A Hierarchy of Nondeterminism

Bader Abu Radi ✉
School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel

Orna Kupferman ✉
School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel

Ofer Leshkowitz ✉
School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel

Abstract

We study three levels in a hierarchy of nondeterminism: A nondeterministic automaton $\mathcal{A}$ is *determinizable by pruning* (DBP) if we can obtain a deterministic automaton equivalent to $\mathcal{A}$ by removing some of its transitions. Then, $\mathcal{A}$ is *good-for-games* (GFG) if its nondeterministic choices can be resolved in a way that only depends on the past. Finally, $\mathcal{A}$ is *semantically deterministic* (SD) if different nondeterministic choices in $\mathcal{A}$ lead to equivalent states. Some applications of automata in formal methods require deterministic automata, yet in fact can use automata with some level of nondeterminism. For example, DBP automata are useful in the analysis of online algorithms, and GFG automata are useful in synthesis and control. For automata on finite words, the three levels in the hierarchy coincide. We study the hierarchy for Büchi, co-Büchi, and weak automata on infinite words. We show that the hierarchy is strict, study the expressive power of the different levels in it, as well as the complexity of deciding the membership of a language in a given level. Finally, we describe a probability-based analysis of the hierarchy, which relates the level of nondeterminism with the probability that a random run on a word in the language is accepting.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Automata on Infinite Words, Expressive power, Complexity, Games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.85

1 Introduction

Nondeterminism is a fundamental notion in theoretical computer science. It allows a computing machine to examine several possible actions simultaneously. For automata on finite words, nondeterminism does not increase the expressive power, yet it leads to an exponential succinctness [25].

A prime application of automata theory is specification, verification, and synthesis of reactive systems [29, 15]. Since we care about the on-going behavior of nonterminating systems, the automata run on infinite words. Acceptance in such automata is determined according to the set of states that are visited infinitely often along the run. In *Büchi* automata [9], the acceptance condition is a subset α of states, and a run is accepting iff it visits α infinitely often. Dually, in *co-Büchi* automata, a run is accepting iff it visits α only finitely often. We also consider *weak* automata, which are a special case of both Büchi and co-Büchi automata in which no cycle contains both states in α and states not in α. We use three-letter acronyms in $\{$D, N$\} \times \{$F, B, C, W$\} \times \{$W$\}$ to describe the different classes of automata. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second for the acceptance condition type (finite, Büchi, co-Büchi or weak); and the third indicates that we consider automata on words.

For automata on infinite words, nondeterminism may increase the expressive power and also leads to an exponential succinctness. For example, NBWs are strictly more expressive than DBWs [19], whereas NCWs are as expressive as DCWs [21]. In some applications of the automata-theoretic approach, such as model checking, algorithms can

46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021).
Editors: Filippo Bonchi and Simon J. Puglisi; Article No. 85; pp. 85:1–85:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

be based on nondeterministic automata, whereas in other applications, such as synthesis and control, they cannot. There, the advantages of nondeterminism are lost, and algorithms involve a complicated determinization construction [26] or acrobatics for circumventing determinization [18]. Essentially, the inherent difficulty of using nondeterminism in synthesis and control lies in the fact that each guess of the nondeterministic automaton should accommodate all possible futures.

A study of nondeterministic automata that can resolve their nondeterministic choices in a way that only depends on the past started in [16], where the setting is modeled by means of tree automata for derived languages. It then continued by means of *good for games* (GFG) automata [12].[1] A nondeterministic automaton $\mathcal{A}$ over an alphabet Σ is GFG if there is a strategy g that maps each finite word $u \in \Sigma^*$ to the transition to be taken after u is read; and following g results in accepting all the words in the language of $\mathcal{A}$. Note that a state q of $\mathcal{A}$ may be reachable via different words, and g may suggest different transitions from q after different words are read. Still, g depends only on the past, namely on the word read so far. Obviously, there exist GFG automata: deterministic ones, or nondeterministic ones that are *determinizable by pruning* (DBP); that is, ones that just add transitions on top of a deterministic automaton. In fact, the GFG automata constructed in [12] are DBP.[2] Beyond the theoretical interest in DBP automata, they are used for modelling online algorithms: by relating the "unbounded look ahead" of optimal offline algorithms with nondeterminism, and relating the "no look ahead" of online algorithms with determinism, it is possible to reduce questions about the competitive ratio of online algorithms and the memory they require to questions about DBPness [2, 3].

In terms of expressive power, it is shown in [16, 24] that GFG-NXWs, for $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}\}$, are as expressive as DXWs. For automata on finite words, GFG-NFWs are always DBP [16, 22]. For automata on infinite words, GFG-NBWs and GFG-NCWs need not be DBP [5]. Moreover, the best known determinization construction for GFG-NBWs is quadratic, and determinization of GFG-NCWs has a tight exponential blow-up [14]. Thus, GFG automata on infinite words are (possibly even exponentially) more succinct than deterministic ones. Further research studies characterization, typeness, complementation, and further constructions and decision procedures for GFG automata [14, 7, 4], as well as an extension of the GFG setting to pushdown ω-automata [20] and to alternating automata [8, 6].

A nondeterministic automaton is *semantically deterministic* (SD, for short) if its nondeterministic choices lead to states with the same language. Thus, for every state q of the automaton and letter $\sigma \in \Sigma$, all the σ-successors of q have the same language. Beyond the fact that semantically determinism is a natural relaxation of determinism, and thus deserves consideration, SD automata naturally arise in the setting of GFG automata. Indeed, though not all GFG automata are DBP, it is not hard to see that they can all be pruned to an SD automaton [14]. Moreover, such a pruning can be done in polynomial time, and so we assume, without loss of generality, that all GFG automata are SD. Thus, SD can be thought also as a natural relaxation of GFG.

Thus, we obtain the following hierarchy, from deterministic to nondeterministic automata, where each level is a special case of the levels to its right.

[1] GFGness is also used in [11] in the framework of cost functions under the name "history-determinism".
[2] As explained in [12], the fact that the GFG automata constructed there are DBP does not contradict their usefulness in practice, as their transition relation is simpler than the one of the embodied deterministic automaton and it can be defined symbolically.

For automata on finite words, all levels of the hierarchy coincide in their expressive power. In fact, the three internal levels coincide already in the syntactic sense: every SD-NFW is DBP. Also, given an NFW, deciding whether it is SD, GFG or DBP, can each be done in polynomial time [2].

For Büchi and co-Büchi automata, the picture is less clear, and is the subject of our research. Before we describe our results, let us mention that an orthogonal level of determinism is that of *unambiguous* automata, namely automata that have a single accepting run on each word in their languages. An unambiguous NFW need not be SD, and a DBP-NFW need not be unambiguous. It is known, however, that a GFG unambiguous NFW, NCW, or NBW, is DBP [7].

We study the following aspect and questions about the hierarchy.

Strictness. Recall that not all GFG-NBWs and GFG-NCWs are DBP [5], and examples for this include also SD automata. On the other hand, all GFG-NWWs (in fact, all GFG-NXWs whose language can be recognized by a DWW) are DBP [7]. We show that SD-NXWs need not be GFG for all $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$. Of special interest is our result on weak automata, whose properties typically agree with these of automata on finite words. Here, while all SD-NFWs are GFG, this is not the case for SD-NWWs.

Expressive power. It is known that for all $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$, GFG-NXWs are as expressive as DXWs. We extend this result to semantic determinism and show that while SD-NXWs need not be GFG, they are not more expressive, thus SD-NXWs are as expressive as DXWs. Since an SD-NXW need not be GFG, this extends the known frontier of nondeterministic Büchi and weak automata that are not more expressive than their deterministic counterpart.

Deciding the determinization level of an automaton. It is already known that deciding the GFGness of a given NXW, for $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$, can be done in polynomial time [2, 14, 4]. On the other hand, deciding whether a given NCW is DBP is NP-complete [13]. We complete the picture in three directions. First, we show that NP-completeness of deciding DBPness applies also to NBWs. Second, we show that in both cases, hardness applies even when the given automaton is GFG. Thus, while it took the community some time to get convinced that not all GFG automata are DBP, in fact it is NP-complete to decide whether a given GFG-NBW or GFG-NCW is DBP. Third, we study also the problem of deciding whether a given NXW is SD, and show that it is PSPACE-complete. Note that our results imply that the nondeterminism hierarchy is not monotone with respect to complexity: deciding DBPness, which is closest to determinism, is NP-complete, then GFGness can be checked in polynomial time, and finally SDness is PSPACE-complete. Also, as PSPACE-hardness of checking SDness applies already to NWWs, we get another, even more surprising, difference between weak automata and automata on finite words. Indeed, for NFWs, all the three levels of nondeterminism coincide and SDness can be checked in polynomial time.

A probability-based analysis of the different levels. Consider a nondeterministic automaton $\mathcal{A}$. We say that $\mathcal{A}$ is *almost-DBP* if we can prune transitions from $\mathcal{A}$ and obtain a deterministic automaton $\mathcal{A}'$ such that the probability of a random word to be in $L(\mathcal{A}) \setminus L(\mathcal{A}')$ is 0. Thus, while $\mathcal{A}'$ need not accept all the words in $L(\mathcal{A})$, it rejects only a negligible fragment of $L(\mathcal{A})$. Clearly, if $\mathcal{A}$ is DBP, then it is almost-DBP. A typical analysis of the performance of an on-line algorithm compares its performance with that of an off-line algorithm. The notion of almost-DBPness captures cases where the on-line algorithm performs, with probability 1, as good as the offline algorithm. We study the almost-DBPness of GFG and SD automata. We show that while for Büchi (and hence also weak) automata, semantic determinism implies almost-DBPness, thus every SD-NBW is almost-DBP, for co-Büchi automata semantic determinism is not enough, and we need GFGness. Thus, there is an SD-NCW that is not almost-DBP, yet all GFG-NCWs are almost-DBP.

2 Preliminaries

2.1 Automata

For a finite nonempty alphabet Σ, an infinite *word* $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of letters from Σ. A *language* $L \subseteq \Sigma^\omega$ is a set of infinite words. For $i, j \geq 0$, we use $w[1, i]$ to denote the (possibly empty) prefix $\sigma_1 \cdot \sigma_2 \cdots \sigma_i$ of w, use $w[i+1, j]$ to denote the (possibly empty) infix $\sigma_{i+1} \cdot \sigma_{i+2} \cdots \sigma_j$ of w, and use $w[i+1, \infty]$ to denote its suffix $\sigma_{i+1} \cdot \sigma_{i+2} \cdots$. We sometimes refer also to languages of finite words, namely subsets of Σ^*. We denote the empty word by ϵ.

A *nondeterministic automaton* over infinite words is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ is an alphabet, Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \to 2^Q \setminus \emptyset$ is a *transition function*, and α is an *acceptance condition*, to be defined below. For states q and s and a letter $\sigma \in \Sigma$, we say that s is a σ-successor of q if $s \in \delta(q, \sigma)$. Note that $\mathcal{A}$ is *total*, in the sense that it has at least one successor for each state and letter. If $|\delta(q, \sigma)| = 1$ for every state $q \in Q$ and letter $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*.

