HAL
open science

Width of Non-deterministic Automata

Denis Kuperberg, Anirban Majumdar

To cite this version:

Denis Kuperberg, Anirban Majumdar. Width of Non-deterministic Automata. 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018), 2018, Caen, France. pp.48 - 49, 10.4230/LIPIcs.STACS.2018.48 . hal-01891180

HAL Id: hal-01891180

https://hal.science/hal-01891180v1

Submitted on 9 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Width of Non-deterministic Automata*

Denis Kuperberg[1] and Anirban Majumdar[2]

1 CNRS, ENS de Lyon, LIP
2 Chennai Mathematical Institute

Abstract

We introduce a measure called width, quantifying the amount of nondeterminism in automata. Width generalises the notion of good-for-games (GFG) automata, that correspond to NFAs of width 1, and where an accepting run can be built on-the-fly on any accepted input. We describe an incremental determinisation construction on NFAs, which can be more efficient than the full powerset determinisation, depending on the width of the input NFA. This construction can be generalised to infinite words, and is particularly well-suited to coBüchi automata in this context. For coBüchi automata, this procedure can be used to compute either a deterministic automaton or a GFG one, and it is algorithmically more efficient in this last case. We show this fact by proving that checking whether a coBüchi automaton is determinisable by pruning is NP-complete. On finite or infinite words, we show that computing the width of an automaton is PSPACE-hard.

1998 ACM Subject Classification F.1.1 Models of computations, F.4.3 Formal Languages

Keywords and phrases width, non-deterministic automata, determinisation, good-for-games, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.48

1 Introduction

Determinisation of non-deterministic automata (NFAs) is one of the cornerstone problems of automata theory, with countless applications in verification. There is a very active field of research for optimizing or approximating determinisation, or circumventing it in contexts like inclusion of NFA or Church Synthesis. Indeed, determinisation is a costly operation, as the state space blow-up is in $O(2^n)$ on finite words, $O(3^n)$ for coBüchi automata [16], and $2^{O(n\log(n))}$ for Büchi automata [17].

If $\mathcal{A}$ and $\mathcal{B}$ are NFAs, the classical way of checking the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is to determinise $\mathcal{B}$, complement it, and test emptiness of $L(A) \cap \overline{L(B)}$. To circumvent a full determinisation, the recent algorithm from [3] proved to be very efficient, as it is likely to explore only a part of the powerset construction. Other approaches use simulation games to approximate inclusion at a cheaper cost, see for instance [8].

Another approach consists in replacing determinism by a weaker constraint that suffices in some particular context. In this spirit, Good-for-Games automata (GFG for short) were introduced in [9], as a way to solve the Church synthesis problem. This problem asks, given a specification L, typically given by an LTL formula, over an alphabet of inputs and outputs, whether there is a reactive system (transducer) whose behaviour is included in L. The classical solution computes a deterministic automaton for L, and solves a game defined on this automaton. It turns out that replacing determinism by the weaker constraint of being GFG is sufficient in this context. Intuitively, GFG automata are non-deterministic

* This work was supported by the grant PALSE Impulsion.

© Denis Kuperberg and Anirban Majumdar;
licensed under Creative Commons License CC-BY
35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 48; pp. 48:1–48:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automata where it is possible to build an accepting run in an online way, without knowledge of the future, provided the input word is in the language of the automaton. In [9], it is shown that GFG automata allow an incremental algorithm for the Church synthesis problem: we can build increasingly large games, with the possibility that the algorithm stops before the full determinisation is needed. One of the aims of this paper is to generalise this idea to determinisation of NFA, for use in any context and not only Church synthesis. We give an incremental determinisation construction, where the emphasis is on space-saving, and that allows in some cases to avoid building the full powerset construction.

The notion of width introduced in this paper generalises the GFG model, by allowing more than one run to be built in an online way. Intuitively, width quantifies how many states we have to keep track of simultaneously in order to build an accepting run in an online way. The maximal width of an automaton is its number of states. The width of an automaton corresponds to the number of steps performed by our incremental determinisation construction before stopping. In the worst case where the width is equal to the number of states of the automaton, we end up performing the full powerset construction (or its generalisations for infinite words). We study here the complexity of directly computing the width of a nondeterministic automaton, and we show that it is PSPACE-hard and in EXPTIME.

The properties of GFG automata and links with other models (tree automata, Markov Decision Processes) are studied in [2, 10, 11]. Colcombet introduced a generalisation of the concept of GFG called history-determinism [5], replacing determinism for automata with counters. It was conjectured by Colcombet [6] that GFG automata were essentially deterministic automata with additional useless transitions. It was shown in [11] that on the contrary there is in general an exponential state space blowup to translate GFG automata to deterministic ones. GFG automata retain several good properties of determinism, in particular they can be composed with trees and games, and easily checked for inclusion.

We give here the first algorithms allowing to build GFG automata from arbitrary non-deterministic automata on infinite words, allowing to potentially save exponential space compared to deterministic automata. Our incremental constructions look for small GFG automata, and aim at avoiding the worst-case complexities of determinisation constructions. Moreover, in the case of coBüchi automata, we show that the procedure is more efficient than its analog looking for a deterministic automaton, since checking for GFGness is polynomial [11], while we show here that the corresponding step for determinisation, that is checking whether a coBüchi automaton is Determinisable By Pruning (DBP) is NP-complete.

