v_i$. Notice that π_1 is a prefix of a play in G.

$$h(\pi) = \begin{cases} g(\pi_0) & \beta = 0 \\ f(\pi_1) & \beta = 1 \end{cases}$$

Consider an h-conform infinite play $\pi = (v_0, m_0, 0),$ $(v_0, m_1, 1), \ldots$. Let π_0 and π_1 denote the limit of the projections defined above. We have to show that π_0 is an accepting run of $\mathcal{M}$. It can be seen that π_1 is an f-conform play in G and that π_0 is a g-conform play in the monitor game over $\mathcal{M}$. As f is a winning strategy, it follows that $\pi_1 \in L(\mathcal{N})$ or equivalently $\pi_1 \in L(\mathcal{M})$. As g is a winning strategy, it follows that $r(\pi_0) = \pi_0 \Downarrow_M$ is an accepting run of $\mathcal{M}$ on $w(\pi_0) = \pi_0 \Downarrow_V = \pi_1$.   □

B   Definitions and Proofs for Section 5

B.1   From NBW to DRW

Here we describe Safra's determinization [29]. The construction takes an NBW and produces an equivalent DRW.

Let $\mathcal{N} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$ be an NBW with $|S| = n$. Let $V = [n]$ and $V' = \{n+1, \ldots, 2n\}$. A *Safra tree* t over S is a tuple $\langle N, 1, p, \psi, l, E, F \rangle$ where the components of t are as follows.

- $N \subseteq V$ is a set of nodes.
- $1 \in N$ is the root node.
- $p : N \to N$ is the parenthood function. We call children of the same node *siblings.*
- $\psi : N \to N$ is a partial order defining "older than" on siblings.

- $l : N \to 2^S$ is a labeling of the nodes with subsets of S.
- $E, F \subseteq V$ are two disjoint subsets of V, defining the set of *red* and *green* nodes.

In addition we require that the label of every node is a proper superset of the union of the labels of its children and that the labels of two siblings are disjoint.

Claim B.1 [29, 17] *The number of nodes in a Safra tree is at most n. The number of Safra trees over S is at most $12^n n^{2n}$.*

Proof: As the labels of siblings are disjoint and the union of labels of children is a proper subset of the label of the parent it follows that every node is the minimal (according to the subset order on the labels) to contain (at least) some state $s \in S$. It follows that there are at most n nodes.

The number of ordered trees on n nodes is the nth Catalan number. That is $Cat(n) = \frac{(2n)!}{n!(n+1)!} \leq 4^n$. We represent the naming of nodes by $f : [n] \to [n]$ that associates the ith node with its name $f(i)$. There are at most n^n such functions. The labeling function is $l : S \to [n]$ where $l(s) = i$ means that s belongs to the ith node and all its ancestors. Finally, we represent E and F by a function $a : V \to \{0, 1, 2\}$ such that $a(i) = 0$ means that $i \notin E \cup F$, $a(i) = 1$ means that $i \in E$, and $a(i) = 2$ means that $i \in F$. There are at most 3^n such functions.

To summarize, the number of trees is at most $4^n \cdot 3^n \cdot n^n \cdot n^n = 12^n \cdot n^{2n}$.   □

We construct the DRW $\mathcal{D}$ equivalent to $\mathcal{N}$. Let $\mathcal{D} = \langle \Sigma, D, \rho, d_0, \alpha' \rangle$ where the components of $\mathcal{D}$ are as follows.