A *run* of $\mathcal{A}$ on $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $r = r_0, r_1, r_2, \ldots \in Q^\omega$, such that $r_0 = q_0$, and for all $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, \sigma_{i+1})$. We extend δ to sets of states and finite words in the expected way. Thus, $\delta(S, u)$ is the set of states that $\mathcal{A}$ may reach when it reads the word $u \in \Sigma^*$ from some state in $S \in 2^Q$. Formally, $\delta : 2^Q \times \Sigma^* \to 2^Q$ is such that for every $S \in 2^Q$, finite word $u \in \Sigma^*$, and letter $\sigma \in \Sigma$, we have that $\delta(S, \epsilon) = S$, $\delta(S, \sigma) = \bigcup_{s \in S} \delta(s, \sigma)$, and $\delta(S, u \cdot \sigma) = \delta(\delta(S, u), \sigma)$. The transition function δ induces a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, where for every two states $q, s \in Q$ and letter $\sigma \in \Sigma$, we have that $\langle q, \sigma, s \rangle \in \Delta$ iff $s \in \delta(q, \sigma)$. For a state $q \in Q$ of $\mathcal{A}$, we define $\mathcal{A}^q$ to be the automaton obtained from $\mathcal{A}$ by setting the initial state to be q. Thus, $\mathcal{A}^q = \langle \Sigma, Q, q, \delta, \alpha \rangle$.

The acceptance condition α determines which runs are "good". We consider here the *Büchi* and *co-Büchi* acceptance conditions, where $\alpha \subseteq Q$ is a subset of states. We use the terms α-*states* and $\bar{\alpha}$-*states* to refer to states in α and in $Q \setminus \alpha$, respectively. For a run r, let $\mathit{inf}(r) \subseteq Q$ be the set of states that r traverses infinitely often. Thus, $\mathit{inf}(r) = \{q \in Q : q = r_i$ for infinitely many i's$\}$. A run r of a Büchi automaton is *accepting* iff it visits states in α infinitely often, thus $\mathit{inf}(r) \cap \alpha \neq \emptyset$. Dually, a run r of a co-Büchi automaton is accepting iff it visits states in α only finitely often, thus $\mathit{inf}(r) \cap \alpha = \emptyset$. A run that is not accepting is *rejecting*. Note that as $\mathcal{A}$ is nondeterministic, it may have several runs on a word w. The word w is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Two automata are *equivalent* if their languages are equivalent.

Consider a directed graph $G = \langle V, E \rangle$. A *strongly connected set* in G (SCS, for short) is a set $C \subseteq V$ such that for every two vertices $v, v' \in C$, there is a path from v to v'. A SCS is *maximal* if it is maximal w.r.t containment, that is, for every non-empty set $C' \subseteq V \setminus C$, it holds that $C \cup C'$ is not a SCS. The *maximal strongly connected sets* are also termed *strongly connected components* (SCCs, for short). The *SCC graph of G* is the graph defined over the

SCCs of G, where there is an edge from an SCC C to another SCC C' iff there are two vertices $v \in C$ and $v' \in C'$ with $\langle v, v' \rangle \in E$. A SCC is *ergodic* iff it has no outgoing edges in the SCC graph.

An automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ induces a directed graph $G_\mathcal{A} = \langle Q, E \rangle$, where $\langle q, q' \rangle \in E$ iff there is a letter $\sigma \in \Sigma$ such that $\langle q, \sigma, q' \rangle \in \Delta$. The SCSs and SCCs of $\mathcal{A}$ are those of $G_\mathcal{A}$. The *α-free SCCs* of $\mathcal{A}$ are the SCCs of $\mathcal{A}$ that do not contain states from α.

A Büchi automaton $\mathcal{A}$ is *weak* [23] if for each SCC C in $G_\mathcal{A}$, either $C \subseteq \alpha$ (in which case we say that C is an accepting SCC) or $C \cap \alpha = \emptyset$ (in which case we say that C is a rejecting SCC). Note that a weak automaton can be viewed as both a Büchi and a co-Büchi automaton, as a run of $\mathcal{A}$ visits α infinitely often, iff it gets trapped in an accepting SCC, iff it visits states in $Q \setminus \alpha$ only finitely often.

We denote the different classes of automata by three-letter acronyms in $\{$D, N$\} \times \{$F, B, C, W$\} \times \{$W$\}$. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second for the acceptance condition type (finite, Büchi, co-Büchi or weak); and the third indicates that we consider automata on words. For example, NBWs are nondeterministic Büchi word automata.

2.2 Probability

Consider the probability space $(\Sigma^\omega, \mathbb{P})$ where each word $w = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdots \in \Sigma^\omega$ is drawn by taking the σ_i's to be independent and identically distributed Unif(Σ). Thus. for all positions $i \geq 1$ and letters $\sigma \in \Sigma$, the probability that σ_i is σ is $\frac{1}{|\Sigma|}$. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ be a determnistic automaton, and let $G_\mathcal{A} = \langle Q, E \rangle$ be its induced directed graph. A *random walk on* $\mathcal{A}$, is a random walk on $G_\mathcal{A}$ with the probability matrix $P(q,p) = \frac{|\{\sigma \in \Sigma : \langle q, \sigma, p \rangle \in \Delta\}|}{|\Sigma|}$. It is not hard to see that $\mathbb{P}(L(\mathcal{A}))$ is precisely the probability that a random walk on $\mathcal{A}$ is an accepting run. Note that with probability 1, a random walk on $\mathcal{A}$ reaches an ergodic SCC $C \subseteq Q$, where it visits all states infinitely often. It follows that $\mathbb{P}(L(\mathcal{A}))$ equals the probability that a random walk on $\mathcal{A}$ reaches an ergodic accepting SCC.

2.3 Automata with Some Nondeterminism

Consider a nondeterministic automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. We say that two states $q, s \in Q$ are *equivalent*, denoted $q \sim_\mathcal{A} s$, if $L(\mathcal{A}^q) = L(\mathcal{A}^s)$. Then, $\mathcal{A}$ is *semantically deterministic* (SD, for short) if different nondeterministic choices in $\mathcal{A}$ lead to equivalent states. Thus, for every state $q \in Q$ and letter $\sigma \in \Sigma$, all the σ-successors of q are equivalent: for every two states $s, s' \in \delta(q, \sigma)$, we have that $s \sim_\mathcal{A} s'$.

An automaton $\mathcal{A}$ is *good for games* (*GFG*, for short) if its nondeterminism can be resolved based on the past, thus on the prefix of the input word read so far. Formally, $\mathcal{A}$ is *GFG* if there exists a *strategy* $f : \Sigma^* \to Q$ such that the following hold:

1. The strategy f is consistent with the transition function. That is, $f(\epsilon) = q_0$, and for every finite word $u \in \Sigma^*$ and letter $\sigma \in \Sigma$, we have that $\langle f(u), \sigma, f(u \cdot \sigma) \rangle \in \Delta$.
2. Following f causes $\mathcal{A}$ to accept all the words in its language. That is, for every infinite word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$, if $w \in L(\mathcal{A})$, then the run $f(w[1,0]), f(w[1,1]), f(w[1,2]), \ldots$, which we denote by $f(w)$, is an accepting run of $\mathcal{A}$ on w.

We say that the strategy f *witnesses* $\mathcal{A}$'s GFGness. For an automaton $\mathcal{A}$, we say that a state q of $\mathcal{A}$ is *GFG* if $\mathcal{A}^q$ is GFG. Note that every deterministic automaton is GFG. Also, every GFG automaton can be made SD. Indeed, removal of transitions that are not used by a strategy that witnesses $\mathcal{A}$'s GFGness does not reduce the language of $\mathcal{A}$ and results in an SD automaton. Moreover, by [14, 4], the detection of such transitions can be done in polynomial time.

We say that a nondeterministic automaton $\mathcal{A}$ is *determinizable by prunning* (DBP) if we can remove some of the transitions of $\mathcal{A}$ and get a deterministic automaton $\mathcal{A}'$ that recognizes $L(\mathcal{A})$. We then say that $\mathcal{A}'$ is a *deterministic pruning* of $\mathcal{A}$. Note that every DBP nondeterministic automaton is GFG. Indeed, the deterministic pruning of $\mathcal{A}$ induces a witness strategy.

3 The Different Levels and Their Expressive Power

In this section we study syntactic and semantic hierarchies induced by the different levels of nondeterminism. For two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ of automata, we use $\mathcal{C}_1 \preceq \mathcal{C}_2$ to indicate that every automaton in $\mathcal{C}_1$ is also in $\mathcal{C}_2$. Accordingly, $\mathcal{C}_1 \prec \mathcal{C}_2$ if $\mathcal{C}_1 \preceq \mathcal{C}_2$ yet there are automata in $\mathcal{C}_2$ that are not in $\mathcal{C}_1$. We first show that the nondeterminism hierarchy is strict, except for all GFG-NWWs being DBP. The latter is not surprising, as all GFG-NFWs are DBP. On the other hand, unlike the case of finite words, we show that not all SD-NWWs are GFG. In fact the result holds already for NWWs that accept co-safety languages, namely all whose states except for an accepting sink are rejecting.

▶ **Theorem 1** (Syntactic Hierarchy). *For* $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$*, we have that DXW* $\prec$ *DBP-NXW* $\preceq$ *GFG-NXW* $\prec$ *SD-NXW* $\prec$ *NXW. For* $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}\}$*, the second inequality is strict.*

Proof. By definition, each class is a special case of the one to its right. We prove strictness. It is easy to see that the first and last strict inequalities hold. Indeed, for all $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$, consider a nonempty DXW $\mathcal{A}$, and obtain an NXW $\mathcal{B}$ by adding to $\mathcal{A}$ a σ-transition from the initial state to a new rejecting state, for a letter σ such that $\mathcal{A}$ accepts some word that starts with σ. Then, $\mathcal{B}$ is a DBP-NXW that is not a DXW. Also, as at least one σ-successor of the initial state of $\mathcal{A}$ is not empty, $\mathcal{B}$ is an NXW that is not a SD-NXW.

The relation between DBPness and GFGness has already been studied. It is shown in [5] that GFG-NXW need not be DBP for $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}\}$, and shown in [7] that GFG-NWW are DBP. It is left to relate GFGness and SDness. Consider the NWW $\mathcal{W}$ in Figure 1. It is not hard to check that $\mathcal{W}$ is indeed weak, it is SD, as all its states recognize the language $\{a, b\}^\omega$, yet is not GFG, as every strategy has a word with which it does not reach q_{acc} – a word that forces each visit in q_a and q_b to be followed by a visit in q_0.

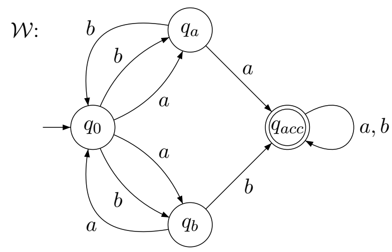

Figure 1 An SD-NWW that is not GFG.

Hence GFG-NWW $\prec$ SD-NWW. As weak automata are a special case of Büchi and co-Büchi, strictness for them follows. ◀

We continue to study expressive power. Now, for two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ of automata, we say that $\mathcal{C}_1$ is less expressive than $\mathcal{C}_2$, denoted $\mathcal{C}_1 \leq \mathcal{C}_2$, if every automaton in $\mathcal{C}_1$ has an equivalent automaton in $\mathcal{C}_2$. Since NCW=DCW, we expect the hierarchy to be strict only in the cases of Büchi and weak automata. As we now show, however, semantically deterministic automata are not more expressive than deterministic ones also in the case of Büchi and weak auotmata.