As a measure of non-determinism, width can be compared with ambiguity, where the idea is to limit the number of possible runs of the automaton. In this context unambiguous automata play a role analogous to GFG automata for width. Unambiguous automata are studied in [12], degrees of ambiguity are investigated in [18, 13, 14]. In the online long version of the paper, we give examples of automata with various width and ambiguity, showing that these two measures are essentially orthogonal.

We start by describing the width approach on finite words, and then move to infinite words, focusing mainly on the coBüchi acceptance condition. We end by briefly describing the picture for Büchi automata.

2 Definitions

We will use Σ to denote a finite alphabet. The empty word is denoted ε. If $i \leq j$, the set $\{i, i+1, i+2, \ldots, j\}$ is denoted $[i, j]$. If X is a set and $k \in \mathbb{N}$, we note $X^{\leq k}$ for $\bigcup_{i=0}^{k} X^i$. The complement of a set X is denoted $\overline{X}$. If $u \in \Sigma^*$ is a word and $L \subseteq \Sigma^*$ is a language, the left quotient of L by u is $u^{-1}L := \{v \in \Sigma^* \mid uv \in L\}$.

2.1 Automata

A non-deterministic automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, q_0, \Delta, F)$ where Q is the set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

The transition function is naturally generalised to 2^Q by setting for any $(X, a) \in 2^Q \times \Sigma$ $\Delta(X, a)$ the set of a-successors of X, i.e. $\Delta(X, a) = \{q \in Q \mid \exists p \in X, q \in \Delta(p, a)\}$.

If for all $(p, a) \in Q \times \Sigma$ there is a unique $q \in Q$ such that $(p, a, q) \in \Delta$, we say that $\mathcal{A}$ is *deterministic*.

If $u = a_1 \ldots a_n$ is a finite word of Σ^*, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 \ldots q_n$ such that for all $i \in [1, n]$, we have $q_i \in \Delta(q_{i-1}, a_i)$. The run is said to be *accepting* if $q_n \in F$.

If $u = a_1 a_2 \ldots$ is an infinite word of Σ^ω, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 q_2 \ldots$ such that for all $i > 0$, we have $q_i \in \Delta(q_{i-1}, a_i)$. A run is said to be *Büchi accepting* if it contains infinitely many accepting states, and *coBüchi accepting* if it contains finitely many non-accepting states. Automata on infinite words will be called Büchi and coBüchi automata, to specify their acceptance condition.

We will note NFA (resp. DFA) for a non-deterministic (resp. deterministic) automaton on finite words, NBW (resp. DBW) for a non-deterministic (resp. deterministic) Büchi automaton, and NCW (resp. DCW) for a non-deterministic (resp. deterministic) coBüchi automaton.

We also mention the *parity condition* on infinite words: each state q has a rank $\mathrm{rk}(q) \in \mathbb{N}$, and an infinite run is accepting if the highest rank appearing infinitely often is even.

The language of an automaton $\mathcal{A}$, noted $L(\mathcal{A})$, is the set of words on which the automaton $\mathcal{A}$ has an accepting run. Two automata are said equivalent if they recognise the same language.

An automaton $\mathcal{A}$ is *determinisable by pruning* (DBP) if an equivalent deterministic automaton can be obtained from $\mathcal{A}$ by removing some transitions.

An automaton $\mathcal{A}$ is *Good-For-Games* (GFG) if there exists a function $\sigma \colon A^* \to Q$ (called *GFG strategy*) that resolves the non-determinism of $\mathcal{A}$ depending only on the prefix of the input word read so far: over every word $u = a_1 a_2 a_3 \ldots$ (finite or infinite depending on the type of automaton considered), the sequence of states $\sigma(\varepsilon)\sigma(a_1)\sigma(a_1 a_2)\sigma(a_1 a_2 a_3)\ldots$ is a run of $\mathcal{A}$ on u, and it is accepting whenever $u \in L(\mathcal{A})$. For instance every DBP automaton is GFG. See [2] for more introductory material and examples on GFG automata.

2.2 Games

A *game* $\mathcal{G} = (V_0, V_1, v_I, E, W_0)$ of infinite duration between two players 0 and 1 consists of: a finite set of *positions* V being a disjoint union of V_0 and V_1; an *initial position* $v_I \in V$; a set of *edges* $E \subseteq V \times V$; and a *winning condition* $W_0 \subseteq V^\omega$.

A *play* is an infinite sequence of positions $v_0 v_1 v_2 \cdots \in V^\omega$ such that $v_0 = v_I$ and for all $n \in \mathbb{N}$, $(v_n, v_{n+1}) \in E$. A play $\pi \in V^\omega$ is *winning* for Player 0 if it belongs to W_0. Otherwise π is *winning* for Player 1.

A *strategy* for Player 0 (resp. 1) is a function $\sigma_0 \colon V^* \times V_0 \to V$ (resp. $\sigma_1 \colon V^* \times V_1 \to V$), describing which edge should be played given the history of the play $u \in V^*$ and the current position $v \in V$. A strategy has to obey the edge relation, i.e. there has to be an edge in E from v to $\sigma_P(u, v)$. A play π is *consistent* with a strategy σ_P of a player P if for every n such that $\pi(n) \in V_P$ we have $\pi(n+1) = \sigma_P(v_0 \dots v_{n-1}, v_n)$.