- D is the set of Safra trees over S.
- d_0 is the tree with a single node 1 labeled $\{s_0\}$ where $E = V - \{1\}$ and $F = \emptyset$.
- Let $\alpha' = \{\langle L_1, U_1 \rangle, \ldots, \langle L_n, U_n \rangle\}$ be the Rabin acceptance condition where $L_i = \{d \in D \mid i \in F_d\}$ and $U_i = \{d \in D \mid i \in E_d\}$.
- For every tree $d \in D$ and letter $\sigma \in \Sigma$ the transition $d' = \rho(d, \sigma)$ is the result of the following transformations on d. We temporarily use the set V' of nodes.
  1. For every node v with label S' replace S' by $\delta(S', \sigma)$ and set E and F to the empty set.
  2. For every node v with label S' such that $S' \cap \alpha \neq \emptyset$, create a new youngest child $v' \in V'$. Set its label to $S' \cap \alpha$.
  3. For every node v with label S' and state $s \in S'$ such that s belongs also to the label of an older sibling v' of v, remove s from the label of v and all its descendants.

4. For every node v whose label is equal to the union of the labels of its children, remove all descendants of v. Add v to F.

5. Remove all nodes with empty labels and add all unused names to E.

6. Change the nodes in V' to nodes in V.

Claim B.2 [29] $L(\mathcal{D}) = L(\mathcal{N})$.

Theorem B.3 [29] *Given an NBW $\mathcal{N}$ with n states we can construct a DRW $\mathcal{D}$ with $12^n n^{2n}$ states and n pairs, such that $L(\mathcal{N}) = L(\mathcal{D})$.*

B.2 Proof of Lemma 5.1

Proof: We show first that $L(\mathcal{N}) \subseteq L(\mathcal{P})$. Consider a word $w \in \Sigma^\omega$ and an accepting run $r = s_0 s_1 \cdots$ of $\mathcal{N}$ on w. Consider the run $r' = q_0 q_1 \ldots$ of $\mathcal{R}$ where $q_i = \langle (A^i, B^i), (\emptyset, \emptyset), \ldots, (\emptyset, \emptyset) \rangle$. We set $A^i = \{s_i\}$. We set $B^i = A_i \cap \alpha$ if $B^{i-1} = \emptyset$ and $B^i = \emptyset$ if $B^{i-1} \neq \emptyset$. That is, the run of $\mathcal{R}$ uses the first set to follow the singletons in the run of $\mathcal{N}$. As r visits α infinitely often, r' visits F_0 infinitely often and r' is accepting.

We have to show that $L(\mathcal{P}) \subseteq L(\mathcal{N})$. Consider a word $w = w_0 w_1 \cdots \in \Sigma^\omega$ and an accepting run $r' = q_0 q_1 \cdots$ of $\mathcal{P}$ on w where forall $i \geq 0$ we have $q_i = \langle (A_1^i, B_1^i), \ldots, (A_n^i, B_n^i) \rangle$. Let $2k$ be the minimal index such that F_{2k} is visited infinitely often.

We first prove two claims.

Claim B.4 *For every $i \in \mathbb{N}$, $j \in [n]$, and every state $s \in A_j^i$ we have s is reachable from s_0 reading $w[0, i-1]$.*

Proof: We prove the claim for all $j \geq 1$ by induction on i. Clearly, it holds for $i = 0$. Suppose that it holds for i. As $A_j^{i+1} \subseteq \rho(A_{j'}^i, w_i)$ for some j' (either j is 1 and j' is 1, j' is j and $A_j^i \neq \emptyset$, or j' is 1) it follows that every state in A_j^{i+1} is reachable from s_0 reading $w[0, i]$. $\square$

Claim B.5 *Consider $i, i' \in \mathbb{N}$ such that $i < i'$ and $q_i, q_{i'} \in F_{2j}$ for some j and forall $j' < 2j$ and forall $i < a < i'$ we have $q_a \notin F_{j'}$. Then every state in $A_{j+1}^{i'}$ is reachable from some state in A_{j+1}^i reading $w[i, i'-1]$ with a run that visits α.*