▶ **Theorem 2** (Expressiveness Hierarchy). *For $X \in \{B, W\}$, we have that DXW = DBP-NXW = GFG-NXW = SD-NXW < NXW.*

Proof. In [16, 14], the authors suggest variants of the subset construction that determinize GFG-NBWs. As we argue below, the construction in [14] is correct also when applied to SD-NBWs. Moreover, it preserves weakness. Thus, DBW=SD-NBW and DWW=SD-NWW. Also, the last inequality follows from the fact DBW<NBW and DWW<NWW [19].

Given an NBW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, the DBW generated in [14][3] is $\mathcal{A}' = \langle \Sigma, Q', q_0', \delta', \alpha' \rangle$, where $Q' = 2^Q$, $q_0' = \{q_0\}$, $\alpha' = \{S \in 2^Q : S \subseteq \alpha\}$, and the transition function δ' is defined for every subset $S \in 2^Q$ and letter $\sigma \in \Sigma$ as follows. If $\delta(S, \sigma) \cap \alpha = \emptyset$, then $\delta'(S, \sigma) = \delta(S, \sigma)$. Otherwise, namely if $\delta(S, \sigma) \cap \alpha \neq \emptyset$, then $\delta'(S, \sigma) = \delta(S, \sigma) \cap \alpha$.

The key observation about the correctness of the construction is that when $\mathcal{A}$ is an SD-NBW, then for all reachable states S of $\mathcal{A}'$, we have that $q \sim_{\mathcal{A}} q'$ for all states $q, q' \in S$. Indeed, if $\mathcal{A}$ is SD, then for every two states $q, q' \in Q$, letter $\sigma \in \Sigma$, and transitions $\langle q, \sigma, s \rangle, \langle q', \sigma, s' \rangle \in \Delta$, if $q \sim_{\mathcal{A}} q'$, then $s \sim_{\mathcal{A}} s'$. Also, by the definition of δ', every reachable state S of $\mathcal{A}'$ contains only α-states or only $\bar{\alpha}$-states. As we formally prove in Appendix A, these properties guarantee that indeed $L(\mathcal{A}') = L(\mathcal{A})$ and that weakness of $\mathcal{A}$ is maintained in $\mathcal{A}'$. ◀

4 Deciding the Nondeterminism Level of an Automaton

In this section we study the complexity of the problem of deciding the nondeterminism level of a given automaton. Note we refer here to the syntactic class (e.g., deciding whether a given NBW is GFG) and not to the semantic one (e.g., deciding whether a given NBW has an equivalent GFG-NBW). Indeed, by Theorem 2, the latter boils down to deciding whether the language of a given NXW can be recognized by a DXW, which is well known: the answer is always "yes" for an NCW, and the problem is PSPACE-complete for NBWs and NWWs [17].[4]

Our results are summarized in Table 1. The entries there describe both the case in which the given automaton is a general NXW, and the case in which the given automaton is an NXW that belongs to a level, that is one level to the right of the questioned one (for example, deciding DBPness of a GFG automaton). In fact, the complexity of the two cases coincide, with one exception: deciding whether a given NWW is DBP, which is PTIME in general, and is $O(1)$ when the given NWW is GFG, in which case the answer is always "yes".

▶ **Theorem 3.** *Deciding whether an NXW is semantically deterministic is PSPACE-complete, for* $\mathrm{X} \in \{\mathrm{B}, \mathrm{C}, \mathrm{W}\}$.

Proof. Membership in PSPACE is easy, as we check SDness by polynomially many checks of language equivalence. Formally, given an NXW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, a PSPACE algorithm goes over all states $q \in Q$, letters σ, and σ-successors s and s' of q, and checks that $s \sim_{\mathcal{A}} s'$. Since language equivalence can be checked in PSPACE [28] and there are polynomially many checks to perform, we are done.

[3] The construction in [14] assumes automata with transition-based acceptance, and (regardless of this) is slightly different: when α is visited, $\mathcal{A}'$ continues with a single state from the set of successors. The key point, however, is the same: $\mathcal{A}$ being SD enables $\mathcal{A}'$ to maintain only subsets of states, rather than Safra trees, which makes determinization much easier.

[4] The proof in [17] is for NBWs, yet the arguments there apply also for weak automata.

Table 1 Deciding the level of an NXW. The results are valid also in the case the given NXW is one level to the right in the nondeterminism hierarchy. Two exceptions are the cases of deciding the DBPness of a GFG NCW and GFG NWW, where the results are specified in ( ).

	DBP	GFG	SD
NBW	NP-complete Th. 4	PTIME [4]	PSPACE-complete Th. 3
NCW	NP-complete [13] (Th. 8)	PTIME [14]	PSPACE-complete Th. 3
NWW	PTIME ($O(1)$) [14, 4]([7])	PTIME [14, 4]	PSPACE-complete Th. 3

Proving PSPACE-hardness, we do a reduction from polynomial-space Turing machines. Given a Turing machine T with space complexity $s : \mathbb{N} \to \mathbb{N}$, we construct in time polynomial in $|T|$ and $s(0)$, an NWW $\mathcal{A}$ of size linear in T and $s(0)$, such that $\mathcal{A}$ is SD iff T accepts the empty tape[5]. Clearly, this implies a lower bound also for NBWs and NCWs. Let $n_0 = s(0)$. Thus, each configuration in the computation of T on the empty tape uses at most n_0 cells. We assume, without loss of generality, that once T reaches a final (accepting or rejecting) state, it erases the tape, moves with its reading head to the leftmost cell, and moves to the initial state. Thus, all computations of T are infinite and after visiting a final configuration for the first time, they consists of repeating the same finite computation on the empty tape that uses n_0 tape cells.

We define $\mathcal{A}$ so that it accepts a word w iff (C1) w is not a suffix of an encoding of a legal computation of T that uses at most n_0 cells, or (C2) w includes an encoding of the initial configuration of T on the empty tape and the final configuration after it, is accepting.

It is not hard to see that if T accepts the empty tape, then $\mathcal{A}$ is universal (that is, accepts all words). Indeed, each word w is either not a suffix of an encoding of a legal computation of T that uses at most n_0 cells, in which case w is accepted thanks to C1. Otherwise, the encoding of the computation of T on the empty tape is a subword of w, in which case w eventually includes an encoding of the initial configuration of T on the empty tape, and the final configuration after it is accepting, and thus w is accepted thanks to C2. Also, if T rejects the empty tape, then $\mathcal{A}$ rejects the word that encodes the computation of T on the empty tape. Indeed, C1 is not satisfied, and since every encoding of the initial configuration is followed by an encoding of the rejecting computation of T, the final configuration after it is rejecting, and so C2 is not satisfied too.

In order to define $\mathcal{A}$ so that it is SD iff T accepts the empty tape, we define all its states to be universal iff T accepts the empty tape. Intuitively, we do it by letting $\mathcal{A}$ guess and check the existence of an infix that witnesses satisfaction of C1 or C2, and also let it, at each point of its operation, go back to the initial state, where it can guess again. Note that when T accepts the empty tape, all the suffixes of a word w satisfy C1 or C2. Thus, $\mathcal{A}$ making a bad guess does not prevent it from later branching into an accepting run.

[5] This is sufficient, as one can define a generic reduction from every language L in PSPACE as follows. Let T_L be a Turing machine that decides L in polynomial space $f(n)$. On input w for the reduction, the reduction considers the machine T_w that on every input, first erases the tape, writes w on its tape, and then runs as T_L on w. Then, the reduction outputs an automaton $\mathcal{A}$, such that T_w accepts the empty tape iff $\mathcal{A}$ is SD. Note that the space complexity of T_w is $s(n) = \max(n, f(|w|))$, and that w is in L iff T_w accepts the empty tape. Since $\mathcal{A}$ is constructed in time polynomial in $s(0) = f(|w|)$ and $|T_w| = \text{poly}(|w|)$, it follows that the reduction is polynomial in $|w|$.

We now describe the operation of $\mathcal{A}$ in more detail (see Figure 2).

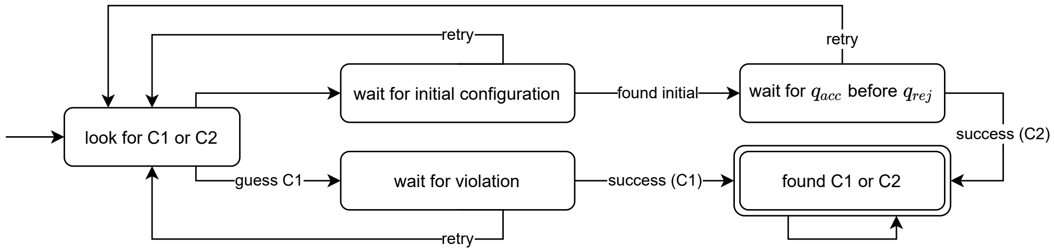

Figure 2 The structure of the NWW constructed in Theorem 3.

In its initial state, $\mathcal{A}$ guesses which of C1 and C2 is satisfied. In case $\mathcal{A}$ guesses that C1 is satisfied, it guesses the place in which w includes a violation of the encoding. As we detail in Appendix B, this amounts to guessing a violation of the transition function of T: in each step, $\mathcal{A}$ may guess that the next three letters encode a position in a configuration and the letter to come n_0 letters later, namely at the same position in the successive configuration, is different from the one that should appear in a legal encoding of two successive configurations. If a violation is detected, $\mathcal{A}$ moves to an accepting sink. Otherwise, $\mathcal{A}$ returns to the initial state and w gets another chance to be accepted. In case $\mathcal{A}$ guesses that C2 is satisfied, it guesses the place in which w encodes an initial configuration. If $\mathcal{A}$ guesses a position of an initial configuration, but the guess fails, then $\mathcal{A}$ goes back to the initial state. If the guess succeeds, $\mathcal{A}$ waits for an accepting configuration of T. If an accepting configuration arrives before a rejecting one, then $\mathcal{A}$ moves to an accepting sink. Otherwise, if a rejecting configuration arrives before an accepting one, then $\mathcal{A}$ returns to the initial state. Also, whenever $\mathcal{A}$ waits to witness some behavior, namely, waits to guess a position of an initial state, waits to guess a position of a violation, or waits to see a final configuration, it may nondeterministically, upon reading the next letter, return to the initial state. It is not hard to see that $\mathcal{A}$ can be defined in size linear in T and n_0. As the only accepting states of $\mathcal{A}$ is the accepting sink, it is clearly weak, and in fact describes a co-safety language.

