A Modular Approach for Büchi Determinization*

Dana Fisman[1] **and Yoad Lustig**[2]

1 **University of Pennsylvania, US**
`fisman@seas.upenn.edu`

2 **Google Inc., Israel**
`yoad.lustig@gmail.com`

Abstract

The problem of Büchi determinization is a fundamental problem with important applications in reactive synthesis, multi-agent systems and probabilistic verification. The first asymptotically optimal Büchi determinization (a.k.a. the Safra construction), was published in 1988. While asymptotically optimal, the Safra construction is notorious for its technical complexity and opaqueness in terms of intuition. While some improvements were published since the Safra construction, notably Kähler and Wilke's construction, understanding the constructions remains a non-trivial task.

In this paper we present a modular approach to Büchi determinization, where the difficulties are addressed one at a time, rather than simultaneously, making the solutions natural and easy to understand. We build on the notion of the skeleton trees of Kähler and Wilke. We first show how to construct a deterministic automaton in the case the skeleton's width is one. Then we show how to construct a deterministic automaton in the case the skeleton's width is k (for any given k). The overall construction is obtained by running in parallel the automata for all widths.

1998 ACM Subject Classification F.1.1 Models of Computation, D.2.4 Formal Methods, G.2.2 Graph Algorithms

Keywords and phrases Büchi automata, Determinization, Verification, Games, Synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.368

1 Introduction

The relationship between deterministic and non-deterministic models of computation is a fundamental question in almost every model of computation. Büchi automata are not an exception. Büchi automata were first introduced, as non-deterministic, in [2], in order to prove the decidability of S1S (second order logic of one successor). The need to consider deterministic automata, however, rose almost immediately. A natural extension of S1S decidability was to prove the decidability of S2S (second order logic of two successors) [27]. (Deciding S2S also allowed solving the fundamental Church problem [3] which is an early formulation of what we now call *synthesis*). To decide S2S, however, one needed automata to run on trees, and while this is natural for deterministic automata, it does not work as expected for non-deterministic automata (see [10] for a comprehensive discussion). Some examples of applications requiring Büchi determization constructions can be found in reactive synthesis [26], multi-agent systems [1] and probabilistic verification [34, 4]. In order to work with deterministic automata, one has to tackle another difficulty: deterministic Büchi automata are less expressive than non-deterministic ones [18]. As a result, the determinization

* The work of the first author was supported by US NSF grant CCF-1138996.

of Büchi automata is the following problem: Given a non-deterministic Büchi automaton, find an equivalent deterministic automaton whose acceptance condition may be other than Büchi. (Modern determinization procedures usually use the Parity condition [25, 11].)

Another problem, which is closely related to Büchi determinization is Büchi complementation. Deterministic automata on finite words can be complemented by dualizing the accepting set. Some of the automata on infinite words, namely Muller and Parity, can also be complemented by dualizing the acceptance condition.[1] For this reason determinization constructions yield almost immediately a complementation procedure [28]. In [15] Kupferman and Vardi proposed a complementation construction that actively avoids determinization due to its complexity. Later, in [11], Kähler and Wilke introduced the reduced and skeleton run trees, and both a complementation and a determinization constructions based on them.

The first asymptotically optimal Büchi determinization construction [28] (a.k.a the Safra construction), was published in 1988. While asymptotically optimal, the Safra construction is notorious for its technical complexity and opaqueness in terms of intuition. It took no less than 18 years(!) before Piterman improved upon the Safra construction by modifying it to produce a Parity automaton [25]. A large step forward, in terms of lifting the veil of technical complexity, was made by Kähler and Wilke in [11] who modeled automaton runs by the clear and elegant reduced and skeleton trees. Their complementation construction is as elegant and simple as one can hope for. Their determinization construction is a large step forward, yet it is still non-trivial to understand or implement.

Building on the reduced and skeleton trees of Kähler and Wilke [11], we present in this work a novel Büchi determinization construction which is considerably simpler than previous constructions. The real strength of the construction lies in its modularity. For the first time, the determinization problem is broken down into simpler problems, where the solution of each simple problem is based on one clear idea. Thus the overall solution can be grasped in a gradual manner following several easy steps. This is in contrast to previous solutions in which the correctness reasoning is deferred to the very end, where one needs to reason on a very complex construction.

Overview. In Section 2, we discuss the notions of the *reduced* and *skeleton trees* of [11]. Both trees concisely summarize the runs of a given Büchi automaton $\mathcal{A}$ on given word w, in that these trees have an accepting path iff $\mathcal{A}$ accepts w (where an accepting path is a path with infinitely many accepting nodes). Some of the paths of the *reduced tree* are infinite and some are finite. The *skeleton tree* is the tree obtained from the reduced tree by eliminating the finite paths (i.e. the nodes with finitely many descendants and the edges leading to them). Our constructions conceptually build on the skeleton tree but practically work on the reduced tree. (This is since the skeleton tree, while being easier to reason about, depends on the infinite suffix of the word, and cannot be computed by a deterministic automaton.)

The width of the skeleton tree, formally defined in Section 2, is a central notion in our construction. Roughly speaking, the width of a given level of a tree, is the number of nodes in that level. The width of levels of the skeleton tree are monotonically non-decreasing and bounded by n, the number of states in the given Büchi automaton $\mathcal{A}$. We refer to the maximal width of the skeleton-tree levels by *skel-width*. We use *slice* to refer to the sequence of nodes on a level.

[1] Some, namely Büchi, cannot. Dualization of a Rabin automaton yields a Streett automaton and vice versa.

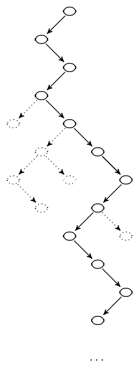

Figure 1 A reduced tree with skel-width one.

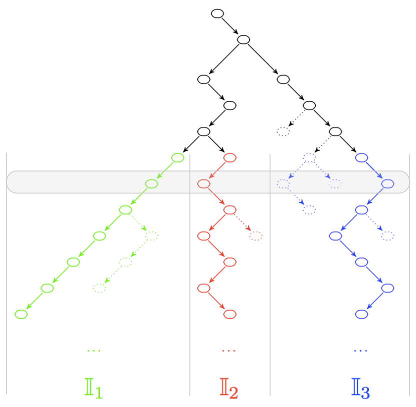

Figure 2 A reduced tree of skel-width 3, marked with the partitions created by the *decide-width* construction. The partitions manage to trap each skeleton path in its own interval in the following sense: Starting from level 5, where the skel-width stabilizes, each slice (see e.g. the marked slice) has three intervals where the i-th interval contains only nodes of the i-th skeleton path, or the finite paths attached to it.