Proof: By assumption, for every $j' < 2j$ the set $F_{j'}$ is not visited between i and i'. Hence, for $j'' \leq j+1$ and for $i \leq a \leq i'$ we have $A_{j''}^a \neq \emptyset$. It follows that forall $i \leq a < i'$ we have $(A_{j+1}^{a+1}, B_{j+1}^{a+1}) \in succ((A_{j+1}^a, B_{j+1}^a), w_a)$. We show that for every a such that $i < a \leq i'$ every state in B_{j+1}^a is reachable from some state in A_{j+1}^i along a run visiting $\mathcal{F}$. As $B_{j+1}^{i+1} = \rho(A_{j+1}^i, w_i) \cap \alpha$

this is obviously true for $i+1$. Suppose it is true for a and prove for $a+1$. We know $B_j^{a+1} = \rho(B_{j+1}^a, w_a) \cup (A_{j+1}^{a+1} \cap \alpha)$. So every state in B_{j+1}^{a+1} is either a successor of a state in B_{j+1}^a or is a state in α. As $A_{j+1}^{i'} = B_{j+1}^{i'}$ the claim follows. $\square$

We construct an infinite tree with finite branching degree. The root of the tree corresponds to the initial state of $\mathcal{N}$. Every node in the tree is labeled by some state of $\mathcal{N}$ and a time stamp i. An edge between the nodes labeled by (s, i) and (t, j) corresponds to a run starting in s, ending in t, reading $w[i, j-1]$, and visiting α. From König's lemma this tree contains an infinite branch. The composition of all the run segments in this infinite branch is an infinite accepting run of $\mathcal{N}$ on w.

Let $(s_0, 0)$ label the root of t. Let i be the maximal location such that forall $j < 2k$ the set F_j is not visited after i. Let i' be the minimal location such that $i' > i$ and $A_{k+1}^{i'} = B_{k+1}^{i'}$. For every state s in $A_{k+1}^{i'}$ we add a node to t, label it by (s, i') and connect it to the root. We extend the tree by induction. We have a tree with leafs labeled by the states in A_{k+1}^a stamped by time a, and $A_{k+1}^a = B_{k+1}^a$. That is, for every state s in A_{k+1}^a there exists a leaf labeled (s, a). We also know that A_{k+1}^a is not empty. We know that F_{2k} is visited infinitely often. Hence, there exists $a' > a$ such that $A_{k+1}^{a'} = B_j^{k'} \neq \emptyset$. For every state s' in $A_{k+1}^{a'}$ there exists a state s in A_{k+1}^a such that s' is reachable form s reading $w[a, a'-1]$. We add (s', a') as a son of (s, a). From Claim B.4 it follows that every edge $(s_0, 0), (s', i')$ corresponds to some run starting in s_0, ending in s', and reading $w[0, i'-1]$. From Claim B.5, every other edge in the tree $(s, a), (s', a')$ corresponds to some run starting in s, ending in s', reading $w[a, a'-1]$, and visiting α. From König's lemma there exists an infinite branch in the tree. This infinite branch corresponds to an accepting run of $\mathcal{N}$ on w. $\square$

B.3 Proof of Lemma 5.2

Proof: The simulation relation H associates a state d of $\mathcal{D}$ with a state q of $\mathcal{P}$ if the label of the root in d is a subset of the first set in q and the labels of the sons of the root is a subset of the marked states in the first set in q. Formally, $H = \{(d, q) \mid l(1) \subseteq A_1 \text{ and } \bigcup_{i>1} l(i) \subseteq B_1\}$.

We establish that H is a fair-simulation. During simulation, we maintain a permutation $\pi : [n] \to [n]$ that associates the ith set in the state of the DPW with a node $\pi(i)$ in the tree-state of the DRW. The permutation π is similar to the *index appearance record*