We prove that T accepts the empty tape iff $\mathcal{A}$ is SD. First, if T rejects the empty tape, then $\mathcal{A}$ is not SD. To see this, consider the word w_ε that encodes the computation of T on the empty tape, and let w'_ε be a word that is obtained from w_ε by making a single violation in the first letter. That is, $w'_\varepsilon[2,\infty] = w_\varepsilon[2,\infty]$, and $w'_\varepsilon[1,1] \neq w_\varepsilon[1,1]$. Note that $w'_\varepsilon \in L(\mathcal{A})$ since it has a violation. Note also that any proper suffix of w'_ε encodes a suffix of a computation of T that uses at most n_0 tape cells and does not have of a final accepting configuration, and hence is not in $L(\mathcal{A})$. Consequently, the word w'_ε can be accepted by $\mathcal{A}$ only by guessing a violation that is caused by the first letter. In particular, if we guess to wait for the initial configuration upon reading the first letter, then we cannot branch to an accepting run. This shows that $\mathcal{A}$ is not SD. For the other direction, we show that if T accepts the empty tape, then all the states of $\mathcal{A}$ are universal. First, note that each infinite word w is either not a suffix of a legal encoding of a computation of T that uses at most n_0 tape cells, in which case it is in the language of $\mathcal{A}$ by C1, or it is a suffix of a legal encoding of a computation that uses only n_0 tapes cells, and is eventually an encoding of the computation of T on the empty tape, in which case, as T accepts the empty tape, w is in the language of $\mathcal{A}$ according to C2. Thus, the initial state of $\mathcal{A}$ is universal. Now by the definition of $\mathcal{A}$, for every infinite word w and for all states q of $\mathcal{A}$ that are not the accepting sink, there is a path from q to the initial state that is labeled by a prefix of w. Thus, the language of all states is universal, and they are all equivalent. This clearly implies that $\mathcal{A}$ is SD.

Thus, we conclude that T accepts the empty tape iff $\mathcal{A}$ is SD. In Appendix B, we give the full technical details of the construction of $\mathcal{A}$. ◀

▶ **Theorem 4.** *The problem of deciding whether a given GFG-NBW is DBP is NP-complete.*

Proof. For membership in NP, observe we can check that a witness deterministic pruning $\mathcal{A}'$ is equivalent to $\mathcal{A}$ by checking whether $L(\mathcal{A}) \subseteq L(\mathcal{A}')$. Since $\mathcal{A}'$ is deterministic, the latter can be checked in polynomial time. For NP-hardness, we describe a parsimonious polynomial time reduction from SAT. That is, given a CNF formula φ, we construct a GFG-NBW $\mathcal{A}_\varphi$ such that there is a bijection between assignments to the variables of φ and DBWs embodied in $\mathcal{A}_\varphi$, and an assignment satisfies φ iff its corresponding embodied DBW is equivalent to $\mathcal{A}_\varphi$. In particular, φ is satisfiable iff $\mathcal{A}_\varphi$ is DBP.

Consider a SAT instance φ over the variable set $X = \{x_1, \dots, x_n\}$ and with $m \geq 1$ clauses $C = \{c_1, \dots, c_m\}$. For $n \geq 1$, let $[n] = \{1, 2, \dots, n\}$. For a variable $x_k \in X$, let $C_k^0 \subseteq C$ be the set of clauses in which x_k appears negatively, and let $C_k^1 \subseteq C$ be the set of clauses in which x_k appears positively. For example, if $c_1 = x_1 \vee \neg x_2 \vee x_3$, then c_1 is in C_1^1, C_2^0, and C_3^1. Assume that all clauses depend on at least two different variables (that is, no clause is a tautology or forces an assignment to a single variable). Let $\Sigma_{n,m} = X \cup C$, and let $R_{n,m} = (X \cdot C)^* \cdot \{x_1 \cdot c_j \cdot x_2 \cdot c_j \cdots x_n \cdot c_j : j \in [m]\} \subseteq \Sigma_{n,m}^*$. We construct a GFG-NBW $\mathcal{A}_\varphi$ that recognizes $L_{n,m} = (R_{n,m})^\omega$, and is DBP iff φ is satisfiable.

Let $D_{n,m}$ be a DFW that recognizes $R_{n,m}$ with $O(n \cdot m)$ states, a single accepting state p, and an initial state q_0 that is visited only once in all runs. For example, we can define $D_{n,m} = \langle \Sigma_{n,m}, Q_{n,m}, q_0, \delta_{n,m}, \{p\} \rangle$ as follows: from q_0, the DFW expects to read only words in $(X \cdot C)^*$ – upon a violation of this pattern, it goes to a rejecting sink. Now, if the pattern is respected, then with $X \setminus \{x_1\}$, the DFW goes to two states where it loops with $C \cdot (X \setminus \{x_1\})$ and, upon reading x_1 from all states that expect to see letters in X, it branches with each c_j, for all $j \in [m]$, to a path where it hopes to detect an $x_2 \cdot c_j \cdots x_n \cdot c_j$ suffix. If the detection is completed successfully, it goes to the accepting state p. Otherwise, it returns to the two-state loop.

Now, we define $\mathcal{A}_\varphi = \langle \Sigma_{n,m}, Q_\varphi, p, \delta_\varphi, \{q_0\} \rangle$, where $Q_\varphi = Q_{n,m} \cup \{q_k^i : (i,k) \in \{0,1\} \times [n]\}$. The idea behind $\mathcal{A}_\varphi$ is as follows. From state p (that is, the accepting state of $D_{n,m}$, which is now the initial state of $\mathcal{A}_\varphi$), the NBW $\mathcal{A}_\varphi$ expects to read a letter in X. When it reads x_k, for $1 \leq k \leq n$, it nondeterministically branches to the states q_k^0 and q_k^1. Intuitively, when it branches to q_k^0, it guesses that the clause that comes next is one that is satisfied when $x_k = 0$, namely a clause in C_k^0. Likewise, when it branches to q_k^1, it guesses that the clause that comes next is one that is satisfied when $x_k = 1$, namely a clause in C_k^1. When the guess is successful, $\mathcal{A}_\varphi$ moves to the α-state q_0. When the guess is not successful, it returns to p. Implementing the above intuition, transitions from the states $Q_{n,m} \setminus \{p\}$ are inherited from $D_{n,m}$, and transitions from the states in $\{q_k^i : (i,k) \in \{0,1\} \times [n]\} \cup \{p\}$ are defined as follows (see also Figure 3).

- For all $k \in [n]$, we have that $\delta_\varphi(p, x_k) = \{q_k^0, q_k^1\}$.
- For all $k \in [n]$, $i \in \{0,1\}$, and $j \in [m]$, if $c_j \in C_k^i$, then $\delta_\varphi(q_k^i, c_j) = \{q_0\}$. Otherwise, $\delta_\varphi(q_k^i, c_j) = \{p\}$. For example, if $c_1 = x_1 \vee \neg x_2 \vee x_3$, then $\delta_\varphi(q_2^0, c_1) = \{q_0\}$ and $\delta_\varphi(q_2^1, c_1) = \{p\}$.

Note that p is the only nondeterministic state of $\mathcal{A}_\varphi$ and that for every deterministic pruning of $\mathcal{A}_\varphi$, all the words in $(X \cdot C)^\omega$ have an infinite run in the pruned automaton. This run, however, may eventually loop in $\{p\} \cup \{q_k^0, q_k^1 : k \in [n]\}$. Note also, that for readability purposes, the automaton $\mathcal{A}_\varphi$ is not total. Specifically, the states of $\mathcal{A}_\varphi$ are partitioned into

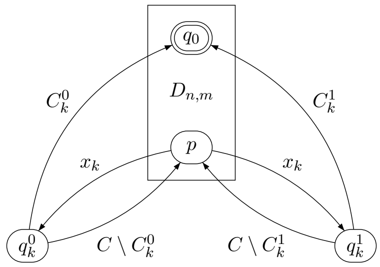

Figure 3 The transitions to and from the states q_k^0 and q_k^1 in $\mathcal{A}_\varphi$.

states that expect to see letters in X and states that expect to see letters in C. In particular, all infinite paths in $\mathcal{A}_\varphi$ are labeled by words in $(X \cdot C)^\omega$. Thus, when defining a GFG strategy g for $\mathcal{A}_\varphi$, we only need to define g on prefixes in $(X \cdot C)^* \cup (X \cdot C)^* \cdot X$.

In the following propositions, we prove that $\mathcal{A}_\varphi$ is a GFG NBW recognizing $L_{n,m}$, and that $\mathcal{A}_\varphi$ is DBP iff φ is satisfiable. ◀

▶ **Proposition 5.** $L(\mathcal{A}_\varphi) \subseteq L_{n,m}$.

Proof. As already mentioned, all infinite paths of $\mathcal{A}_\varphi$, accepting or rejecting, are labeled by words in $(X \cdot C)^\omega$. Further, any accepting run of $\mathcal{A}_\varphi$ has infinitely many sub-runs that are accepting finite runs of $D_{n,m}$. Since $L_{n,m} = (R_{n,m})^\omega = (X \cdot C)^\omega \cap (\infty R_{n,m})$, it follows that $L(\mathcal{A}_\varphi) \subseteq L_{n,m}$. ◀

▶ **Proposition 6.** *There exists a strategy $g : \Sigma^* \to Q_\varphi$ for $\mathcal{A}_\varphi$ that accepts all words in $L_{n,m}$. Formally, for all $w \in L_{n,m}$, the run $g(w) = g(w[1,0]), g(w[1,1]), g(w[1,2]), \ldots,$ is an accepting run of $\mathcal{A}_\varphi$ on w.*

Proof. The definition of $L_{n,m}$ is such that when reading a prefix that ends with a subword of the form $x_1 \cdot c_j$, for some $j \in [m]$, then we can guess that the word continues with $x_2 \cdot c_j \cdot x_3 \cdot c_j \cdots x_n \cdot c_j$; thus that c_j is the clause that is going to repeat. Therefore, when we are at state p after reading a word that ended with $x_1 \cdot c_j$, and we read x_2, it is a good GFG strategy to move to a state q_2^i such that the assignment $x_2 = i$ satisfies c_j (if such $i \in \{0, 1\}$ exists; otherwise the strategy can choose arbitrary between q_2^0 and q_2^1), and if the run gets back to p, the strategy continues with assignments that hope to satisfy c_j, until the run gets to q_0 or another occurrence of x_1 is detected. Note that while it is not guaranteed that for all $k \in [n]$ there is $i \in \{0, 1\}$ such that the assignment $x_k = i$ satisfies c_j, it is guaranteed that such an i exists for at least two different k's (we assume that all clauses depend on at least two variables). Thus, even though we a priori miss an opportunity to satisfy c_j with an assignment to x_1, it is guaranteed that there is another $2 \leq k \leq n$ such that c_j can be satisfied by x_k.

We define g inductively as follows. Recall that $\mathcal{A}_\varphi$ is nondeterministic only in the state p, and so in all other states, the strategy g follows the only possible transition. First, for all $k \in [n]$, we define $g(x_k) = q_k^0$. Let $v \in (X \cdot C)^* \cdot X$, be such that g has already been defined on v and let $j \in [m]$. Since $v \notin (X \cdot C)^*$, we have that $g(v) \neq p$ and so $g(v \cdot c_j)$ is uniquely defined. We continue and define g on $u = v \cdot c_j \cdot x_k$, for all $k \in [n]$. If $g(v \cdot c_j) \neq p$, then $g(u)$ is uniquely defined. Otherwise, $g(v \cdot c_j) = p$ and we define $g(u)$ as follows,

- If $k = 1$, then we define $g(u) = q_1^0$.
- If $k > 1$ and x_k participates in c_j, then we define $g(u) = q_k^i$, where $i \in \{0, 1\}$ is minimal with $c_j \in C_k^i$. That is, i is the minimal assignment to x_k that satisfies c_j.
- If $k > 1$ and x_k does not participate in c_j, then the value of c_j is not affected by the assignment to x_k, and in that case we define $g(u) = q_k^0$.