In Section 3, we discuss only words on which the *skel-width* is one, as depicted in Figure 1.[2] For such words, we note that the word is accepted by $\mathcal{A}$ iff it is accepted by the construction of Miyano-Hayashi [22] (henceforth MH) applied to the reduced tree. This is since the MH construction answers if *all* infinite runs of an automaton are accepting, and in the case there exists just *one* infinite run, asking whether *all* infinite runs is the same as asking if there *exists* an infinite run. For the same reason, if we apply it on the reduced tree when the width is greater than one, we may reject if one path accepts but another does not. So in a sense, our mission is to separate the paths of the skeleton tree.

In Section 4 we answer the decision problem: Is the skel-width of the reduced tree smaller than a given k? The solution to this problem creates a partition of a slice of the reduced tree into *intervals*, i.e. consecutive sequences of nodes of the slice, such that each skeleton path (along with the finite paths attached to it) resides in its own interval, as depicted in Figure 2. We capitalize on this separation in consequent constructions.

In Section 5, we show that given a possible width k, we can construct a deterministic automaton that accepts a word of width exactly k iff $\mathcal{A}$ accepts it. This construction essentially runs both previous constructions together, by applying the MH construction on the intervals of the *decide width* construction.

In Section 6, we show how to run in parallel the constructions for all k (upto the number of $\mathcal{A}$'s states), and deduce the correct answer for words of any width. Finally, we show that the complexity of this construction is bounded by $n^{O(n)}$ states, essentially matching the known lower bound of $n!$ given by Michel [21]. In Appendix A, we give a one page illustrative description of the constructions using tokens, bells and buzzers, as a recap. All missing proofs are available in the full version of the paper. We conclude in Section 7.

[2] In this figure (and the rest of the figures in the paper) the reduced tree consists of all nodes and edges, and the skeleton tree of only the solid nodes and edges.

Related Work. The first construction for Büchi determinization is due to McNaughton [20] and dates to 1966. The complexity of this construction was $2^{2^{O(n)}}$, and it took a series of improvements [31, 24] until in 1988, Safra came up with the first asymptotically optimal construction [28] of $n^{O(n)}$. There was a series of works closely studying the exact bounds of Safra, and suggesting improvements obtaining tighter bounds [29, 15, 32, 25, 8, 36, 19].

The works most related to the one presented here were concerned, as we are, in obtaining a determinzation construction simpler than the Safra construction (and the subsequent work), while remaining in $n^{O(n)}$. A state in the determinized automaton of the Safra construction is a labeled tree of sets of states of the given Büchi automaton. The transition between states, and the acceptance condition relies on the nodes labeling. Piterman [25] simplified the labeling and provided a determinized automaton based on Parity acceptance condition, which is simpler and better for many applications [25]. Schewe [30] moved the acceptance condition from states to edges. In 1995, Muller and Schupp [23] proposed a different exponential determinization construction. Kähler and Wilke [11] identified from [23] the so called *skeleton tree* and proposed a construction unifying the determinization, complementation and disambiguation problems of Büchi automata. The most recent works in this thread of research introduce the notion of *profile trees* where a profile is the history of visits to accepting states [33, 6, 7].

A related line of research is that of finding a Büchi complementation construction that does not rely on Büchi determinization [31, 12, 15, 32, 16, 9, 8, 35]. Avoiding determinization, or the Safra construction, was also pursued in other contexts e.g. games and tree automata [17], compositional synthesis [14] and translating linear temporal logic to automata [5].

2 Preliminaries

Buchi and Parity Automata. An automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, I, \delta, \alpha)$ where Σ is the alphabet, Q is a set of states, $I \subseteq Q$ is the set of initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition relation, and α is the acceptance condition. A run of the automaton on an infinite word $w = \sigma_1\sigma_2\sigma_3 \ldots$ is a sequence of states $q_0, q_1, q_2, \ldots$ such that $q_0 \in I$ and $q_i \in \delta(q_{i-1}, \sigma_i)$ for every $i \geq 1$. For Büchi automata the acceptance condition is a set $F \subseteq Q$ and a run is *accepting* if it visits states in F infinitely often. We use $\overline{F}$ to denote the set of non-accepting states, i.e. $Q \setminus F$. The acceptance condition of a Parity automaton is a coloring (or ranking) function χ that associates with each state a color (or rank) from a given finite set of colors $\{0, 1, 2, \ldots, m\}$. A run of the automaton is considered *accepting* iff the minimal color visited infinitely often is odd. An automaton may have several runs on the same word, and it accepts a word iff one of its runs on that word accepts it. An automaton is *deterministic* if $|I| = 1$ and $|\delta(q, \sigma)| = 1$ for every $q \in Q$ and $\sigma \in \Sigma$. A deterministic automaton has a single run on each word.

We refer to automata by three letter acronyms. The first letter may be D or N signifying if the automaton is deterministic or non-deterministic, the second letter may be B, P or F signifying whether it is a Büchi automaton, a Parity automaton or an automaton on finite words. The third letter signifies the object on which the automaton runs, which is W for words in all automata in this paper. For example, NBW stands for non-deterministic Büchi automaton on words, and DFW stands for deterministic finite automaton on words.

Words and Trees. We use the term word for a finite or infinite sequence of letters. We use $w[j]$ for the j-the letter of w, $w[j..]$ for the suffix of w starting at j, $w[i..j]$ for the infix of w starting at i and ending in j, and $w[..j]$ for the prefix of w up to z. Note that $w[1]$ is the first letter of w, so is $w[..1]$. We use $w[..0]$ for the empty string. In general, we use $[1..n]$ for the set $\{1, 2, \ldots, n\}$.