[30]. While playing the fair-simulation game, this permutation is updated according to the changes done to the state of $\mathcal{D}$ and to the state of $\mathcal{P}$. Consider two states d and q such that $d \leq_f q$. Let $d = \langle N, 1, p, \psi, l, F, E \rangle$ and $q = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$. Let $d' = \langle N', 1, p', \psi', l', F', E' \rangle$ be $\delta(d, \sigma)$. Let π be the permutation that maintains the association between d and q. We set π' to the permutation that is obtained from π by moving all the nodes in E' to the end of π' so that nodes in $E' \cap N'$ appear before nodes in $E' - N'$. We choose $q' \in \eta(q, \sigma)$ such that $d' \leq_f q'$ in a way that best mimics the transition from d to d'. We choose $q' = \langle (A'_1, B'_1), \ldots, (A'_n, B'_n) \rangle$ so that forall i either $A'_i = l'(\pi(i))$ and B'_i is the union of the labels of the descendants of $\pi(i)$ in d' or $A'_i = \emptyset$.

For a node $v \in d$ we introduce the notation $l(\preceq v)$ to denote the union of the labels of sons of v. Formally, $l(\preceq v) = \bigcup_{v':p(v')=v} l(v')$. We distinguish between two possible moves in going from q to q'. The first type of move is *initialization* where we set q' in a structure that is similar to that of d' according to π'. The second type of move is *simulation* where we have q and d of a similar structure according to π and q' follows the transition from d to d' according to π'.

The initialization move sets A'_1 to $l'(1)$ and B'_1 to $l'(\preceq 1)$. Forall $i > 1$ we set $A'_i = B'_i = \emptyset$. As $d \leq_f q$ it follows that $l(1) \subseteq A_1$ and $l(\preceq 1) \subseteq B_1$ so this is a legal move of $\mathcal{P}$. We assume that π is the permutation so that $\pi(i) = i'$ and i' is the ith active node in d according to the numbers of the nodes. We set π' according to the update policy explained above.

We explain now a simulation move. A simulation move starts from states d and q such that forall i we have either $A_i = l(\pi(i))$ and $B_i = l(\preceq \pi(i))$ or $A_i = B_i = \emptyset$. We build the simulation move so that d' and q' maintains the same invariant according to π'. We first set the permutation π'. The permutation π' is obtained from π by moving all the values i such that $i \in E'$ to the end. We keep the values in $E' \cap N$ before the values in $E' - N$. For example, if $\pi = 1234$, $N = \{1, 3, 4\}$, and $E' = \{2, 4\}$ then $\pi' = 1342$. We are now ready to handle the sets in q'. We handle them according to their order.

- If $\pi(i)$ was removed during the transition to d' or is not in d' (i.e, $\pi(i) \in E'$), we set $A_{i'} = B_{i'} = \emptyset$ forall $i' \geq i$ (notice that we use $\pi(i)$ and not $\pi'(i)$). Clearly, this is a legal transition in $\mathcal{P}$.
- Otherwise, it is the case that $\pi(i)$ remains in d' and so do $\pi(i')$ forall $i' < i$. It follows that $\pi'(i) = \pi(i)$.
  - Consider the case that $\pi(i)$ exists in d and in d'. It must be the case that $A_i = l(\pi(i)) \neq \emptyset$ and $B_i = l(\preceq \pi(i))$. We set $A'_i = l'(\pi(i))$ and $B'_i = l'(\preceq \pi(i))$. In the case that $B'_i = \emptyset$

in d' node i has no descendants. As forall $i' \leq i$ we already set $A'_{i'} = l'(\pi(i'))$ and $B'_{i'} = l'(\preceq \pi(i'))$ this is a legal transition of in $\mathcal{P}$.
  - If $A_i = \emptyset$ and $l'(\pi(i)) \neq \emptyset$ then we set $A'_i = l'(\pi(i))$ and $B'_i = l'(\preceq \pi(i))$. In this case, we can choose a subset of the successors of A_1 and choose freely which states to mark. It follows that this is a legal transition in $\mathcal{P}$.

It is simple to see that starting from states $d \leq_f q$ such that forall i we have either $A_i = l(\pi(i))$ and $B_i = l(\preceq \pi(i))$ or $A_i = B_i = \emptyset$ we produce states $d' \leq_f q'$ that maintain the same invariant and in addition $A'_i = \emptyset$ in the case that some $i' \leq i$ is removed in the transition from d to d'.