A strategy for Player 0 (resp. Player 1) is *positional* if it does not use the history of the play, i.e. it is a function $V_0 \to V$ (resp. $V_1 \to V$).

We say that a strategy σ_P of a player P is *winning* if every play consistent with σ_P is winning for P. In this case, we say that P *wins* the game $\mathcal{G}$.

A game is *positionally determined* if exactly one of the players has a positional winning strategy in the game.

3 Finite words

3.1 Width of a NFA

Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ be a NFA, and $n = |Q|$ be the size of $\mathcal{A}$.

We want to define the *width* of a $\mathcal{A}$ as the minimum number of simultaneous states that need to be tracked in order to be able to deterministically build an accepting run in an online way.

In order to define this notion formally, we introduce a family of games $\mathcal{G}_w(\mathcal{A}, k)$, parameterized by an integer $k \in [1, n]$.

The game $\mathcal{G}_w(\mathcal{A}, k)$ is played on $Q^{\leq k}$, starts in $X_0 = \{q_0\}$, and the round i of the game from a position $X_i \in Q^{\leq k}$ is defined as follows:

- Player 1 chooses a letter $a_{i+1} \in \Sigma$.
- Player 0 moves to a subset $X_{i+1} \subseteq \Delta(X_i, a_{i+1})$ of size at most k.

A play is winning for Player 0 if for all $r \in \mathbb{N}$, whenever $a_1 a_2 \dots a_r \in L(\mathcal{A})$, X_r contains an accepting state.

▶ **Definition 1.** The width of a NFA $\mathcal{A}$, denoted $\mathrm{width}(\mathcal{A})$, is the least k such that Player 0 wins $\mathcal{G}_w(\mathcal{A}, k)$.

Intuitively, the width measures the "amount of non-determinism" in an automaton: it counts the number of simultaneous states we have to keep track of, in order to be sure to find an accepting run in an online way.

▶ Fact 1. A NFA $\mathcal{A}$ is GFG if and only if $\mathrm{width}(\mathcal{A}) = 1$.

3.2 Partial powerset construction

We give here a generalisation of the powerset construction, following the intuition of the width measure.

We define the k-subset construction of $\mathcal{A}$ to be the subset construction where the size of each set is bounded by k. Formally, it is the NFA $\mathcal{A}_k = (Q^{\leq k}, \Sigma, \{q_0\}, \Delta', F')$ where:

- $\Delta'(X, a) := \begin{cases} \{\Delta(X, a)\} & \text{if } |\Delta(X, a)| \leq k \\ \{X' \mid X' \subseteq \Delta(X, a), |X'| = k\} & \text{otherwise} \end{cases}$
- $F' := \{X \in Q^{\leq k} \mid X \cap F \neq \emptyset\}$

▶ **Lemma 2.** *$\mathcal{A}_k$ has less than $\dfrac{n^k}{(k-1)!} + 1$ states.*

Proof. The number of states of $\mathcal{A}_k$ is (at most) $|Q^{\leq k}| = \sum_{i=0}^{k} \binom{n}{i}$. Using the fact that $\binom{n}{i} \leq \frac{n^i}{i!}$, we can bound the number of states of $\mathcal{A}_k$ by $\sum_{i=0}^{k} \frac{n^i}{i!} \leq \sum_{i=0}^{k} \frac{n^k}{k!} \leq 1 + \sum_{i=1}^{k} \frac{n^k}{k!} = \frac{n^k}{(k-1)!} + 1$. ◀

The following lemma shows the link between width and the k-powerset construction.

▶ **Lemma 3.** $\mathrm{width}(\mathcal{A}) \leq k$ *if and only if* $\mathcal{A}_k$ *is GFG.*

Proof. Winning strategies in $\mathcal{G}_w(\mathcal{A}, k)$ are in bijection with GFG strategies for $\mathcal{A}_k$. ◀

3.3 GFG automata on finite words

We recall here results on GFG automata on finite words.

We start with a Lemma characterizing GFG strategies. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ be a NFA recognising a language L, and $\sigma : \Sigma^* \to Q$ be a potential GFG strategy. If $q \in Q$, we denote $L(q)$ the language accepted from q in $\mathcal{A}$, i.e. $L(q)$ is the language of $\mathcal{A}$ with q as initial state.

▶ **Lemma 4.** *σ is a GFG strategy if and only if for all $u \in \Sigma^*$, $L(\sigma(u)) = u^{-1}L$*

We now go to the main result of this section. This result has first been proved in [1], and then a more general version allowing lookahead was proved using a game-based approach in [15].

▶ **Theorem 5.** *[1, 15] A NFA $\mathcal{A}$ is GFG if and only if it is DBP. Moreover, it is in $O(n^2)$ to determine whether a NFA of size n is GFG, and to compute an equivalent DFA by removing transitions.*

3.4 Incremental determinisation procedure

We can now describe an incremental determinisation procedure, aiming at saving resources in the search of a deterministic automaton. In the process, we also compute the width of the input NFA.

The algorithm goes as follows:

```
k = 0
Repeat
    k := k + 1
    Construct A_k
Until A_k is GFG
Compute an equivalent DFA D from A_k by removing transitions
Return D, k
```

The usual determinisation procedure uses the full powerset construction, i.e. assumes that we are in the case of maximal width. In a second step, the deterministic automaton can be minimized easily.

Our method here is to approach this construction "from below", and incrementally increase the width until we find the good one. In some cases, this allows to compute directly a smaller automaton, and avoiding using the full powerset construction of exponential state complexity.