We use annotated binary trees. The nodes of such trees are strings in $\{0,1\}^*$. The root node is the empty string ϵ. The left successor of a node t is $t0$ and the right successor is $t1$. An *annotated binary tree* is a function from $\{0,1\}^*$ to some domain D. A node mapped to $\emptyset$ is regarded as absent from the tree. The *depth* of node t, denoted $|t|$, is its distance from the root, and it equals the length of the string t. So the depth of the root is 0, and the depth of its successors are 1. For $t_1, t_2 \in \{0,1\}^*$ we say that $t_1 <_{\text{lex}} t_2$ if the string t_1 is lexicographically smaller than t_2. We say that $t_1 <_{\text{lft}} t_2$ if $|t_1| = |t_2|$ and $t_1 <_{\text{lex}} t_2$. The *i-th level* of an annotated tree T consists of all nodes in depth i. If $t_1 <_{\text{lex}} t_2 <_{\text{lex}} \cdots <_{\text{lex}} t_n$ are all nodes the i-th level of T then the *i-th slice* of T is the sequence $\langle T(t_1), T(t_2), \ldots, T(t_n)\rangle$. The width of the i-th level is the number of nodes in that level.

Summarizing Runs Concisely. The main task of a determinization construction is to find a way to summarize all the information needed on all the runs of a given non-deterministic automaton in the single run of the constructed deterministic automaton. On finite words, it sufficed to record the set of reachable states. It is instructive to see why this approach fails for Büchi automata. The unfamiliar reader is referred to the full version.

Kähler and Wilke [11] introduced the *split tree*, the *reduced tree* and the *skeleton tree*, all of which concisely summarize the information needed on the runs of the non-deterministic Büchi automaton. The split- reduced- and skeleton-trees are defined per a given word w. A key invariant that is maintained is that if there exists an accepting run of the automaton on w then there is an accepting infinite path in all of these trees. The formal definitions of these trees is given Appendix B of the full version. Roughly speaking, the *split tree* refines the subset construction by separating accepting and non-accepting states. From each node of the tree the left son holds the accepting states and the right son the non-accepting. Thus an accepting path has infintely many left turns, and is also referred to as *left recurring*. The width of a slice of the split-tree is generally unbounded. The *reduced tree* bounds the number of nodes on a slice of the tree to n, the number of states of the given Büchi automaton $\mathcal{A}$, by eliminating from a node of the tree all states that appeared in a node to its left. The *skeleton-tree* is the smallest sub-tree of the reduced-tree that contains all its infinite paths.

▶ **Example 1.** Figure 3 shows a Büchi automaton $\mathcal{W}$ and the reduced and skeleton trees for a^ω and $(ab)^\omega$ both of which have width 2. The word a^ω is rejected and indeed none of the two skel-paths is left recurring. The word $(ab)^\omega$ is accepted and indeed one of the two skel-paths is left recurring.

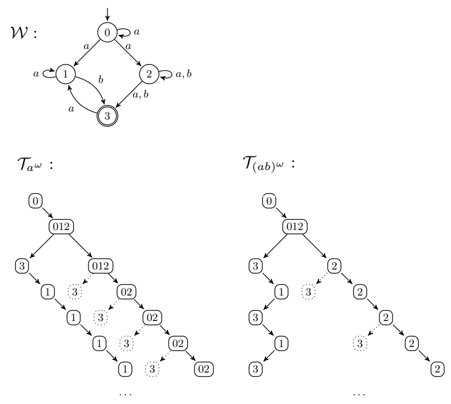

Figure 3 A Büchi automaton $\mathcal{W}$ and its reduced and skeleton trees for a^ω and $(ab)^\omega$ (where a node labeled with multiple digits stands for the subset containing the corresponding states (e.g. 012 stands for $\{0,1,2\}$).

Computing the slices of the reduced tree. The skeleton tree thus concisely captures whether the original automaton $\mathcal{A}$ has an accepting path on the given word w. Unfortunately, it requires knowing or guessing which states will eventually have no successors, which seems a difficult if not impossible task for a deterministic automaton to conduct. Working with the reduced tree, however, is practical.

A *slice* of the reduced tree is a sequence of length at most n of nodes of the reduced tree i.e. an element of $(2^Q)^{\leq n}$. The 0-th slice is $\langle I \rangle$. Given a slice $\mathbb{S} = \langle S_1, S_2, \ldots, S_k \rangle$ we can compute the next slice with respect to a next letter σ as follows [11]. For $1 \leq i \leq k$, let

$$\tilde{S}_i = \cup \{S'_j \mid j < 2i\} \qquad S'_{2i} = \delta(S_i, \sigma) \cap F \setminus \tilde{S}_i \qquad S'_{2i+1} = \delta(S_i, \sigma) \cap \overline{F} \setminus \tilde{S}_i$$

Let $\mathbb{S}' = \langle S'_2, S'_3, \ldots, S'_{2k+1} \rangle$. Let $\mathbb{S}'' = \langle S''_1, S''_2, \ldots, S''_\ell \rangle$ be the sequence obtained from $\mathbb{S}'$ by deleting all the empty sets. Then the next slice of $\mathbb{S}$ with respect to $\mathcal{A}$ and σ, denoted $\delta_{\mathbf{RS}}(\mathbb{S}, \sigma)$ is $\mathbb{S}''$. We define $\delta_{\mathbf{RS}}$ also for a given interval $\mathbb{I} = \langle S_i, ..., S_j \rangle$ of the slices' nodes. Let $\mathbb{I}' = \langle S'_{2i}, S'_{2i+1}, \ldots, S'_{2j+1} \rangle$. Let $\mathbb{I}''$ be the sequence obtained from $\mathbb{I}'$ by deleting all the empty sets. Then $\delta_{\mathbf{RS}}(\mathbb{S}, \mathbb{I}, \sigma) = \mathbb{I}''$.

3 Determinizing assuming the width is one

We first tackle the simplest case where the skeleton's width is one, i.e, there is a single infinite run in the reduced tree. Our automaton then needs to check whether this path visits the accepting set infinitely often. The problem is that we can process the reduced tree, not the skeleton tree, so if we encounter an accepting node we don't know if it is on the skeleton tree or not. Demanding that there exists infinitely many slices where *all* the nodes are accepting is too strong, as can be seen by considering $\mathcal{T}^{\mathcal{A}}_{a^\omega}$ of Fig. 4, which although is accepting all its slices consists at least one non-accepting node. Demanding that there exists infinitely many slices where *there exists* at least one node which is accepting is too weak, as can be seen by considering $\mathcal{T}^{\mathcal{R}}_{a^\omega}$ of Fig. 4, which although is rejecting all its slices consists at least one accepting node.