We show that the choice of successors as above associates a fair run of $\mathcal{D}$ with a fair run of $\mathcal{P}$. Consider a state q_0 of $\mathcal{P}$ such that $d_0 \leq q_0$. Let $r = d_0 d_1 \cdots$ where $d_i = \langle N_i, 1, p_i, l_i, f_i, e_i \rangle$ be an accepting run of $\mathcal{D}$ on $w = w_0 w_1 \cdots$ and let $r' = q_0 q_1 \cdots$ be the run of $\mathcal{P}$ constructed according to the above strategy. Let $q_i = \langle (A_1^i, B_1^i), \ldots, (A_n^i, B_n^i) \rangle$. Let $\pi_0 \pi_1 \cdots$ be the sequence of permutations that maintain the association between the sets of q_i and the nodes in d_i. By assumption r is accepting. By construction, forall $i > 0$ we have $A_1^i = l_i(1)$ and $B_1^i = l_i(\preceq 1)$.

Let $\langle L_k, U_k \rangle$ be the pair according to which r is accepting. That is, L_k appears infinitely often in r and U_k finitely often. It follows that from some point onwards the node k is always in d_i. Formally, there exists i' such that forall $i > i'$ we have $k \in N_i$ and $k \notin E_i$. In addition, for every $i > i'$ there exists $j > i$ such that $k \in F_j$. Equivalently, in the transition from d_{j-1} to d_j step 4 is applied to node k.

Consider the sequence of permutations $\pi_0 \pi_1 \cdots$. As forall $i > i'$ we have $k \notin E_i$, it follows that forall $i > i'$ we have $\pi_i^{-1}(k)$ does not increase. Hence, there exists some $o' > i'$ such that forall $o > o'$ we have $\pi_o^{-1}(k) = p$ for some p.

By the strategy above, forall $o > o'$ and for every $p' \leq p$ we have $l_i(\pi_i(p')) = A_{p'}^i \neq \emptyset$ and $l_i(\preceq \pi_i(p')) = B_{p'}^i$. As forall $o > o'$ and forall $p' \leq p$ we have $A_{p'}^i \neq \emptyset$ it follows that F_{2l+1} for $2l + 1 < 2p + 2$ are visited finitely often in r'. However, L_k is visited infinitely often in r. Forall $o \geq o'$ we have $B_p^i = l_i(\preceq \pi_i(p))$ and L_k is visited in r when node $k = \pi(p)$ has no descendants. It follows that infinitely often $B_p^i = \emptyset$ and F_{2p+2} is visited infinitely often in r'.

We conclude that r' is an accepting run of $\mathcal{P}$. $\square$

C Proof of Lower Bound

Claim C.1 *Player 0 wins from every state in G.*

13

Proof: Player 0 uses as memory a permutation $\pi \in \Pi$. Let $+1$ impose some cyclic order on Π. The strategy of player 0 is in state 1 choose the successor π and then increase the memory to $\pi + 1$.

We show that this strategy is winning. Suppose that infinitely often player 1 chooses sequences from different permutations during the passage of the regions $[n]^2$. Clearly, by Lemma 5.4 the sequence of labels is accepted by N_n. Suppose that eventually player 0 always chooses the same permutation π. As player 0 tries all permutations in cyclic order, she infinitely often chooses the same permutation π. Then player 1 chooses some pair in π in reverse order. It follows that some pair in π appears infinitely often in the play in reverse order and by Lemma 5.4 the sequence of labels is accepted by N_n. $\square$