The reason for the distinction between the cases $k = 1$ and $k > 1$ is that when we see a finite word that ended with $c_j \cdot x_1$, then there is no special reason to hope that the next letter is going to be c_j. This is in contrast, for example, to the case we have seen a word that ends with $c_{j'} \cdot x_1 \cdot c_j \cdot x_2$, where it is worthwhile to guess we are about to see c_j as the next letter.

By the definition of g, it is consistent with Δ_φ. In Appendix C we formally prove that g is a winning GFG strategy for $\mathcal{A}_\varphi$. Namely, that for all $w \in L_{n,m}$, the run $g(w)$ on w, generated by g is accepting. ◀

We now examine the relation between prunings of $\mathcal{A}_\varphi$ and assignments to φ. Consider an assignment $i_1, \ldots, i_n \in \{0,1\}$, for X. I.e., $x_k = i_k$ for all $k \in [n]$. Then a possible memoryless GFG strategy, is to always move from p to $q_k^{i_k}$ when reading x_k. This in fact, describes a one to one correspondence, between assignments and prunings of $\mathcal{A}_\varphi$. Assume that the assignment $i_k \in \{0,1\}$, for $k \in [n]$, satisfies φ, then the corresponding pruning recognizes $L_{n,m}$. Indeed, instead of trying to satisfy the last read clause c_j, we may ignore this extra information, and rely on the fact that one of the assignments $x_k = i_k$ is going to satisfy c_j. In other words, the satisfiability of φ allows us to ignore the history and still accept all words in $L_{n,m}$, which makes $\mathcal{A}_\varphi$ DBP. On the other hand, if an assignment does not satisfy some clause c_j, then the corresponding pruning will fail to accept the word $(x_1 \cdot c_j \cdots x_n \cdot c_j)^\omega$, which shows that if φ is not satisfiable then $\mathcal{A}_\varphi$ is not DBP. In Appendix C we formally prove that there is a one to one correspondence between prunings of $\mathcal{A}_\varphi$ and assignments to φ, and that an assignment satisfies φ iff the corresponding pruning recognizes $L_{n,m}$, implying Proposition 7.

▶ **Proposition 7.** *The formula φ is satisfiable iff the GFG-NBW $\mathcal{A}_\varphi$ is DBP.*

We continue to co-Büchi automata. In [13], the authors prove that deciding the DBPness of a given NCW is NP-complete. For the lower bound, they describe a reduction from the *Hamiltonian-cycle* problem. Essentially, given a connected graph $G = \langle [n], E \rangle$, the reduction outputs an NCW $\mathcal{A}_G$ over the alphabet $[n]$ that is obtained from G by adding self loops to all vertices, labelling the loop at a vertex i by the letter i, and labelling the edges from vertex i to all its neighbours in G by every letter $j \neq i$. Then, the co-Büchi condition requires a run to eventually get stuck at a self-loop[6]. Accordingly, $L(\mathcal{A}_G) = [n]^* \cdot \bigcup_{i \in [n]} i^\omega$.

It is not hard to see that $\mathcal{A}_G$ is GFG. Indeed, a GFG strategy can decide to which neighbour of i to proceed with a letter $j \neq i$ by following a cycle c that traverses all the vertices of the graph G. Since when we read $j \neq i$ at vertex i we move to a neighbour state, then by following the cycle c upon reading i^ω, we eventually reach the vertex i and get stuck at the i-labeled loop. Thus, the NP-hardness result of [13] apply already for GFG-NCWs, and we can conclude with the following.

▶ **Theorem 8.** *The problem of deciding whether a given GFG-NCW is DBP is NP-complete.*

5 A probability-Based Analysis of the Different Levels

Consider a nondeterministic automaton $\mathcal{A}$. We say that $\mathcal{A}$ is *almost-DBP* if there is a deterministic pruning $\mathcal{A}'$ of $\mathcal{A}$ such that $\mathbb{P}(L(\mathcal{A}) \setminus L(\mathcal{A}')) = 0$. Thus, while $\mathcal{A}'$ need not accept all the words accepted by $\mathcal{A}$, it rejects only a negligible set of words in $L(\mathcal{A})$. Clearly, if $\mathcal{A}$ is DBP, then it is almost-DBP. In this section we study the almost-DBPness of GFG

[6] The exact reduction is more complicated and involves an additional letter # that forces each deterministic pruning of $\mathcal{A}_G$ to proceed to the same neighbour of i upon reading a letter $j \neq i$ from the vertex i.

and SD automata. We show that while for Büchi (and hence also weak) automata, semantic determinism implies almost-DBPness, thus every SD-NBW is almost-DBP, for co-Büchi automata semantic determinism is not enough, and we need GFGness. Thus, there is an SD-NCW that is not almost-DBP, yet all GFG-NCWs are almost-DBP.

We first show that, unsurprisingly, not all NBWs are almost-DBP.

▶ **Theorem 9.** *There is an NBW that is not almost-DBP.*

Proof. Consider the NBW $\mathcal{A}_1$ in Figure 4. It is not hard to see that $L(\mathcal{A}_1) = \{a,b\}^\omega$, and so $\mathbb{P}(L(\mathcal{A}_1)) = 1$. Moreover, every deterministic pruning of $\mathcal{A}_1$ is such that q_{rej} is reachable from all states, which implies that $\{q_{rej}\}$ is the only ergodic SCC of any pruning. Since $\{q_{rej}\}$ is α-free, it follows that every deterministic pruning of $\mathcal{A}_1$ recognizes a language of measure zero, and hence $\mathcal{A}_1$ is not almost-DBP. As an example, consider the deterministic pruning $\mathcal{A}'_1$ described on the right hand side of Figure 4. The only ergodic SCC of $\mathcal{A}'_1$ is α-free, and as such $\mathbb{P}(L(\mathcal{A}'_1)) = 0$. ◀

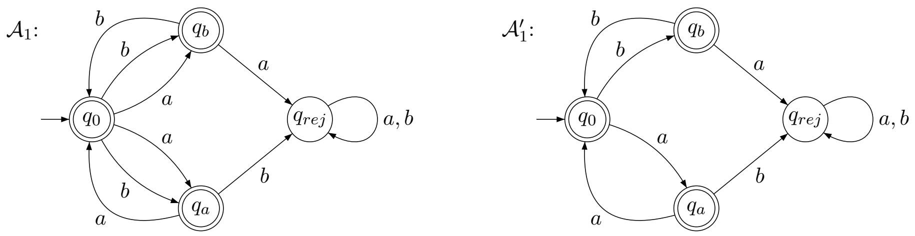

Figure 4 An NBW that is not almost-DBP.

We continue to the positive result about Büchi automata. Consider an NBW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. We define a *simple stochastic Büchi game* $\mathcal{G}_\mathcal{A}$ as follows.[7] The game is played between Random and Eve. The positions of Random are Q, these of Eve are $Q \times \Sigma$. The game starts from position q_0. A round in the game starts at some position $q \in Q$ and proceeds as follows.

1. Random picks a letter $\sigma \in \Sigma$ uniformly, and the game moves to position (q, σ).
2. Eve picks a transition $(q, \sigma, p) \in \Delta$, and the game moves to position p.

A probabilistic strategy for Eve is $f : (Q \times \Sigma)^+ \to [0,1]^Q$, where for all histories $x \in (Q \times \Sigma)^*$ and positions of Eve $(q, \sigma) \in Q \times \Sigma$, the function $d = f(x \cdot (q, \sigma)) : Q \to [0,1]$, is a distribution on Q such that $d(p) \neq 0$ implies that $p \in \delta(q, \sigma)$. As usual, we say that a strategy f is *memoryless*, if it depends only on the current position, thus for all histories $x, y \in (Q \times \Sigma)^*$ and positions of Eve $(q, \sigma) \in Q \times \Sigma$, it holds that $f(x \cdot (q, \sigma)) = f(y \cdot (q, \sigma))$. A strategy for Eve is *pure* if for all histories $x \in (Q \times \Sigma)^*$ and positions of Eve $(q, \sigma) \in Q \times \Sigma$, there is a position $p \in \delta(q, \sigma)$ such that $f(x \cdot (q, \sigma))(p) = 1$. When Eve plays according to a strategy f, the outcome of the game can be viewed as a run $r_f = q_f^0, q_f^1, q_f^2, \ldots$ in $\mathcal{A}$, over a random word $w_f \in \Sigma^\omega$. (The word that is generated in a play is independent of the strategy of Eve, but we use the notion w_f to emphasize that we are considering the word that is generated in a play where Eve plays according to f).

Let $Q_{rej} = \{q \in Q : \mathbb{P}(L(\mathcal{A}^q)) = 0\}$. The outcome r_f of the game is *winning* for Eve iff r_f is accepting, or r_f visits Q_{rej}. Note that for all positions $q \in Q_{rej}$ and $p \in Q$, if p is reachable from q, then $p \in Q_{rej}$. Hence, the winning condition can be defined by the Büchi

[7] In [10] these games are called simple $1\frac{1}{2}$ -player games with Büchi winning objectives and almost-sure winning criterion.

objective $\alpha \cup Q_{rej}$. Note that r_f is winning for Eve iff $\inf(r_f) \subseteq Q_{rej}$ or $\inf(r_f) \cap \alpha \neq \emptyset$. We say that f is an *almost-sure winning* strategy, if r_f is winning for Eve with probability 1, and Eve *almost-sure wins* in $\mathcal{G}_\mathcal{A}$ if she has an almost-sure winning strategy.

▶ **Theorem 10.** *All SD-NBWs are almost-DBP.*

Proof. We first show that Eve has a probabilistic strategy to win $\mathcal{G}_\mathcal{A}$ with probability 1, even without assuming that $\mathcal{A}$ is semantically deterministic. Consider the probabilistic strategy g where from (q, σ), Eve picks one of the σ-successors of q uniformly by random. Note that this strategy is memoryless, and hence the outcome of the game can be thought as a random walk in $\mathcal{A}$ that starts at q_0 and gives positive probabilities to all transitions. Thus, with probability 1, the run r_g is going to reach an ergodic SCC of $\mathcal{A}$ and visit all its states. If the run r_g reaches an α-free ergodic SCC, then $\inf(r_g) \subseteq Q_{rej}$, and hence r_g is then winning for Eve. Otherwise, r_g reaches a non α-free ergodic SCC, and with probability 1, it visits all the states in that SCC. Thus, with probability 1, we have $\inf(r_g) \cap \alpha \neq 0$, and r_g is winning for Eve. Overall, Eve wins $\mathcal{G}_\mathcal{A}$ with probability 1 when playing according to g.