For a NFA with n states and width k, the complexity of this algorithm is in $O\left(\frac{n^{2k}}{(k-1)!^2}\right)$, by Lemma 2 and Theorem 5.

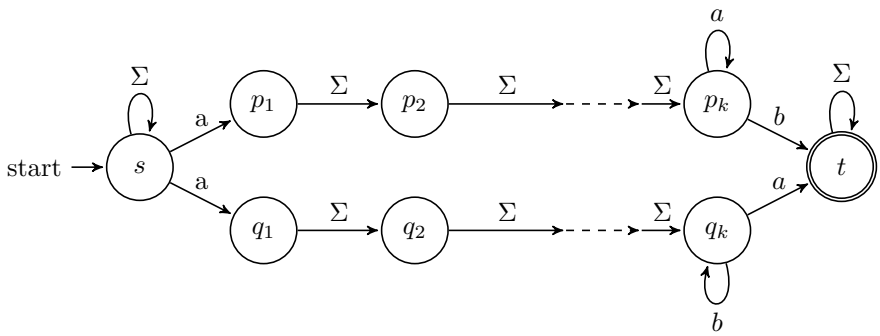

Figure 1 Example: 2-subset construction is enough

▶ **Example 6.** Here the language recognised by this automaton is $L(\mathcal{A}) = \Sigma^* a \Sigma^{\geq k}$, and it has width 2. Therefore, our determinisation procedure uses time $O(n^4)$ and directly builds a DFA of size $O(n^2)$, while a classical determinisation via powerset construction would build an exponential-size DFA.

But in some other cases, the powerset construction is actually more efficient than the k-powerset construction, in terms of number of reachable states. It would therefore be interesting to be able to either run the two methods in parallel, or guess which one is more efficent based on the shape of the input NFA.

3.5 Complexity results on the width problem

In this section, we study the complexity of the *width problem*: given a NFA $\mathcal{A}$ and an integer k, is it true that $\text{width}(\mathcal{A}) \leq k$?

Being able to solve this problem efficiently would allow us to optimize the incremental determinisation algorithm, by aiming at the optimal k matching the width right away instead of trying the different width candidates incrementally.

▶ **Theorem 7.** *The width problem is PSPACE-hard.*

Proof. We prove this by reduction from the universality of NFA Problem (i.e. does an input NFA accept all words?) which is known to be PSPACE-Complete.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be a NFA that we want to check for universality. Let $n = |Q|$.

Let a be a letter in Σ and $\#$ be a new letter not in Σ.

We build a NFA $\mathcal{B}$ over $\Sigma' = \Sigma \cup \{\#\}$ as the union of two NFAs $\mathcal{B}_1$ and $\mathcal{B}_2$ as shown in Figure 2.

Formally $\mathcal{B}_1 = (Q_1, \Sigma', q_0, \Delta_1, F_1)$, with $Q_1 = Q \cup \{q_\#\}$, $F_1 = \{q_\#\}$, and

$$\Delta_1(p, x) = \begin{cases} \Delta(p, x) & \text{if } p \in Q \text{ and } x \neq \# \\ \{q_\#\} & \text{if } (p, x) \in F \times \{\#\} \text{ or if } (p, x) = (q_\#, a) \\ \emptyset & \text{otherwise.} \end{cases}$$

Its language is $L(\mathcal{B}_1) = L(\mathcal{A})\#a^*$.

The NFA $\mathcal{B}_2 = (Q_2, \Sigma', p_I, \Delta_2, \{p_0\})$ has $n + 2$ states as described on the picture, and recognises the language $L(\mathcal{B}_2) = \Sigma^* \# a^{\geq n}$.

We define $\mathcal{B} = (Q_\mathcal{B}, \Sigma', \{q_0, p_I\}, \Delta_\mathcal{B}, F_\mathcal{B})$ as the union of $\mathcal{B}_1$ and $\mathcal{B}_2$. We allow here multiple initial states for simplicity, but it is straigthforward to adapt the construction in order to have a unique initial state.

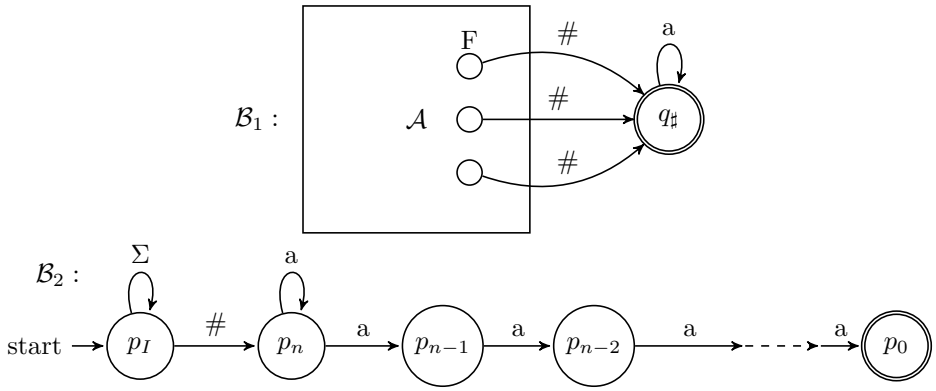

Figure 2 Automaton $\mathcal{B}$

The intuition here is that if $\mathcal{A}$ is universal, then $\mathcal{B}_2$ is useless in $\mathcal{B}$ as $L(\mathcal{B}) = L(\mathcal{B}_1) = \Sigma^*\#a^*$, and $\text{width}(\mathcal{B}) = \text{width}(\mathcal{A}) \leq n$. However, if $\mathcal{A}$ is not universal, then $\mathcal{B}$ is forced to use its $\mathcal{B}_2$ component, inducing a width at least $n+1$. This is formalized in the following lemma.