Claim C.2 *Winning G requires $n!$ memory values.*

Proof: Suppose that player 0 can use at most $n! - 1$ memory values. Then, one of the outgoing edges from state 1 cannot be used. Let π be this unreachable permutation. Player 1 chooses the permutation π in the first section. Then player 0 chooses some permutation $\pi' \neq \pi$. Let $\pi = i_0 \cdots i_n$ and $\pi' = i'_0 \cdots i'_n$. Then there exists $l < m$ and $l' < m'$ such that $i_l = i'_{m'}$ and $i_m = i'_{l'}$. Player 1 chooses $i'_{m'}$ and then $i'_{l'}$. Thus, player 1 chooses a pair that conforms with the order imposed by π. By Lemma 5.4 player 1 wins. $\square$

D Proof of Incremental Construction

Claim D.1 *Forall $i \geq 0$, player 0 wins from every location in G_i.*

Proof: We prove the claim by induction on i. In G_0 it is easy to see that player 0 wins.

Assume that player 0 wins from every location in G_{i-1}. Consider an infinite play in G_i. Either, the play eventually stays in G_{i-1} and is winning for player 0 by the induction assumption. Otherwise, there are two options, either the play eventually remains in state s_2^i or the play visits s_3^i infinitely often. In the first case, the run of N_i that waits until the play gets stuck in s_2^i and then goes to state $2i - 2$ and remains there is winning. Thus, such plays are won by player 0. In the second case, the run of N_i that visits $2i + 2$ whenever the play is in s_3^i and is in $2i - 1$ at all other times is winning. Again, such plays are won by player 0. $\square$

Let P_i denote the GFG constructed from N_i with $i + 1$ sets according to the construction in Section 5. We show that P_i is sufficient in order to determine that player 0 wins from every state in G_i. For a game G, we say that G is *won* by player 0 and that player 0 *wins* G if player 0 wins from every state in G.

Claim D.2 *Forall $i \geq 0$, player 0 wins the parity game $G_i \times P_i$.*

Proof: We prove the claim by induction on i. For the case $i = 0$ the claim holds. Indeed, even the product of G_0 and N_0 is won by player 0.

Suppose that the product of G_{i-1} and P_{i-1} is won by player 0. We show that the product of G and P_i is won by player 0. We have $i + 1$ sets in P_i. The first set is used to monitor the reachable states in N_i. While the play stays in G_{i-1} we use the rest i sets in P_i in order to mimic the behavior in the product of G_{i-1} and P_{i-1}. Whenever, the play leaves G_{i-1} and enters $S^i = V^i - V^{i-1}$ there are two options. In case that the play ends in s_2^i and stays there forever, we can use the first set to follow the state $2i$ of N_i and win. In case that the play returns to G_{i-1} via s_3^i then we keep in the first set of P_i only the state $2i + 2$. At this point all the states in the first set are marked and P_i visits the set F_0.

A play that visits s_3^i infinitely often is winning for player 0 as F_0 is visited infinitely often. Otherwise, a play eventually remains in G_{i-1} and is won by induction. $\square$

Let q_0 denote the initial state of P_i. By definition q_0 uses just the first set that contains the initial state of N_i. It follows that q_0 can be viewed as the initial state of all the automata $P_{i'}$ for $i' \leq i$.

Claim D.3 *Forall $i \geq 1$, player 1 wins from (s_0^0, q_0) in $G_i \times P_{i-1}$.*

Proof: We prove the claim for the case $i = 1$. Consider the game $G_1 \times P_0$. The automaton P_0 uses one set of marked states. The play starts in the state (s_0^0, q_0) and q_0 follows the reachable states in N_1. As long as the play stays in s_0^0 the reachable states in N_1 are $0, 1$ and 1 is unmarked. The winning strategy of player 1 is to stay in s_0^0 as long as the first (and only) set in P_0 follows the state 1 of N_1. As long as this is the case, the first set in P_0 is not marked as accepting. In order to mark the first set in P_0 accepting, player 0 has to choose to remove the state 1 of N_1. Once this is done, player 1 chooses to go to s_1^1 and the single set of P_0 becomes empty. It follows that player 0 looses from (s_0^0, q_0).