So we need something a little more sophisticated. The breakpoint construction of Miyano and Hayashi [22] (henceforth, the MH-construction) answers whether all infinite runs of a given non-deterministic Büchi automaton are accepting. Since, in the case considered here, we have just one infinite path, asking whether *all* infinite paths are accepting, is the same as asking whether *the single* infinite path is accepting. Thus, applying the MH-construction on the slices of the reduced tree achieves what we want, when the skeleton width is one.

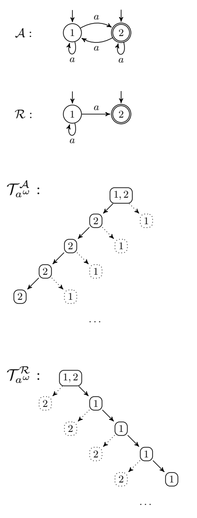
$\mathcal{A}$:
$\mathcal{R}$:
$\mathcal{T}^{\mathcal{A}}_{a^\omega}$:
$\mathcal{T}^{\mathcal{R}}_{a^\omega}$:
a
1
2
1,2
...

Figure 4 The reduced and skeleton trees of a^ω for $\mathcal{A}$ and $\mathcal{R}$.

The Miyano-Hayashi Construction. The idea of the MH-construction is, in addition to tracking the successors of the current level as in the subset construction, to maintain a bookkeeping about which path has *visited the accepting set recently*. Other states are considered to *owe* a visit to the accepting set. Each state of a layer thus carries with it a bit informing whether it owes a visit or not. When a new layer is constructed the sons of states that owe a visit, are also marked as owing a visit unless they are accepting (in which case the corresponding path has just paid its debt). The other states on the layer are not marked as owing since they are known to have visited the accepting set recently. When none of the states is marked as owing we have found evidence for all of them to visit the accepting set recently. We thus start charging them again for a visit to the accepting set. Such a step is considered a *reset* step. Visiting an accepting state recently thus means visiting an accepting state since the last reset occurred. If there are infinitely many reset steps, we know that between every two adjacent reset points all paths have visited an accepting state at least once, and therefore all paths have visited the accepting set infinitely often.

Formally, if $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ then the MH-construction results in $\mathcal{E} = (\Sigma, Q_\mathcal{E}, Q^0_\mathcal{E}, \delta_\mathcal{E}, F_\mathcal{E})$ defined as follows. One can think of a state of $\mathcal{E}$ as a mapping $f : Q \to \{\mathsf{a}, \mathsf{o}, \mathsf{v}\}$ where a state of $\mathcal{A}$ mapped to a is *absent* from the layer, a state mapped to o is on the layer and *owes* a visit, and a state mapped to v is in the layer and *visited* the accepting set since the last reset. Alternatively, we can think of the states of $\mathcal{E}$ as pairs of subsets $\langle Q_L, Q_O\rangle$ where Q_L are the states in the layer (those that are mapped to v or o) and Q_O are the states in the layer that owe a visit (those that are mapped to o). The transition relation is then defined as $\delta_\mathcal{E}(\langle Q_L, Q_O\rangle, \sigma) = \langle Q'_L, Q'_O\rangle$ where $Q'_L = \delta(Q_L, \sigma)$ and if $Q_O \neq \emptyset$ then $Q'_O = \delta(Q_O, \sigma) \setminus F$ and otherwise (when $Q_O = \emptyset$) $Q'_O = \delta(Q_L, \sigma) \setminus F$. The accepting states of $\mathcal{E}$ are the reset states, i.e. those of the form $\langle Q_L, \emptyset\rangle$. The initial state is $\langle I, \emptyset\rangle$.

▶ **Claim 1** ([22]). *$\mathcal{E}$ accepts w iff all paths of $\mathcal{A}$ accept w.*

Miyano-Hayashi on the Reduced Tree. The reduced tree already computes the next successors and separates the states into accepting and non-accepting. The remaining task is to carry the bookkeeping bit of the MH-construction. This can be achieved by the DBW $D_1 = (\Sigma, Q_1, I_1, \delta_{\mathbf{MHoR}}, F_1)$ as follows. The states Q_1 are annotated slices of the form $\langle(S_1, b_1), (S_2, b_2), \ldots, (S_k, b_k)\rangle$. That is, $Q_1 \subseteq (2^Q \times \{\mathsf{o}, \mathsf{v}\})^{\leq n}$. The accepting states F_1 are those where all the b_i components are v. The initial state I_1 is $\langle(I, \mathsf{o})\rangle$. The transition relation builds the next slice of the reduced tree, and annotates the node as dictated by the MH-construction. (A node of the reduced tree is considered to be *accepting* if it consists of accepting states, otherwise it is considered *non-accepting.* Note that in the reduced tree there are no nodes with both accepting and rejecting states.) Formally, given an annotated slice $\mathbb{S} = \langle(S_1, b_1), (S_2, b_2), \ldots, (S_k, b_k)\rangle$, a next letter σ, for $1 \leq i \leq k$, let

$$\begin{array}{ll} \tilde{S}_i = \cup\,\{S'_j \mid j < 2i\} & b'_{2i} = \mathsf{v} \\ S'_{2i} = \delta(S_i, \sigma) \cap F \setminus \tilde{S}_i, & b'_{2i+1} = \begin{cases} \mathsf{o} & \text{if } \forall i.\ b_i = \mathsf{v} \\ b_i & \text{otherwise} \end{cases} \\ S'_{2i+1} = \delta(S_i, \sigma) \cap \overline{F} \setminus \tilde{S}_i & \end{array}$$

Let $\mathbb{S}' = \langle(S'_2, b'_2), (S'_3, b'_3), \ldots, (S'_{2k+1}, b'_{2k+1})\rangle$. Let $\mathbb{S}''$ be the sequence obtained from $\mathbb{S}'$ by deleting all pairs (S'_i, b'_i) where S'_i is the empty set. Then $\delta_{\mathbf{MHoR}}(\mathbb{S}, \sigma) = \mathbb{S}''$.

▶ **Proposition 1.** *For any word w for which skel-width$(\mathcal{A}, w) = 1$ the* DBW *constructed above accepts w iff $\mathcal{A}$ accepts w.*

Similar to $\delta_{\mathbf{RS}}$, for later constructions only, we define $\delta_{\mathbf{MHoR}}(\mathbb{S}, \mathbb{I}, \sigma)$ for an interval $\mathbb{I} = \langle(S_i, b_i), \ldots, (S_j, b_j)\rangle$ of $\mathbb{S}$, for some $1 \leq i \leq j \leq k$. Let $\mathbb{I}'$ be the sequence $\langle(S'_{2i}, b'_{2i}),$ $(S'_{2i+1}, b'_{2i+1}), \ldots, (S'_{2j+1}, b'_{2j+1})\rangle$. Let $\mathbb{I}''$ be the sequence obtained from $\mathbb{I}'$ by removing all pairs whose first component is the empty set then $\delta_{\mathbf{MHoR}}(\mathbb{S}, \mathbb{I}, \sigma) = \mathbb{I}''$.