Hence, by pure memoryless determinacy of simple stochastic parity games [10], we may consider a pure memoryless winning strategy f for Eve in $\mathcal{G}_\mathcal{A}$. We say that r_f *is correct* if $w_f \in L(\mathcal{A})$ implies that r_f is accepting. Note that $w_f \notin L(\mathcal{A})$ always implies that r_f is rejecting. We show that if $\mathcal{A}$ is SD, then r_f is correct with probability 1, where f is a pure memoryless winning strategy for Eve. Since f is pure memoryless, it induces a pruning of $\mathcal{A}$. Denote this pruning by $\mathcal{A}^f$. We may think of r_f as a random walk in $\mathcal{A}^f$. With probability 1, the walk r_f reaches an ergodic SCC C of $\mathcal{A}^f$, and visits all its states. Since f is a winning strategy, we know that $C \subseteq Q_{rej}$ or $C \cap \alpha \neq \emptyset$ with probability 1. If $C \cap \alpha \neq \emptyset$, then clearly r_f is accepting with probability 1, and hence is correct with probability 1. Otherwise, $C \subseteq Q_{rej}$, but then we claim that $w_f \in L(\mathcal{A})$ with probability 0. For $i \geq 1$, let w_f^i be the i-th letter of w_f, and for $i \geq 0$ let q_f^i be the i-th state in r_f. Then, by semantically determinism, for all $i \geq 0$, it holds that $w_f \in L(\mathcal{A})$ iff $w_f[i+1, \infty] \in L(\mathcal{A}^{q_f^i})$. Moreover, the word $w_f[i+1, \infty]$ is independent of q_f^i, and hence for all $q \in Q$ and $i \geq 0$, the event $w_f[i+1, \infty] \in L(\mathcal{A}^q)$ is independent of q_f^i. Thus, for all $q \in Q$ and $i \geq 0$, it holds that $\mathbb{P}(w_f \in L(\mathcal{A}) | q_f^i = q) = \mathbb{P}(w_f[i+1, \infty] \in L(\mathcal{A}^q)) = \mathbb{P}(L(\mathcal{A}^q))$. Hence, by definition of Q_{rej}, and by the fact that Q_{rej} is finite, we have that $\mathbb{P}(w_f \in L(\mathcal{A}) | r_f^i \in Q_{rej}) = 0$ for all $i \geq 0$, and so $\mathbb{P}(w_f \in L(\mathcal{A}) | r_f \text{ visits } Q_{rej}) = 0$. Overall, we showed that $\mathbb{P}(r_f \text{ is correct}) = 1$.

Notice that $\mathbb{P}(L(\mathcal{A}) \setminus L(\mathcal{A}^f))$, is precisely the probability that a random word w_f is in $L(\mathcal{A})$ but not accepted by $\mathcal{A}^f$. Namely, the probability that r_f is not correct. Hence, $\mathbb{P}(L(\mathcal{A}) \setminus L(\mathcal{A}^f)) = 0$, and $\mathcal{A}$ is almost-DBP. ◀

We continue to co-Büchi automata and show that unlike the case of Büchi, here semantic determinism does not imply almost-DBPness.

▶ **Theorem 11.** *There is an SD-NCW that is not almost-DBP.*

Proof. Consider the NCW $\mathcal{A}_2$ in Figure 5. It is not hard to see that $L(\mathcal{A}_2) = \{a, b\}^\omega$, and hence $\mathbb{P}(L(\mathcal{A}_2)) = 1$. In fact all the states q of $\mathcal{A}_2$ have $L(\mathcal{A}_2^q) = \{a, b\}^\omega$, and so it is semantically deterministic. Moreover, every deterministic pruning of $\mathcal{A}_2$ is strongly connected and not α-free. It follows that any deterministic pruning of $\mathcal{A}_2$ recognizes a language of measure zero, and hence $\mathcal{A}_2$ is not almost-DBP. As an example, consider the deterministic pruning $\mathcal{A}_2'$ described on the right hand side of Figure 5. It is easy to see that $\mathcal{A}_2'$ is strongly connected and not α-free, and as such, $\mathbb{P}(L(\mathcal{A}_2')) = 0$. ◀

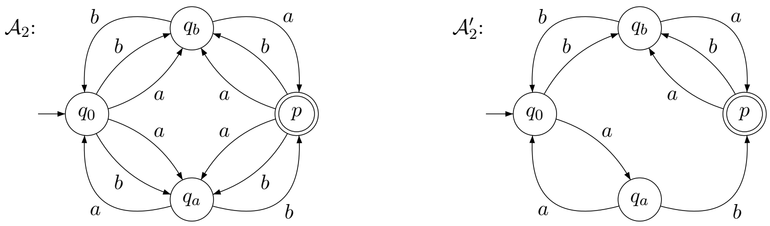

Figure 5 An SD-NCW that is not almost-DBP.

Consider a language $L \subseteq \Sigma^\omega$ of infinite words. We say that a finite word $x \in \Sigma^*$ is a *good prefix for* L if $x \cdot \Sigma^\omega \subseteq L$. Then, L is a *co-safety* language if every word in L has a good prefix [1]. Let $co\text{-}safe(L) = \{x \cdot w \in \Sigma^\omega : x \text{ is a good prefix of } L\}$. Clearly, $co\text{-}safe(L) \subseteq L$. The other direction is not necessarily true. For example, if $L \subseteq \{a,b\}^\omega$ is the set of all words with infinitely many a's, then $co\text{-}safe(L) = \emptyset$. In fact, $co\text{-}safe(L) = L$ iff L is a co-safety language. As we show now, when L is NCW-recognizable, we can relate L and $co\text{-}safe(L)$ as follows.

▶ **Lemma 12.** *If L is NCW-recognizable, then $\mathbb{P}(L(\mathcal{A}) \setminus co\text{-}safe(L(\mathcal{A}))) = 0$.*

Proof. Consider an NCW-recognizable language L. Since NCW=DCW, there is a DCW $\mathcal{D}$ that recognizes $L(\mathcal{A})$. Assume without loss of generality that $\mathcal{D}$ has a single state q with $L(\mathcal{A}^q) = \Sigma^\omega$, in particular, $C = \{q\}$ is the only ergodic α-free SCC of $\mathcal{A}$. Then, for every word $w \in \Sigma^\omega$, we have that $w \in co\text{-}safe(L)$ iff the run of $\mathcal{D}$ on w reaches C. Hence, the probability that $w \in L(\mathcal{A}) \setminus co\text{-}safe(L(\mathcal{A}))$ equals the probability that $inf(r)$ is α-free but is not an ergodic SCC of $\mathcal{D}$. Since the later happens w.p 0, we have that $\mathbb{P}(L(\mathcal{A}) \setminus co\text{-}safe(L(\mathcal{A}))) = 0$. ◀

By Lemma 12, pruning an NCW in a way that would make it recognize $co\text{-}safe(L(\mathcal{A}))$ results in a DCW that approximates $\mathcal{A}$, and thus witnesses that $\mathcal{A}$ is almost-DBP. We now show that for GFG-NCWs, such a pruning is possible, and conclude that GFG-NCWs are almost-DBP.

▶ **Theorem 13.** *All GFG-NCWs are almost-DBP.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ be a GFG-NCW. Consider the NCW $\mathcal{A}' = \langle \Sigma, Q, q_0, \delta, \alpha' \rangle$, where $\alpha' = \alpha \cup \{q \in Q : L(\mathcal{A}^q) \neq \Sigma^\omega\}$. We prove that $\mathcal{A}'$ is a GFG-NCW with $L(\mathcal{A}') = co\text{-}safe(L(\mathcal{A}))$. Consider a word $w \in L(\mathcal{A}')$, and let $r = q_0, q_1, q_2, \ldots$ be an accepting run of $\mathcal{A}'$ on w. There exists a prefix $x \in \Sigma^*$ of w such that r reaches some $q \notin \alpha'$ when reading x. Hence $L(\mathcal{A}^q) = \Sigma^\omega$, and so $x \cdot \Sigma^\omega \subseteq L(\mathcal{A})$. That is, x is a good prefix and $w \in co\text{-}safe(L(\mathcal{A}))$. Thus, $L(\mathcal{A}') \subseteq co\text{-}safe(L(\mathcal{A}))$.

In order to see that $\mathcal{A}'$ is GFG and that $co\text{-}safe(L(\mathcal{A})) \subseteq L(\mathcal{A}')$, we consider a GFG strategy f of $\mathcal{A}$ and use it as a strategy for $\mathcal{A}'$. We need to prove that for all $w \in co\text{-}safe(L(\mathcal{A}))$, the run r that f generates on w eventually visits only states $q \notin \alpha'$. Since $co\text{-}safe(L(\mathcal{A})) \subseteq L(\mathcal{A})$, we know that $\inf(r) \cap \alpha = \emptyset$. It is left to show that r eventually visits only states $q \in Q$ with $L(\mathcal{A}^q) = \Sigma^\omega$. Observe that if $x \in \Sigma^*$ is a good prefix of $L(\mathcal{A})$, then for all $y \in \Sigma^*$, we have that $x \cdot y$ is also a good prefix. Moreover, if $x \in \Sigma^*$ is a good prefix, then since f is a GFG strategy, it follows that for all $u \in \Sigma^\omega$ the run $f(x \cdot u)$ is accepting, and hence $u \in L(\mathcal{A}^{f(x)})$. I.e, $L(\mathcal{A}^{f(x)}) = \Sigma^\omega$. Thus, w has only finitely many bad prefixes, and so $f(x) \in \{q \in Q : L(A^q) \neq \Sigma^\omega\}$ for only finitely many prefixes x of w. That is, $\inf(r) \cap \alpha' = \emptyset$, and f is a GFG strategy for $\mathcal{A}'$.

So, $\mathcal{A}'$ is a GFG-NCW with $L(\mathcal{A}') = \textit{co-safe}(L(\mathcal{A}))$. Since $\textit{co-safe}(L(\mathcal{A}))$ is co-safe, it is DWW-recognizable [27]. By [7], GFG-NCWs whose language is DWW-realizable are DBP. Let δ' be the restriction δ to a deterministic transition function such that $\mathcal{D}' = \langle \Sigma, Q, q_0, \delta', \alpha' \rangle$ is a DCW with $L(\mathcal{D}') = L(\mathcal{A}') = \textit{co-safe}(L(\mathcal{A}))$. Consider now the DCW $\mathcal{D} = \langle \Sigma, Q, q_0, \delta', \alpha \rangle$ that is obtained from $\mathcal{D}'$ by replacing α' with α. It is clear that $\mathcal{D}$ is a pruning of $\mathcal{A}$. Note that, $\alpha \subseteq \alpha'$, and hence $\textit{co-safe}(L(\mathcal{A})) = L(\mathcal{D}') \subseteq L(\mathcal{D})$. That is, $\mathcal{D}$ is a pruning of $\mathcal{A}$ that approximates $L(\mathcal{A})$ up to a negligible set, and $\mathcal{A}$ is almost-DBP. ◀

References

1. B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2:117–126, 1987.
2. B. Aminof, O. Kupferman, and R. Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2), 2010.
3. G. Avni and O. Kupferman. Stochastization of weighted automata. In *40th Int. Symp. on Mathematical Foundations of Computer Science*, volume 9234 of *Lecture Notes in Computer Science*, pages 89–102. Springer, 2015.
4. M. Bagnol and D. Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *Proc. 38th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 122 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
5. U. Boker, D. Kuperberg, O. Kupferman, and M. Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 89–100, 2013.
6. U. Boker, D. Kuperberg, K. Lehtinen, and M. Skrzypczak. On the succinctness of alternating parity good-for-games automata. In *Proc. 40th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs*, pages 41:1–41:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
7. U. Boker, O. Kupferman, and M. Skrzypczak. How deterministic are Good-For-Games automata? In *Proc. 37th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, 2017.
8. U. Boker and K. Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proc. 30th Int. Conf. on Concurrency Theory*, volume 140 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
9. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.
10. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Simple stochastic parity games. In *Proc. 12th Annual Conf. of the European Association for Computer Science Logic*, volume 2803, pages 100–113. Springer, 2003.
11. Th. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proc. 36th Int. Colloq. on Automata, Languages, and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009.
12. T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.
13. D. Kuperberg and A. Majumdar. Width of non-deterministic automata. In *Proc. 35th Symp. on Theoretical Aspects of Computer Science*, volume 96 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
14. D. Kuperberg and M. Skrzypczak. On determinisation of good-for-games automata. In *Proc. 42nd Int. Colloq. on Automata, Languages, and Programming*, pages 299–310, 2015.