▶ **Lemma 8.** $\text{width}(\mathcal{B}) \leq n \iff L(\mathcal{A}) = \Sigma^*$.

Proof. ($\Rightarrow$) Suppose $\text{width}(\mathcal{B}) \leq n$ but $\exists u \in \Sigma^* \setminus L(\mathcal{A})$. Let $\mathcal{C}_n$ be the n-subset construction of $\mathcal{B}$. By Lemma 3, $\mathcal{C}_n$ is GFG, let $\sigma : \Sigma^* \to (Q_\mathcal{B})^{\leq n}$ be a GFG strategy of $\mathcal{C}_n$.

Consider the word $w = u\#a^n$. Note that $w \in L(\mathcal{B}_2) \setminus L(\mathcal{B}_1)$. Let $X \in (Q_\mathcal{B})^{\leq n}$ be the subset reached by σ on w, i.e. $X = \sigma(w)$. Notice that since $u \notin L(\mathcal{A})$, we have $X \subseteq (Q_2 \setminus \{p_I\})$, i.e. $X \subseteq \{p_0, p_1, \ldots, p_n\}$. Since $|X| \leq n$, there is $i \in [0, n]$ such that $p_i \notin X = \sigma(w)$. This means that $p_0 \notin \sigma(wa^i)$, hence $\sigma(wa^i)$ is not accepting. But this word is in $L(\mathcal{B})$ (as it is in $L(\mathcal{B}_2)$), this contradicts the fact that σ is a GFG strategy. Therefore it must be the case that $L(\mathcal{A}) = \Sigma^*$.

($\Leftarrow$) We now assume that $L(\mathcal{A}) = \Sigma^*$. A GFG strategy in $\mathcal{C}_n$ is given by following the powerset construction in $\mathcal{B}_1$, and ignoring $\mathcal{B}_2$. This shows that $\text{width}(\mathcal{B}) \leq n$. ◀

This constitutes a polynomial reduction from universality to the width problem, so the width problem is PSPACE-hard. Actually, we even showed that the particular case of checking n-width of an automaton of size $2n+3$ is PSPACE-hard. ◀

▶ **Theorem 9.** *The width problem is in EXPTIME.*

Proof. To show the EXPTIME upper bound, it suffices to build the game $\mathcal{G}_w(\mathcal{A}, k)$ of exponential size. Solving such a game is polynomial in the size of the game, so this algorithm runs in exponential time. Also note that the algorithm given in section 3.4 computes the width of a NFA in EXPTIME. ◀

We currently do not know if the width problem is complete for PSPACE or EXPTIME, and we leave this problem open.

▶ Remark. Although the present section deals with finite words, all results are immediately transferable to safety and reachability automata on infinite words. These automata have special acceptance conditions, which are particular cases of both Büchi and coBüchi conditions. Any infinite run is accepting in a safety automaton, and a run is accepting in a reachability automaton if it contains an accepting state. These dual acceptance conditions are of particular interest in verification, as they describe very natural properties.

4 CoBüchi Automata

We now turn to the case of coBüchi automata, and their determinisation problem. Here, since GFG and DBP are no longer equivalent [2, 11], we will also be interested in building GFG automata. As we will see, coBüchi automata are particularly well-suited for this approach for several reasons.

First of all, we recall that NCW and DCW have same expressive power, i.e. the determinisation of coBüchi automata does not need to introduce more complex acceptance conditions.

4.1 Width of ω-automata

We define here the width of automata on infinite words in a general way, as the definition is independent of the accepting condition.

Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \alpha)$ be an automaton on infinite words with acceptance condition α, and $n = |Q|$ be the size of $\mathcal{A}$.

As before, we want to define the *width* of a $\mathcal{A}$ as the minimum number of states that need to be tracked in order to deterministically build an accepting run in an online way.

We will use the same family of games $\mathcal{G}_w(\mathcal{A}, k)$ as in Section 3.1, they will only differ in the winning condition.

The game $\mathcal{G}_w(\mathcal{A}, k)$ is played on $Q^{\leq k}$, starts in $X_0 = \{q_0\}$, and the round i of the game from a position $X_i \in Q^{\leq k}$ is defined as follows:

- Player 1 chooses a letter $a_{i+1} \in \Sigma$.
- Player 0 moves to a subset $X_{i+1} \subseteq \Delta(X_i, a_{i+1})$ of size at most k.

An infinite play is winning for Player 0 if whenever $a_1 a_2 \cdots \in L(\mathcal{A})$, the sequence $X_0 X_1 X_2 \ldots$ contains an accepting run. That is to say there is a valid accepting run $q_0 q_1 q_2 \ldots$ of $\mathcal{A}$ on $a_1 a_2 \ldots$ such that for all $i \in \mathbb{N}$, $q_i \in X_i$.

▶ **Definition 10.** The width of $\mathcal{A}$, denoted $\text{width}(\mathcal{A})$, is the least k such that Player 0 wins $\mathcal{G}_w(\mathcal{A}, k)$.