4 Deciding the width

Perhaps the simplest question regarding widths is: *what is the width of the tree?* or put as a decision problem: *is the width of the tree smaller than a given k?* Before tackling this problem, let's consider a simplified case as a motivating discussion. We know the width of the skeleton tree is monotonically non-decreasing, and bounded by n. Suppose we could start at a time point z after the width has already stabilized. Suppose we could track each of the nodes separately from the others, within its own *interval.* If we have ℓ nodes in level z

we would start with ℓ intervals. The successor of a node in interval j are kept within the same interval j.

If the width is k, then exactly k of the nodes we started with at level z have infinitely many descendants, therefore their interval will always be non-empty. If we started with $\ell > k$ nodes, then $\ell - k$ of these have only finitely many descendants and therefore eventually their interval will become empty. If we throw away empty intervals, then we will at some point remain with exactly k non-empty intervals, forever long. Below we develop this intuition to tackle the exact problem of deciding the width.

The states of the constructed DBW $\mathcal{B}_k$ are sequences of sequences of nodes of the reduced tree, corresponding to a slice partitioned into several intervals. While a node of a reduced tree is a subset of $\mathcal{A}$'s states, in this construction a node can be thought of as the smallest element. If $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ we say that $\mathbb{S}$ has ℓ intervals, and denote it $|\mathbb{S}| = \ell$. If $|\mathbb{S}| < k$ we put all nodes of the next step in different (singleton) intervals, to track each of them separately. We call such step a *shredding* step, and such state a *shredding state*. The accepting set is the set of all shredding states. The initial state consists of a single interval $\mathbb{S}_0 = \langle \mathbb{I}_1 \rangle$ where $\mathbb{I}_1 = \langle I \rangle$, i.e. the interval consists of one node – the root of the reduced tree.

We first define the transition relation for a non-shredding state. Let $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ be a state of $\mathcal{B}_k$. We use $\cup\mathbb{S}$ to abbreviate $\cup_{1 \leq i \leq \ell}\ \mathbb{I}_i$, i.e. the set of nodes in all the intervals together. For $1 \leq i \leq \ell$ let $\mathbb{I}'_i = \delta_{\mathbf{RS}}(\cup\mathbb{S}, \mathbb{I}_i, \sigma)$ and let $\mathbb{S}' = \langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_\ell \rangle$. Let $\mathbb{S}''$ be the sequences obtained from $\mathbb{S}'$ by deleting empty intervals. Then if $\mathbb{S}$ is non shredding (i.e. $\ell \geq k$) then $\delta_{\mathcal{B}_k}(\mathbb{S}, \sigma) = \mathbb{S}''$.

To define the transition relation for a shredding state we introduce an additional notation. If $\cup\mathbb{S} = \{S_1, S_2, \ldots, S_m\}$ then we use *Shred*$(\mathbb{S})$ to denote the sequence $\langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_m \rangle$ where $\mathbb{I}'_i = \langle S_i \rangle$. That is, in *Shred*$(\mathbb{S})$ every node of $\mathbb{S}$ is put in its own interval. Now let $\mathbb{I}''_i = \delta_{\mathbf{RS}}(\cup\mathbb{S}, \mathbb{I}'_i, \sigma)$ and let $\mathbb{S}'' = \langle \mathbb{I}''_1, \mathbb{I}''_2, \ldots, \mathbb{I}''_\ell \rangle$. Let $\mathbb{S}'''$ be the sequences obtained from $\mathbb{S}''$ by deleting empty intervals. Then if $\mathbb{S}$ is a shredding state (i.e. $\ell < k$) then $\delta_{\mathcal{B}_k}(\mathbb{S}, \sigma) = \mathbb{S}'''$.

▶ **Proposition 2.** *The* DBW $\mathcal{B}_k$ *constructed above accepts a word* w *iff skel-width*$(\mathcal{A}, w) < k$.

▶ **Example 2.** Consider the NBW $\mathcal{W}$ of Figure 3. Both words a^ω and $(ab)^\omega$ have width 2 (as is evident from the respective skeleton trees). The automata $\mathcal{B}_2$ and $\mathcal{B}_3$ for widths < 2 and < 3 respectively, are given in Fig. 5. The accepting states are marked with a double edge (and also colored blue). It can be seen that $\mathcal{B}_2$ rejects both a^ω and $(ab)^\omega$ as required (since their width is not smaller than 2), while $\mathcal{B}_3$ accepts both words as required (since their width is smaller than 3).

Tracking the path of either words a^ω or $(ab)^\omega$ on $\mathcal{B}_2$ or $\mathcal{B}_3$ (and comparing to the respective trees in Figure 3) we can see how a state represents a slice of the reduced tree, separated into intervals. And that the transition relation maintains the successors within the same interval, unless shredding occurred in which case they are separated into singleton intervals.

Last, we can see that each skeleton path is eventually trapped within its own interval. This brings us to our next claim.

▶ **Claim 2** (skel-paths separation). *Let* w *be a word with skel-width* k, *and let* $\rho_1, \rho_2, \ldots, \rho_k$ *be the skel-paths from left to right. Let* $\mathbb{S}_0, \mathbb{S}_1, \mathbb{S}_2, \ldots$ *be the run of* $\mathcal{B}_k$ *on* w. *Then there exists* $z_1 \in \mathbb{N}$ *such that* $|\mathbb{S}_z| = k$ *for every* $z > z_1$. *Moreover, assume* $\mathbb{S}_z = \langle \mathbb{I}^z_1, \mathbb{I}^z_2, \ldots, \mathbb{I}^z_k \rangle$ *then for every skel-path* $\rho_i = S^i_0, S^i_1, S^i_2, \ldots$, *for every* $z > z_1$, *we have* $S^i_z \in \mathbb{I}^z_i$.