15 O. Kupferman. Automata theory and model checking. In *Handbook of Model Checking*, pages 107–151. Springer, 2018.
16 O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006.
17 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.
18 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
19 L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.
20 K. Lehtinen and M. Zimmermann. Good-for-games ω-pushdown automata. In *Proc. 35th IEEE Symp. on Logic in Computer Science*, 2020.
21 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.
22 G. Morgenstern. Expressiveness results at the bottom of the ω-regular hierarchy. M.Sc. Thesis, The Hebrew University, 2003.
23 D.E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd IEEE Symp. on Logic in Computer Science*, pages 422–427, 1988.
24 D. Niwinski and I. Walukiewicz. Relating hierarchies of word and tree automata. In *Proc. 15th Symp. on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*. Springer, 1998.
25 M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.
26 S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.
27 A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.
28 A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
29 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

A Determinization of a SD-NBW

Given an SD-NBW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, the DBW generated in [14] is $\mathcal{A}' = \langle \Sigma, Q', q_0', \delta', \alpha' \rangle$, where $Q' = 2^Q$, $q_0' = \{q_0\}$, $\alpha' = \{S \in 2^Q : S \subseteq \alpha\}$, and the transition function δ' is defined for every subset $S \in 2^Q$ and letter $\sigma \in \Sigma$ as follows. If $\delta(S, \sigma) \cap \alpha = \emptyset$, then $\delta'(S, \sigma) = \delta(S, \sigma)$. Otherwise, if $\delta(S, \sigma) \cap \alpha \neq \emptyset$, then $\delta'(S, \sigma) = \delta(S, \sigma) \cap \alpha$.

Thus, we proceed as the standard subset construction, except that whenever a constructed set contains a state in α, we leave in the set only states in α. Accordingly, every reachable state $S \in Q'$ contains only α-states of $\mathcal{A}$ or only $\bar{\alpha}$-states of $\mathcal{A}$. Note that as $\mathcal{A}$ is SD, then for every two states $q, q' \in Q$, letter $\sigma \in \Sigma$, and transitions $\langle q, \sigma, s \rangle, \langle q', \sigma, s' \rangle \in \Delta$, if $q \sim_{\mathcal{A}} q'$, then $s \sim_{\mathcal{A}} s'$. Consequently, every reachable state S of $\mathcal{A}'$ consists of $\mathcal{A}$-equivalent states. Without loss of generality, we restrict $\mathcal{A}'$ to its reachable states.

The following two propositions follow immediately from the definitions:

▶ **Proposition 14.** *Consider states $q \in Q$ and $S \in Q'$, a letter $\sigma \in \Sigma$, and transitions $\langle q, \sigma, q' \rangle$ and $\langle S, \sigma, S' \rangle$ of $\mathcal{A}$ and $\mathcal{A}'$, respectively. If q is $\mathcal{A}$-equivalent to the states in S, then q' is $\mathcal{A}$-equivalent to the states in S'.*

▶ **Proposition 15.** *Consider a state S of $\mathcal{A}'$ and a letter $\sigma \in \Sigma$. If $\langle S, \sigma, S' \rangle \in \Delta'$ and $S' \notin \alpha'$, then all the σ-successors of a state $s \in S$ are in $S' \setminus \alpha$.*

We can now prove the correctness of the construction:

▶ **Proposition 16.** *The automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.*

Proof. We first prove that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. Let $r_{\mathcal{A}'} = S_0, S_1, S_2, \ldots$ be an accepting run of $\mathcal{A}'$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots$. We construct an accepting run of $\mathcal{A}$ on w. Since $r_{\mathcal{A}'}$ is accepting, there are infinitely many positions $j_1, j_2, \ldots$ with $S_{j_i} \in \alpha'$. We also define $j_0 = 0$. Consider the DAG $G = \langle V, E \rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is the union $\bigcup_{i \geq 0} (S_{j_i} \times \{i\})$.
- $E \subseteq \bigcup_{i \geq 0} (S_{j_i} \times \{i\}) \times (S_{j_{i+1}} \times \{i+1\})$ is such that for all $i \geq 0$, it holds that $E(\langle s', i \rangle, \langle s, i+1 \rangle)$ iff there is a finite run from s' to s over $w[j_i + 1, j_{i+1}]$. Then, we label this edge by the run from s' to s.

By the definition of $\mathcal{A}'$, for every $j \geq 0$ and state $s_{j+1} \in S_{j+1}$, there is a state $s_j \in S_j$ such that $\langle s_j, \sigma_j, s_{j+1} \rangle \in \Delta$. Thus, it follows by induction that for every $i \geq 0$ and state $s_{i+1} \in S_{j_{i+1}}$, there is a state $s_i \in S_{j_i}$ such that there is a finite run from s_i to s_{i+1} on $w[j_i + 1, j_{i+1}]$. Thus, the DAG G has infinitely many reachable vertices from the vertex $\langle q_0, 0 \rangle$. Also, as the nondeterminism degree of $\mathcal{A}$ is finite, so is the branching degree of G. Thus, by König's Lemma, G includes an infinite path, and the labels along the edges of this path define a run of $\mathcal{A}$ on w. Since for all $i \geq 1$, the state S_{j_i} is in α', and so all the states in S_{j_i} are in α, this run is accepting, and we are done.

For the other direction, assume that $w = \sigma_1 \cdot \sigma_2 \cdots \in L(\mathcal{A})$, and let $r = r_0, r_1, \ldots$ be an accepting run of $\mathcal{A}$ on w. Let $S_0, S_1, S_2 \ldots$ be the run of $\mathcal{A}'$ on w, and assume, by way of contradiction, that there is a position $j \geq 0$ such that $S_j, S_{j+1}, \ldots$ is an α-free run on the suffix $w[j + 1, \infty]$. Then, an iterative application of Proposition 15 implies that all the runs of a state $s_j \in S_j$ on $w[j + 1, \infty]$ are α-free in $\mathcal{A}$. Also, an iterative application of Proposition 14 implies that $r_j \sim_{\mathcal{A}} s_j$, and since r is an accepting run of $\mathcal{A}$, it holds that $\mathcal{A}^{s_j}$ has an accepting run on $w[j + 1, \infty]$, and we have reached a contradiction. ◀

It is left to prove that weakness of $\mathcal{A}$ is preserved in $\mathcal{A}'$.

▶ **Proposition 17.** *If $\mathcal{A}$ is an NWW, then $\mathcal{A}'$ is a DWW.*

Proof. Assume by way of contradiction that there are reachable states $S \in \alpha'$ and $S' \notin \alpha'$, and an infinite run $r_{\mathcal{A}'} = S_0, S_1, S_2, \ldots$ that visits both S and S' infinitely often. Recall that a reachable state in Q' contains only α-states of $\mathcal{A}$ or only $\bar{\alpha}$-states of $\mathcal{A}$. Hence, S' contains only $\bar{\alpha}$-states of $\mathcal{A}$.

As in the proof of Proposition 16, the run $r_{\mathcal{A}'}$ induces an infinite run $r_{\mathcal{A}} = s_0, s_1, s_2, \ldots$, where for all positions $j \geq 0$, it holds that $s_j \in S_j$. Since the run $r_{\mathcal{A}'}$ visits S infinitely often, then $r_{\mathcal{A}}$ visits infinitely many α-states. Likewise, since $r_{\mathcal{A}'}$ visits S' infinitely often, then $r_{\mathcal{A}}$ also visits infinitely many $\bar{\alpha}$-states. This contradicts the weakness of $\mathcal{A}$, and we are done. ◀

B Details of the Reduction in Theorem 3

We describe the technical details of the construction of $\mathcal{A}$. Let $T = \langle \Gamma, Q, \rightarrow, q_0, q_{acc}, q_{rej} \rangle$, where Γ is the working alphabet, Q is the set of states, $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the left/right, according to Δ), q_0 is the initial state, q_{acc} is the accepting states, and q_{rej} is the rejecting one. The transitions function $\rightarrow$ is defined also for the final states q_{acc} and q_{rej}: when a computation of T reaches them, it erases the tape, goes to the leftmost cell in the tape, and moves to the initial state q_0. Recall that $s : \mathbb{N} \rightarrow \mathbb{N}$ is the polynomial space function of T. Thus, when T runs on the empty tape, it uses at most $n_0 = s(0)$ cells.

We encode a configuration of T on a word of length at most n_0 by a word of the form $\#\gamma_1\gamma_2 \dots (q, \gamma_i) \dots \gamma_{n_0}$. That is, a configuration starts with $\#$, and all its other letters are in Γ, except for one letter in $Q \times \Gamma$. The meaning of such a configuration is that the j'th cell in T, for $1 \leq j \leq n_0$, is labeled γ_j, the reading head points at cell i, and T is in state q. For example, the initial configuration of T is $\#(q_0, b)b \dots b$ (with $n_0 - 1$ occurrences of b's) where b stands for an empty cell. We can now encode a computation of T by a sequence of configurations.

Let $\Sigma = \{\#\} \cup \Gamma \cup (Q \times \Gamma)$ and let $\#\sigma_1 \dots \sigma_{n_0}\#\sigma'_1 \dots \sigma'_{n_0}$ be two successive configurations of T. We also set σ_0, σ'_0, and σ_{n_0+1} to $\#$. For each triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ with $1 \leq i \leq n_0$, we know, by the transition relation of T, what σ'_i should be. In addition, the letter $\#$ should repeat exactly every $n_0 + 1$ letters. Let $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ denote our expectation for σ'_i. That is,

- $next(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1} \rangle) = next(\langle \#, \gamma_i, \gamma_{i+1} \rangle) = next(\langle \gamma_{i-1}, \gamma_i, \# \rangle) = \gamma_i$.
- $next(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1} \rangle) = next(\langle (q, \gamma_{i-1}), \gamma_i, \# \rangle) =$
  $$\begin{cases} \gamma_i & \text{If } (q, \gamma_{i-1}) \rightarrow (q', \gamma'_{i-1}, L) \\ (q', \gamma_i) & \text{If } (q, \gamma_{i-1}) \rightarrow (q', \gamma'_{i-1}, R) \end{cases}$$
- $next(\langle \gamma_{i-1}, (q, \gamma_i), \gamma_{i+1} \rangle) = next(\langle \#, (q, \gamma_i), \gamma_{i+1} \rangle) =$
  $next(\langle \gamma_{i-1}, (q, \gamma_i), \# \rangle) = \gamma'_i$ where $(q, \gamma_i) \rightarrow (q', \gamma'_i, \Delta)$ [8].
- $next(\langle \gamma_{i-1}, \gamma_i, (q, \gamma_{i+1}) \rangle) = next(\langle \#, \gamma_i, (q, \gamma_{i+1}) \rangle) =$
  $$\begin{cases} \gamma_i & \text{If } (q, \gamma_{i+1}) \rightarrow (q', \gamma'_{i+1}, R) \\ (q', \gamma_i) & \text{If } (q, \gamma_{i+1}) \rightarrow (q', \gamma'_i, L) \end{cases}$$
- $next(\langle \sigma_{n_0}, \#, \sigma'_1 \rangle) = \#$.