As before, an automaton $\mathcal{A}$ is GFG if and only if $\text{width}(\mathcal{A}) = 1$.

4.2 GFG coBüchi automata

We recall here some results from [11] on GFG coBüchi automata.

The first result is the exponential succinctness of coBüchi GFG automata compared to deterministic ones.

▶ **Theorem 11** ([11]). *There is a family of languages $(L_n)_{n\in\mathbb{N}}$ such that for all n, L_n is accepted by a coBüchi GFG automaton of size n, but any deterministic parity automaton for L_n must have size in $\Omega\left(\frac{2^n}{n}\right)$.*

Despite this apparent complexity of GFG NCW, the next theorem shows that they can be recognised efficiently.

▶ **Theorem 12** ([11]). *Given a NCW $\mathcal{A}$, it is in PTIME to decide whether $\mathcal{A}$ is GFG.*

The conjunction of these results make the coBüchi class particularly interesting in our setting: the succinctness allows us to potentially save a lot of space compared to classical determinisation, and Theorem 12 can be used to stop the incremental construction. This is

in the context where we aim at building a GFG automaton, for instance in a context where we want to test for inclusion, or compose it with a game.

We examine later the case where GFG automata are not enough and we are aiming at building a DCW instead.

4.3 Partial breakpoint construction

We generalize here the breakpoint construction from [16], in the same spirit as Section 3.2.

For a parameter k, we want the k-breakpoint construction to be able to keep track of at most k states simultaneously.

Given a NCW $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, we define the k-breakpoint construction of $\mathcal{A}$ as the NCW $\mathcal{A}_k = (Q', \Sigma, \Delta', (\{q_0\}, \{q_0\}), F')$, with

$Q' = \{(X, Y) | X, Y \in Q^{\leq k} \text{ and } Y \subseteq X\}$,

$$\Delta'((X,Y),a) := \begin{cases} \{(\Delta(X,a), \Delta(X,a))\} & \text{if } Y = \emptyset \text{ and } |\Delta(X,a)| \leq k \\ \{(X', X') |\ X' \subseteq \Delta(X,a), |X'| = k\} & \text{if } Y = \emptyset \text{ and } |\Delta(X,a)| > k \\ \{(\Delta(X,a), \Delta(Y,a) \cap F)\} & \text{if } Y \neq \emptyset \text{ and } |\Delta(X,a)| \leq k \\ \{(X', X' \cap (\Delta(Y,a) \cap F)) \mid X' \subseteq \Delta(X,a), |X'| = k\} & \text{otherwise} \end{cases}$$

$F' := \{(X, Y) \in Q' \mid Y \neq \emptyset\}$

That is, a run is accepting in $\mathcal{A}_k$ if it visits the states of the form $(X, \emptyset)$ finitely many times.

▶ **Lemma 13.** *The number of states of $\mathcal{A}_k$ is at most $\sum_{i=0}^{k} \binom{n}{i} 2^i$, which is in $O(\frac{(2n)^k}{k!})$.*

Proof. A state of $\mathcal{A}_k$ is of the form (X, Y) with $|X| \leq k$ and $Y \subseteq X$. Therefore, there are at most $\sum_{i=0}^{k} \binom{n}{i} 2^i$ such states. Since $\binom{n}{i} \leq \frac{n^i}{i!}$, we can bound the number of states by $\sum_{i=0}^{k} \frac{n^k}{k!} 2^i \leq \frac{n^k}{k!} 2^{k+1} = O(\frac{(2n)^k}{k!})$ ◀

▶ **Lemma 14.** *$L(A) = L(\mathcal{A}_k)$, and $\text{width}(\mathcal{A}) \leq k \iff \mathcal{A}_k$ is GFG.*

Proof. This amounts to verifying that the automaton $\mathcal{A}_k$ faithfully simulates the winning condition of $\mathcal{G}_w(\mathcal{A}, k)$. The proof naturally follows from the correctness proof of the breakpoint construction.

◀

4.4 Incremental construction of GFG NCW

Supppose we are given a NCW $\mathcal{A}$, and we want to build an equivalent GFG automaton.

We can do the same as in Section 3.4: incrementally increase k and test for GFGness of $\mathcal{A}_k$, which is in PTIME by Theorem 12. However in the coBüchi setting, the GFG automaton is not necessarily DBP, and can actually be more succinct than any deterministic automaton for the language (Theorem 11).

If we are in a context where we are satisfied with a GFG automaton, such as synthesis or inclusion testing, this procedure can provide us one much more efficiently than determinisation.

Indeed, the example from [11] showing that GFG NCW are exponentially succinct compared to deterministic automata can be easily generalized to any width. For instance if our procedure is applied to the product of this automaton from [11] with the one from Example 6, our construction will stop at the second step and generate a GFG automaton of quadratic

size. This shows that the incremental construction for finding an equivalent GFG NCW can be very efficient compared to determinisation.

Directly computing the width of a NCW is PSPACE-hard and in EXPTIME, by the same arguments as in Section 3.1.

4.5 Aiming for determinism

In cases where a GFG automaton is not enough, and we want instead to build a DCW, we can test for DBPness instead of GFGness in the incremental algorithm. If we find the automaton is DBP, we can remove the useless transitions, and obtain an equivalent DCW.