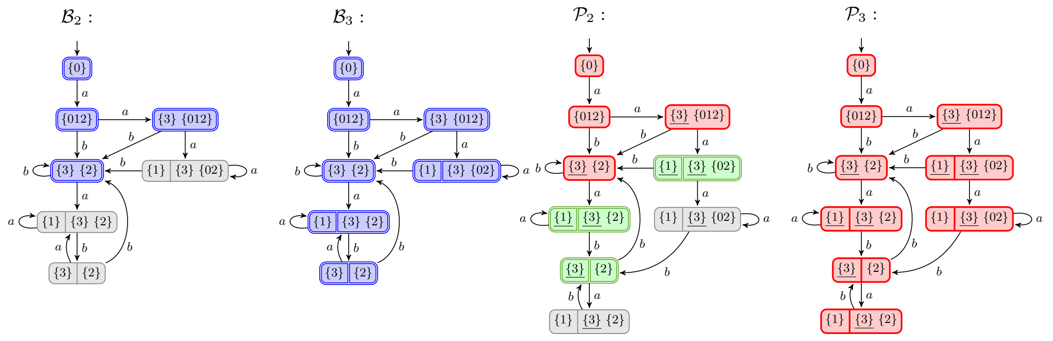

Figure 5 On the left: automata $\mathcal{B}_2$ and $\mathcal{B}_3$ answering whether the width of a word w.r.t. $\mathcal{W}$ given in Fig. 3 is smaller than 2 and 3, respectively. The accepting states are those with a double frame (also colored blue). On the right: automata $\mathcal{P}_2$ and $\mathcal{P}_3$ answering whether a word with width exactly 2 and 3, resp. (w.r.t. $\mathcal{W}$ given in Fig. 3) is accepted by $\mathcal{A}$. The states with a thick frame have color 0 (red). The states with a double frame have color 1 (green). The states with a thin frame have color 2 (gray). An underlined node is a node whose MH-bit is set to v.

5 Determinizing for a given width

We now show that we can build a deterministic parity automaton $\mathcal{P}_k$ that accepts a word of width k iff it is accepted by $\mathcal{A}$. The idea is to combine both constructions we have seen previously. That is, we use the idea of the construction of $\mathcal{B}_k$ to keep the skel-paths separated one from the other in different intervals, and we use the MHoR construction to check on each of these intervals whether the skel-path is accepting. Essentially, we add the Miyano-Hayashi bit to the construction of $\mathcal{B}_k$.

As in $\mathcal{B}_k$ we work with k intervals, and apply shredding when the number of intervals is smaller than k. Instead of using $\delta_{\mathbf{RS}}$ as in $\mathcal{B}_k$ we use $\delta_{\mathbf{MHoR}}$ as in $\mathcal{D}_1$, so the transition relation computes for each successor of a node its MH-bit, and an MH-reset is preformed when all bits of that interval are v. The states of $\mathcal{P}_k$ are sequences of at most n intervals, where each interval is a sequence of pairs whose first element is a node of the reduced tree, and whose second element is a bit in $\{\mathsf{o}, \mathsf{v}\}$. (Again, nodes are subsets of $\mathcal{A}$'s states, but can be thought of as the smallest elements of this construction.) The initial state is the sequence $\mathbb{S}_0$ where $\mathbb{S}_0 = \langle \mathbb{I}_1 \rangle$ and $\mathbb{I}_1 = \langle (I, \mathsf{o}) \rangle$, i.e. the root of the reduced tree, labeled as owing.

Let $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ be a state of $\mathcal{P}_k$. First we define a non-shredding transition (i.e. assume $\ell \geq k$). In this case, for $1 \leq i \leq \ell$ and $\sigma \in \Sigma$, let $\mathbb{I}'_i = \delta_{\mathbf{MHoR}}(\cup \mathbb{S}, \mathbb{I}_i, \sigma)$ (where $\delta_{\mathbf{MHoR}}$ is as defined in Section 3). Let $\mathbb{S}' = \langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_k \rangle$ and let $\mathbb{S}''$ be the sequence obtained from $\mathbb{S}'$ by deleting all pairs with an empty interval. Then $\delta_{\mathcal{P}_k}(\mathbb{S}, a) = \mathbb{S}''$.

For a shredding transition (i.e. when $\ell < k$), if $\cup \mathbb{S} = \{P_1, P_2, \ldots, P_m\}$ where P_i is a pair (S_i, b_i) where S_i is a reduced-tree node and $b_i \in \{\mathsf{o}, \mathsf{v}\}$ then we use *Shred*$(\mathbb{S})$ to denote the sequence $\langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_m \rangle$ where $\mathbb{I}'_i = \langle P_i \rangle$. That is, in *Shred*$(\mathbb{S})$ every pair of $\mathbb{S}$ is put in its own interval. Now let $\mathbb{I}''_i = \delta_{\mathbf{MHoR}}(\cup \mathbb{S}, \mathbb{I}'_i, \sigma)$ and let $\mathbb{S}'' = \langle \mathbb{I}''_1, \mathbb{I}''_2, \ldots, \mathbb{I}''_\ell \rangle$. Let $\mathbb{S}'''$ be the sequences obtained from $\mathbb{S}''$ by deleting the empty intervals. Then $\delta_{\mathcal{P}_k}(\mathbb{S}, \sigma) = \mathbb{S}'''$.

Let $\mathbb{S}$ be a state. The coloring function assigns it 0 if it is a shredding states (i.e. $|\mathbb{S}| < k$); it assigns it 1 if $|\mathbb{S}| \geq k$ and an MH-reset occurs, (i.e. for some interval i all the MH-bits of pairs in that interval are v); and it assigns it 2 otherwise.

▶ **Proposition 3.** *For all words w if skel-width$(\mathcal{A}, w) = k$ then $\mathcal{P}_k$ accepts w iff $\mathcal{A}$ does.*

▶ **Example 3.** Consider again the NBW $\mathcal{W}$ of Figure 3. Figure 5 provides the automata $\mathcal{P}_2$ and $\mathcal{P}_3$ for widths 2 and 3, respectively. The red states (also having a thick frame) have color 0, the green states (also having a double frame) have color 1 and the gray states have color 2. An underlined node corresponds to a node whose MH-bit is set to v. (The un-underlined nodes have their MH-bit set to o.)

It is easy to construct $\mathcal{P}_2$ and $\mathcal{P}_3$ given $\mathcal{B}_2$ and $\mathcal{B}_3$. Indeed they just add the MH-bit for the nodes of $\mathcal{B}_2$ and $\mathcal{B}_3$, respectively, undergoing an MH-reset, if all nodes of an interval are underlined (i.e. have visited the accepting set recently). Note that $\mathcal{P}_2$ has more states than $\mathcal{B}_2$ since for instance, we need two version of the lower right state, for different settings of the node's MH-bits.