Suppose that the claim holds for $i - 1$ and prove for i. The game starts in (s_0^0, q_0). As long as the first set in P_{i-1} is used to monitor the full set of reachable state in N_i player 1 stays in G_{i-1}. It follows that the

state $2i+1$ of $\mathcal{N}_i$ is followed in the first set and this set cannot be marked accepting. As long as the first set of $\mathcal{P}_{i-1}$ contains $2i+1$ player 1 remains in G_{i-1}. It follows that player 0 can use only $i-1$ sets in $\mathcal{P}_{i-1}$ to follow a play in G_{i-1}. By induction, player 0 looses. If at some point player 0 decides to give up on following the state $2i+1$ in the first set of $\mathcal{P}_{i-1}$, player 1 immediately goes to s_1^i. The first set in $\mathcal{P}_{i-1}$ gets empty and player 0 looses. $\qquad\square$

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

Claim D.4 *Forall $I \subseteq [i]$ such that $i \in I$ and $|I| = j$, player 0 wins the parity game $G_i^I \times \mathcal{P}_j$.*

Proof: For G_1 we demand that $1 \in I$ so this is equivalent to Claim D.2. Consider the game $G_2^{\{2\}}$ and the automaton $\mathcal{P}_1$ (with two sets). The winning strategy of player 0 is to use the first set to monitor the reachable states (i.e., maintain the state 3 of $\mathcal{N}_2$). While the play is in S^0 the second set follows the state 0 of $\mathcal{N}_2$. While the play is in S^1 the second set follows the states $1, 2$ of $\mathcal{N}_2$. Whenever the play visits S^2 the first set is reduced to follow the state 4 of $\mathcal{N}_2$ and is marked accepting. If S^2 is visited infinitely often then F_0 is visited infinitely often and the play is winning for player 0. If S^2 is visited finitely often then F_1 is visited finitely often (set 2 is empty) and F_2 is visited infinitely often (set 2 is fully marked).

Consider the game G_i^I. Let i' be the maximal in $I'=I-\{i\}$ and $j'=|I'| = j-1$. By induction, the product of $G_{i'}^{I'}$ and $\mathcal{P}_{j'}$ is won by player 0. We use the first set in $\mathcal{P}_j$ to follow the set of reachable states of $\mathcal{N}_i$. As long as the play is in $G_{i'}$ we use the rest of the sets to simulates $\mathcal{P}_{j'}$. Once the play goes to $S^{i''}$ for $i' < i'' < i$ we throw away the information in all the sets but the first set. We use the second set to follow the states $2i'' + 2$ and $2i'' + 1$. If the play visits infinitely often S^i then F_0 is visited infinitely often and player 0 wins. If the play eventually stays in $G_{i'}$ then $\mathcal{P}_{j'}$ is sufficient by induction. If the play eventually stays in $G_{i''}$ for $i' < i'' < i$ then one set in addition to the first set is sufficient. $\qquad\square$

Claim D.5 *Forall $I \subseteq [i]$ such that $i \in I$ and $j = |I|-1$, player 1 wins from (s_0^0, q_0) in $G_i^I \times \mathcal{P}_j$.*

Proof: Consider the game $G_2^{\{2\}}$ and the automaton $\mathcal{P}_0$. Clearly, player 1 wins.

Consider a set $I \subseteq [i]$ such that $i \in I$. Let $i' < i$ be the maximal in $I-\{i\}$. Let $I' = I-\{i\}$ and $j' = |I'| = j-1$. By induction, player 1 wins from (s_0^0, q_0) in the product $G_{i'}^{I'} \times \mathcal{P}_{j'}$. As before, when playing in $G_i^I \times \mathcal{P}_j$ the first set in $\mathcal{P}_j$ must be used to follow the reachable states in $\mathcal{N}_i$. We use the induction assumption to give a winning strategy for player 1. $\qquad\square$