Notice that the number of steps in this procedure corresponds to an alternative notion of width that can be called *det-width*. The det-width of an automaton $\mathcal{A}$ is the least k such that Player 0 has a positional winning strategy in $\mathcal{G}_w(\mathcal{A}, k)$. Det-width always matches width on finite words by Theorem 5, but the notions diverge on infinite words.

This section studies the complexity of checking DBPness for NCW. The next theorem shows that surprisingly, DBPness is harder to check than GFGness on NCW.

▶ **Theorem 15.** *Given a NCW $\mathcal{A}$, it is NP-complete to check whether it is DBP.*

We first show the hardness with the following lemma.

▶ **Lemma 16.** *Checking whether a NCW is DBP is NP-hard.*

Proof. We prove this by reduction from the Hamiltonian Cycle problem on a directed graph, which is known to be NP-complete.

Recall that a Hamiltonian cycle is a cycle using each vertex of the graph exactly once.

Suppose, we have a directed graph $G = ([1, n], E)$ and we want to check whether it contains a Hamiltonian cycle. W.l.o.g. we can assume that the graph is strongly connected, otherwise the answer is trivially no.

We construct a NCW $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, where F is the set of accepting states, such that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle. The components of $\mathcal{A}$ are defined as follows: $Q := \bigcup_{i\in[1,n]}\{p_i, q_i, r_i\}$, $\Sigma := \{a_1, a_2, \cdots, a_n, \#\}$, $q_0 := p_1$, $F := \bigcup_{i\in[1,n]}\{p_i, q_i\}$, and finally Δ contains the following transitions, for all $i \in [1, n]$:

$$p_i \xrightarrow{a_i} q_i, \quad p_i \xrightarrow{a_j} r_i \text{ for all } j \neq i, \quad q_i \xrightarrow{\#} p_i, \quad \text{and } r_i \xrightarrow{\#} p_k \text{ if } (i, k) \in E \ .$$

The only non-determinism in $\mathcal{A}$ occurs at the r_i states when reading #: we then have a choice between all the p_k where $(i, k) \in E$.

We give an example for G in figure 3, where solid lines show the Hamiltonian cycle, and the construction of $\mathcal{A}$ from G in figure 4, where solid lines show a determinisation by pruning witnessing this Hamiltonian cycle.

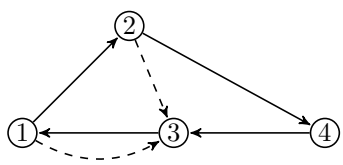

Figure 3 An instance of G

For each $i \in [1, n]$, we can think of the set of states $\{p_i, q_i, r_i\}$ as a cloud in $\mathcal{A}$ representing the vertex i of the graph G.

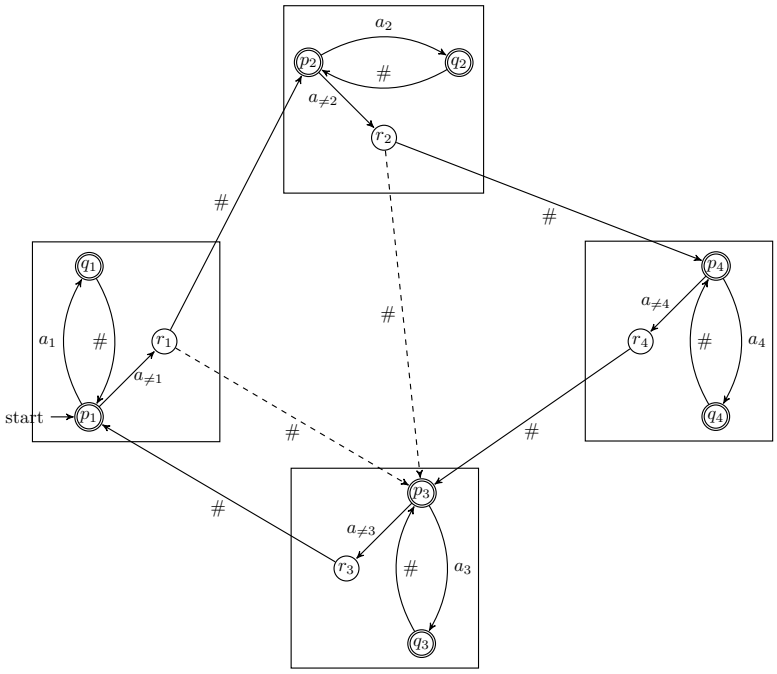

Figure 4 Construction of NCW $\mathcal{A}$ from G of figure 3

Let $\Sigma' := \Sigma \setminus \{\#\}$, and $L = \bigcup_{i=1}^{n} (\Sigma'\#)^*(a_i\#)^\omega$. First note that, provided G is strongly connected, we have $L(\mathcal{A}) = L$. Indeed, for a run to be accepting by $\mathcal{A}$, it has to visit r_i finitely many times for all i, i.e. after some point it has to loop between p_i and q_i for some fixed i, so the input word must be in L. This shows $L(\mathcal{A}) \subseteq L$. On the other hand, consider a word $w \in L$ of the form $u(a_i\#)^\omega$ with $u \in (\Sigma'\#)^*$. Then $\mathcal{A}$ will have a run on u reaching some cloud j, and since the graph is strongly connected, the run can be extended to the cloud i reading a word of $(a_i\#)^*$. From there, the automaton will read $(a_i\#)^\omega$ while looping between p_i and q_i. We can build an accepting run of $\mathcal{A}$ on any word $w \in L$, so $L \subseteq L(\mathcal{A})$.