Recall that both words a^ω and $(ab)^\omega$ have width 2, thus $\mathcal{P}_2$ should provide the correct result for both of them. Indeed $\mathcal{P}_2$ accepts $(ab)^\omega$ and rejects a^ω exactly as $\mathcal{A}$ does.

We note that $\mathcal{D}_1$ is a private case of the construction of $\mathcal{P}_k$ for $k = 1$. Indeed, for $k = 1$ shredding will never occur, and so all states consist of a single interval which is the entire slice. Thus $\mathcal{P}_1$, exactly as $\mathcal{D}_1$, simply tracks the MH-bits on the reduced tree.

We next turn to understand what answer $\mathcal{P}_k$ provides for words with skel-width different from k. If the actual skel-width is k_* and $k > k_*$ then after the stabilization point, we won't be able to maintain for long more than k_* non-empty intervals, therefore shredding will occur infinitely often and $\mathcal{P}_k$ will reject. If $k_* > k$ then as we state in Proposition 4 below, if $\mathcal{P}_k$ accepts it does so rightfully.

▶ **Proposition 4.** *The* DPW $\mathcal{P}_k$ *described above uses three colors* $\{0, 1, 2\}$ *and for any word* w

- *if skel-width*$(\mathcal{A}, w) = k$ *then* $\mathcal{P}_k$ *accepts* w *iff* $\mathcal{A}$ *does,*
- *if skel-width*$(\mathcal{A}, w) < k$ *then* $\mathcal{P}_k$ *rejects* w*,*
- *if* $\mathcal{P}_k$ *accepts* w *then* w *is accepted by* $\mathcal{A}$*, and*
- $\mathcal{P}_k$ *visits* 0 *infinitely often iff skel-width*$(\mathcal{A}, w) < k$*.*

Self-Correction

We observe that in a sense, the intervals are self-correcting. That is, for every word w and every suffix $w[j..]$ of w, if we are given the slice of the reduced tree on the respective prefix $w[1..j-1]$, partition it arbitrarily to intervals, and apply $\mathcal{B}_k$ from there on that suffix, we will still get a correct result (i.e. a correct answer to the question whether the width of w is smaller than k). Intuitively, because the intervals' role is to detect a property that depends only on the future.

In a similar manner, the MH-bits are self correcting. That is, for every word w and every suffix $w[j..]$ of w, if we are given the slice of the reduced tree on the respective prefix $w[1..j-1]$, and annotate it arbitrarily with MH-bits, and apply MHoR from there on that suffix, we will still get a correct result (i.e. an answer whether w is accepted on all skel-paths). Again, intuitively because the role of the MH-bits is to detect a property that depends only on the future. Putting these together we get the following claim.

▶ **Claim 3** (Intervals and MH-bits are self-correcting)**.** *Let* $\mathbb{S}$ *be a slice of the reduced tree on prefix* $w[1..j]$ *partitioned arbitrarily into intervals, and annotated arbitrarily with MH-bits. Applying the construction of* $\mathcal{P}_k$ *on the suffix* $w[j+1..]$ *from state* $\mathbb{S}$ *still satisfies all premises of Proposition 4.*

6 Determinizing for any width

We now build a deterministic parity automaton $\mathcal{P}$ that recognizes the same language as $\mathcal{A}$. Intuitively, we want to run the automata $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ from the previous section, in parallel. We know that if the exact width is k_* then the answer we look for is given by $\mathcal{P}_{k_*}$ and apart from this, $\mathcal{P}_k$ for all $k > k_*$ rejects, and for all $k < k_*$, if $\mathcal{P}_k$ accepts it does so rightfully. Thus, prioritizing results of $\mathcal{P}_k$'s with lower k's should give us the correct result.

Before we continue, let's contemplate on the complexity we'll get by running all $\mathcal{P}_k$'s in parallel. A state of $\mathcal{P}_k$ needs to encode, among other things, the partition into intervals, which requires at least 2^n states. Since we have n such components in a state, we'd need 2^{n^2} which is more that we can allow, if we'd like to stay in the bounds of $2^{O(n \log n)}$.

We overcome this by working with a modification of $\mathcal{P}_k$, termed $\mathcal{R}_k$, which is obtained using the observation of Claim 3. Specifically, $\mathcal{R}_k$ will differ from $\mathcal{P}_k$ in two places. One is that $\mathcal{R}_k$ will undergo shredding whenever one of the $\mathcal{R}_j$ with $j \leq k$ decides to undergo shredding. This entails that the partitions to intervals of $\mathcal{R}_{k+1}$ will be a refinement of that of $\mathcal{R}_k$. Or put otherwise, every interval of $\mathcal{R}_{k+1}$ would be fully contained in an interval of $\mathcal{R}_k$. The second place is that an interval of $\mathcal{R}_k$ will undergo an MH-reset whenever a subsuming interval of one of the $\mathcal{R}_j$'s for $j \leq k$ decides to undergo an MH-reset. As we shall see in the next subsection, this will enable an encoding of the state space that does not exceed $n^{O(n)}$.

To run these automata in parallel a state of our parity automaton will have n components one corresponding to each of the automata $\mathcal{R}_k$. The initial state will be the n-tuple consisting of the initial states of $\mathcal{R}_1, \mathcal{R}_2 \ldots, \mathcal{R}_n$ and the transition relation will follow that of the $\mathcal{R}_k$'s. For the acceptance condition, we need to assign colors to these compound states.

A compound state has the following form $(\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$ where $\mathbb{S}_i$ is a state of $\mathcal{R}_i$ for $1 \leq i \leq n$. To assign a color to the compound state, it is convenient to let each $\mathcal{R}_i$ use its own set of colors. More precisely, we will assume automaton $\mathcal{R}_k$ assigns colors in $\{2k,\ 2k{+}1,\ 2n{+}2\}$ instead of $\{0, 1, 2\}$, respectively. That is, the colors $2k$ and $2k{+}1$ are uniquely used by $\mathcal{R}_k$ whereas the color $2n{+}2$ may be used by all $\mathcal{R}_k$'s. Now for a given state $\mathbb{P} = (\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$, we can look at the corresponding sequence of colors $(c_1, c_2, \ldots, c_n)$, where c_i has values in $\{2k,\ 2k{+}1,\ 2n{+}2\}$, and color the compound state $\mathbb{P}$ with the minimal color among the c_i's.