Consistency with *next* now gives us a necessary condition for a word to encode a legal computation that uses n_0 tape cells.

In order to accept words that satisfy C1, namely detect a violation of *next*, the NWW $\mathcal{A}$ use its nondeterminism and guesses a triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle \in \Sigma^3$ and guesses a position in the word, where it checks whether the three letters to be read starting this position are σ_{i-1}, σ_i, and σ_{i+1}, and checks whether $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ is not the letter to come $n_0 + 1$ letters later. Once $\mathcal{A}$ sees such a violation, it goes to an accepting sink. If *next* is respected, or if the guessed triple and position is not successful, then $\mathcal{A}$ returns to its initial state. Also, at any point that $\mathcal{A}$ still waits to guess a position of a triple, it can guess to return back to the initial state.

In order to accept words that satisfy C2, namely detect an encoding of the initial configuration of T on the empty tape and a final configuration after it that is accepting, the NWW $\mathcal{A}$ guesses a position where it compares the next $n_0 + 1$ letters with $\#(q_0, b)b \dots b$. If the initial configuration is indeed detected, it waits for letters in $\{q_{acc}, q_{rej}\} \times \Gamma$. If a letter with q_{acc} arrives before a letter with q_{rej}, then $\mathcal{A}$ goes to the accepting sink. Otherwise if a letter with q_{rej} arrives before a letter with q_{acc}, then $\mathcal{A}$ returns back to the initial state. Also, at any point that $\mathcal{A}$ still waits to detect the initial configuration, or when it waits to

[8] We assume that the reading head of T does not "fall" from the right or the left boundaries of the tape. Thus, the case where $(i = 1)$ and $(q, \gamma_i) \rightarrow (q', \gamma'_i, L)$ and the dual case where $(i = n_0)$ and $(q, \gamma_i) \rightarrow (q', \gamma'_i, R)$ are not possible.

see a letter in $\{q_{acc}, q_{rej}\} \times \Gamma$, it can guess to return back to the initial state. Note that we could have added the option to keep on waiting for q_{acc} even if q_{rej} arrives first. Indeed, if w includes the initial configuration and both q_{rej} and q_{acc} afterwards, then there must be a violation of *next*.

C Correctness and full details of the reduction in Theorem 4

We first prove that the GFG strategy g defined in Proposition 6 satisfies two essential properties. Then, in Lemma 20, we show that these properties imply that g is a winning GFG strategy for $\mathcal{A}_\varphi$.

▶ **Lemma 18.** *For all $u \in (X \cdot C)^*$ and $v \in R_{n,m}$, if $g(u) = p$, then there is a prefix $y \in (X \cdot C)^*$ of v such that $g(u \cdot y) = q_0$.*

Proof. Let $j \in [m]$ and $2 \leq k \leq n$, be such that v ends with the word $x_k \cdot c_j \cdot x_{k+1} \cdot c_j \cdots x_n \cdot c_j$, and k is the minimal index that is greater than 1, for which x_k participates in c_j. Since we assume that each of the clauses of φ depend on at least two variables, such $k > 1$ exists. Let $i \in \{0, 1\}$ be minimal with $c_j \in C_k^i$, and let $z \in (X \cdot C)^*$ be a prefix of v such that $v = z \cdot x_k \cdot c_j \cdots x_n \cdot c_j$. If there is a prefix $y \in (X \cdot C)^*$ of z, such that $g(u \cdot y) = q_0$ then we are done. Otherwise, $g(u \cdot z) = p$. By definition of g and the choice of k, we know that $g(u \cdot z \cdot x_k) = q_k^i$, where the assignment $x_k = i$ satisfies c_j. Thus, if we take $y = z \cdot x_k \cdot c_j$, then $g(u \cdot y) = q_0$, and y is a prefix of v. ◀

▶ **Lemma 19.** *For all $u \in (X \cdot C)^*$ and $v \in R_{n,m}$, if $g(u) = q_0$, then there is a prefix $z \in (X \cdot C)^*$ of v such that $g(u \cdot z) = p$.*

Proof. This follows immediately from the fact that $D_{n,m}$ is a DFW that recognizes $R_{n,m}$ and p is the only accepting state of $D_{n,m}$. Thus, we may take z to be the minimal prefix of v that is in $R_{n,m}$. ◀

Recall that a GFG strategy $g : \Sigma^* \to Q$ has to agree with the the transitions of $\mathcal{A}_\varphi$. That is, for all $w \in (X \cdot C)^*$, $x_k \in X$, and $c_j \in C$, it holds that $(g(w), x_k, g(w \cdot x_k))$ and $(g(w \cdot x_k), c_j, g(x \cdot x_k \cdot c_j))$ are in Δ_φ. In addition, if g satisfies the conditions in Lemmas 18 and 19, we say that g *supports a* (p, q_0)*-circle.*

▶ **Lemma 20.** *If $g : \Sigma^* \to Q$ is consistent with Δ_φ and supports a (p, q_0)-circle, then for all words $w \in L_{n,m}$, the run $g(w)$ is accepting.*

Proof. Consider a word $w \in L_{n,m} = (R_{n,m})^\omega$. Observe that if $w' \in (X \cdot C)^\omega$ is a suffix of w, then $w' \in L_{n,m}$, and hence has a prefix in $R_{n,m}$. Thus, if g supports a (p, q_0)-circle, there exist $y_1, z_1 \in (X \cdot C)^*$, such that $y_1 \cdot z_1$ is a prefix of w, $g(y_1) = q_0$, and $g(y_1 \cdot z_1) = p$. Let $w' \in (X \cdot C)^\omega$ be the suffix of w with $w = y_1 \cdot z_1 \cdot w'$. By the above, $w' \in L_{n,m}$, and we can now apply again the assumption on g to obtain $y_2, z_2 \in (X \cdot C)^*$ such that $y_2 \cdot z_2$ is a prefix of w', $g(y_1 \cdot z_1 \cdot y_2) = q_0$, and $g(y_1 \cdot z_1 \cdot y_2 \cdot z_2) = p$. By iteratively applying this argument, we construct $\{y_i, z_i : i \geq 1\} \subseteq (X \cdot C)^*$, such that $w^i = y_1 \cdot z_1 \cdot y_2 \cdot z_2 \cdots y_{i-1} \cdot z_{i-1} \cdot y_i$ is a prefix of w, and $g(w^i) = q_0$, for all $i \geq 1$. We conclude that $q_0 \in \inf(g(w))$, and hence $g(w)$ is accepting. ◀

It is easy to see that there is a correspondence between assignments to the variables in X and deterministic prunnings of $\mathcal{A}_\varphi$. Indeed, a pruning of p amounts to choosing, for each $k \in [n]$, a value $i_k \in \{0, 1\}$: the assignment $x_k = i_k$ corresponds to keeping the transition $\langle p, x_k, q_k^{i_k} \rangle$ and removing the transition $\langle p, x_k, q_k^{\neg i_k} \rangle$. For an assignment $a : X \to \{0, 1\}$, we denote by $\mathcal{A}_\varphi^a$ the deterministic pruning of $\mathcal{A}_\varphi$ that is associated with a. We prove that a satisfies φ iff $\mathcal{A}_\varphi^a$ is equivalent to $\mathcal{A}_\varphi$. Thus, the number of deterministic prunnings of $\mathcal{A}_\varphi$ that result in a DBW equivalent to $\mathcal{A}_\varphi$, equals to the number of assignments that satisfy φ. In particular, φ is satisfiable iff $\mathcal{A}_\varphi$ is DBP. This concludes the proof of the lower bound in Theorem 4.

▶ **Proposition 21.** *For every assignment $a : X \to \{0, 1\}$, we have that $L(\mathcal{A}_\varphi^a) = L(\mathcal{A}_\varphi)$ iff φ is satisfied by a.*

Proof. Assume first that φ is not satisfied by a. We prove that $L_{n,m} \neq L(\mathcal{A}_\varphi^a)$. Let $j \in [m]$ be such that c_j is not satisfied by a. I.e, for all $k \in [n]$ the assignment $x_k = i_k$ does not satisfy c_j. Since q_k^i is reachable in $\mathcal{A}_\varphi^a$ iff $i = i_k$, and all c_j-labeled transitions from $\{q_k^{i_k} : k \in [n]\}$ are to p, it follows that the run of $\mathcal{A}_\varphi^a$ on $\{x_1 \cdot c_j \cdot x_2 \cdot c_j \cdots x_n \cdot c_j\}^\omega$ never visits q_0, and hence is rejecting. Thus, $(x_1 \cdot c_j \cdot x_2 \cdot c_j \cdots x_n \cdot c_j)^\omega \in L_{n,m} \setminus L(\mathcal{A}_\varphi^a)$.

For the other direction, we assume that a satisfies φ and prove that $L(\mathcal{A}_\varphi^a) = L_{n,m}$. Let $g^a : \Sigma^* \to Q$ be the memoryless strategy that correspond to the pruning $\mathcal{A}_\varphi^a$. By Lemma 20, it is sufficient proving that g^a supports a (p, q_0)-circle. Note that every strategy for $L_{n,m}$ satisfies Lemma 19. Indeed, the proof only uses the fact that $D_{n,m}$ is a DFW that recognizes $R_{n,m}$ with a single accepting state p. Thus, we only need to prove that g^a satisfies Lemma 18. That is, for all $u \in (X \cdot C)^*$ and $v \in R_{n,m}$, if $g^a(u) = p$, then there is a prefix $y \in (X \cdot C)^*$ of v, such that $g^a(u \cdot y) = q_0$. Consider such words u and v, and let $j \in [m]$ be such that c_j is the last letter of v. Let $k \in [n]$ be the minimal index for which $c_j \in C_k^{i_k}$, and let $z \in (X \cdot C)^*$ be a prefix of v such that $v = z \cdot x_k \cdot c_j \cdot x_{k+1} \cdot c_j \cdots x_n \cdot c_j$. If there exists a prefix $y \in (X \cdot C)^*$ of z such that $g^a(u \cdot y) = q_0$, then we are done. Otherwise, the finite run of $\mathcal{A}_\varphi^a$ on z from p, returns back to p, and hence $g^a(u \cdot z) = p$. Now $g^a(u \cdot z \cdot x_k) = q_k^{i_k}$, and since $x_k = i_k$ satisfies c_j we have $g^a(u \cdot z \cdot x_k \cdot c_j) = q_0$. Thus, we may take $y = z \cdot x_k \cdot c_j$ which is a prefix of v, and we are done. ◀