Now we shall prove that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle.

($\Rightarrow$) Suppose $\mathcal{A}$ is DBP, and let $\mathcal{D}$ be an equivalent DCW obtained from $\mathcal{A}$ by removing transitions. Notice that this corresponds to choosing one out-edge for each vertex of G. This means it induces a set of disjoint cycles in G. We show that it actually is a unique Hamiltonian cycle. Indeed, assume that some vertex of i is not reachable from 1 in G. Equivalently, it means that some cloud i is not reachable from p_1 in $\mathcal{D}$. This implies that $(a_i\#)^\omega \notin L(\mathcal{D})$, which contradicts $L(\mathcal{D}) = L(\mathcal{A}) = L$. Therefore, $\mathcal{D}$ is strongly connected, and describes a Hamiltonian cycle in G.

($\Leftarrow$) Conversely, if G has an Hamiltonian cycle π , we can build the automaton $\mathcal{D}$ accordingly, by setting for all $i \in [1, n]$, $\Delta_{\mathcal{D}}(r_i, \sharp) = \{p_j\}$ where j is the successor of i in π. Since $\mathcal{D}$ is strongly connected, it still recognises L, and since it is deterministic it is a witness that $\mathcal{A}$ is DBP.

This completes the proof of the fact that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle. Since this is a polynomial time reduction from Hamiltonian Cycle to DBPness of NCW, we showed that checking DBPness of a NCW is NP-hard.

Note that we used $n + 1$ letters here, but it is straightfoward to re-encode this reduction

using only two letters. Therefore, the problem is NP-hard even on a two-letters alphabet. It is trivially in PTIME on a one-letter alphabet, as there is a unique infinite word. ◀

The second part of Theorem 15 is given by the following lemma.

▶ **Lemma 17.** *Checking whether a NCW is DBP is in NP.*

Proof. Suppose a NCW $\mathcal{A}$ is given. We want to check whether it is DBP. We do this via the following NP algorithm.

- Nondeterministically prune transitions of $\mathcal{A}$ to get a deterministic automaton $\mathcal{D}$.
- Check whether $L(\mathcal{A}) \subseteq L(\mathcal{D})$. For that, we check if $L(\mathcal{A}) \cap \overline{L(\mathcal{D})} = \emptyset$

The second step of the algorithm can be done polynomially, since it amounts to finding an accepting lasso in $\mathcal{A} \times \overline{\mathcal{D}}$, where $\overline{\mathcal{D}}$ is a Büchi automaton obtained by dualizing the acceptance condition of $\mathcal{D}$. Finding such a lasso is actually in NL.

Therefore, the above algorithm is in NP, and its correctness follows from the fact that $L(\mathcal{D}) \subseteq L(\mathcal{A})$ is always true, as any run of $\mathcal{D}$ is in particular a run of $\mathcal{A}$.

◀

4.6 Towards Büchi automata

NBW corresponds to the general case of non-deterministic ω-automata, as they allow to recognise any regular language, and are easily computable from non-deterministic automata with stronger accepting conditions.

We will briefly describe the generalisation of previous constructions here, and explain what is the main open problem remaining to solve in order to obtain a satisfying generalisation. We take Safra's construction [17] as the canonical determinisation for Büchi automata. Safra's construction outputs a Rabin automaton.

The idea behind the previous partial determinisation construction can be naturally adapted to Safra: it suffices to restrict the image of the Safra tree labellings to sets of states of size at most k. The bottleneck of the incremental determinisation is then to test for GFGness (or DBPness) of Rabin automata. For DBPness, the same proof as Theorem 15 shows that it is NP-complete. However for GFGness, the complexity is widely open. The only known hardness result is the complexity of solving the games with same acceptance condition [11], known to be in QuasiP for parity [4] and NP-complete for Rabin [7]. In both cases, it is in P if the acceptance condition is fixed. On the other hand, the best known upper bound for GFGness is EXPTIME [11], even for fixed condition, say parity with 3 ranks. Finding an efficient algorithm for GFGness of Rabin (or Parity) automata would be of great interest for this incremental procedure, and would allow to efficiently build GFG automata from NBW.

Acknowledgements. We thank Nathalie Bertrand for helpful discussions.

References

1 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 835–844, 2009.

2 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 89–100, 2013.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 457–468, 2013.

4 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263, 2017.

5 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

6 Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, pages 1–23, 2012.

7 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 328–337, 1988.

8 Kousha Etessami. A hierarchy of polynomial-time computable simulations for automata. In *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, pages 131–144, 2002.

9 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 395–410, 2006.

10 Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. Are good-for-games automata good for probabilistic model checking? In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 453–465, 2014.

11 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 299–310, 2015.

12 H. Leung. Structurally unambiguous finite automata. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4094 LNCS:198–207, 2006. cited By 1.

13 Hing Leung. *Separating exponentially ambiguous NFA from polynomially ambiguous NFA*, pages 221–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

14 HING LEUNG. Descriptional complexity of nfa of different ambiguity. *International Journal of Foundations of Computer Science*, 16(05):975–984, 2005.

15 Christof Löding and Stefan Repke. Decidability results on the existence of lookahead delegators for NFA. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 327–338, 2013.

16 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.

17 Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327, 1988.

18 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325 – 349, 1991.