▶ **Theorem 4.** *Let $\mathcal{A}$ be an* NBW*, with n states. The* DPW *$\mathcal{P}$ described above has $2n + 1$ colors and it accepts an infinite word w iff $\mathcal{A}$ accepts w.*

Complexity. We turn to show that the complexity of the construction of the DPW $\mathcal{P}$ in Theorem 4 is $n^{O(n)}$.

A state $\mathbb{S}_k$ of $\mathcal{R}_k$ for some $1 \leq k \leq n$ is a slice of the reduced tree, partitioned into intervals, where each node is annotated with its MH-bit. A state of $\mathcal{P}$ is an n-tuple $(\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_k)$ of such states. We shall see that we can encode a state of $\mathcal{P}$ more succinctly than directly encoding each state $\mathbb{S}_i$ of the tuple. In fact, we chose to work with $\mathcal{R}_k$ instead of $\mathcal{P}_k$ for this reason exactly. For $\mathcal{R}_k$ we can show that the intervals of $\mathcal{R}_{k+1}$ refine those of $\mathcal{R}_k$. This will entail also that the MH-bits for all $\mathcal{R}_k$'s can be encoded more compactly.

First, we claim that the intervals of $\mathcal{R}_{k+1}$ refine those of $\mathcal{R}_k$, for every $1 \leq k \leq n$.

▶ **Claim 4** (Intervals refinement)**.** *Every interval $\mathbb{J}_m$ of the state of $\mathcal{R}_{k+1}$ after reading $w[..z]$, is fully contained in an interval $\mathbb{I}_\ell$ of the state of $\mathcal{R}_k$ after reading $w[..z]$, for any $z \in \mathbb{N}$.*

Next, we show that if $b_1, b_2, \ldots, b_n$ are the MH-bit of a given node in $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$ resp. then $b_1 b_2 \ldots b_n \in \mathsf{v}^*\mathsf{o}^*$.

▶ **Claim 5** (MH-bits entailment). *Let $b_k(S)$ and $b_{k+1}(S)$ be the MH-bit encoding of node S in the state of $\mathcal{R}_k$ and $\mathcal{R}_{k+1}$, resp. after reading $w[..z]$ for some $z \in \mathbb{N}$. Then $b_k(S) = \mathsf{o}$ implies $b_{k+1}(S) = \mathsf{o}$, and $b_{k+1}(S) = \mathsf{v}$ implies $b_k(S) = \mathsf{v}$.*

We are now ready to fully encode a state $\mathbb{P} = (\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$ of $\mathcal{P}$. A state $\mathbb{S}_i$ needs to record (1) the original automaton states that are on the slice, in the order they appear, (2) the partition to nodes, (3) the partition to intervals and (4) the Miyano-Hayshi bit of each node. We have that (1) and (2) are shared among all $\mathbb{S}_i$'s since they all track the obtained slice of the reduced tree. We'll use Claims 4 and 5 to represent (3) and (4) in a combined manner, rather than separately for each $\mathbb{S}_i$.

The states on the slice, can be represented by a permutation on $[1..n]$.[3] This requires $n! < n^{O(n)}$. The partitions of states to nodes, can be encoded using a function h from $[1..n]$ to $\{0, 1\}$, so that $h(i) = 1$ means that a new node starts after the i-th state of slice (i.e. the i-th and $i{+}1$-th states of the slice are in different nodes). This requires $2^n < n^{O(n)}$. The partition to intervals for all $\mathcal{R}_k$'s together, by Claim 4, can be encoded by a function F from $[1..n]$ to $[0..n]$ so that $F(i) = j$ means that a new interval begins between states i and $i{+}1$ of the slice, in all $\mathcal{R}_k$ for $k > j$. This requires $n^{n+1} < n^{O(n)}$. Finally, let $[1..n]_h$ be the set of indices for which $h(i) = 1$. That is, $[1..n]_h$ represents the nodes of the slice. By Claim 5, the MH-bits of nodes in all $\mathcal{R}_k$'s together can be represented by a function G from $[1..n]_h$ to $[1..n+1]$ so that $G(i) = j$ means that node i is v in all $\mathcal{R}_k$ for $k < j$ and it is o, in all $\mathcal{R}_k$ for $k \geq j$. This requires $n^{n+1} < n^{O(n)}$. So all in all, $n^{O(n)}$ suffices to represent all states of $\mathcal{P}$ from Theorem 4.[4]

▶ **Corollary 5.** *Let $\mathcal{A}$ be an* NBW*, with n states. The* DPW *$\mathcal{P}$ described above has $n^{O(n)}$ states and $2n + 1$ colors and it accepts an infinite word w iff $\mathcal{A}$ accepts w.*

7 Conclusions

Building on the reduced and skeleton trees of Kähler and Wilke [11] we provide a novel simple construction for Büchi determinization.

A key observation is that *conceptually* we can partition the reduced tree into *intervals* so that each infinite path of the reduced tree (along with the finite paths attached to it) resides in an interval of its own, as depicted in Figure 2. The word is accepted by $\mathcal{A}$ iff in one of these intervals the infinite path is accepting. The question whether in such an interval the infinite path is accepting, can be answered by applying the MH-construction on that interval.

We show how we can build an automaton $\mathcal{P}_k$ that for words with width k, finds this conceptual separation and by applying the MH-construction to each of the k intervals, returns a correct result for all words of width k. Our overall construction runs in parallel (a minor tweaking of) the automata $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ to produce a correct result for words of any width.[5]

We have thus broken the determinization problem into two simpler problems (1) *partitioning* the reduced tree into its skeleton-paths, and (2) providing an answer for a *single* infinite path (in the presence of finite paths). For the latter we adapted the MH-construction.

[3] We assume some arbitrary given order $\prec$ on the states of $\mathcal{A}$ so that states in the same node will be ordered according to $\prec$, and we obtain a total order on the states of a slice.

[4] In fact, the partition to nodes can be seen as the finest refinement of the partition to intervals, so we can represent both (2) and (3) together by a function from $[1..n]$ to $[0..n{+}1]$. So the overall number of state is bounded by $n! \cdot n^{n+2} \cdot n^{n+1} < n^{3n+3}$.

[5] Appendix A provides an illustrative presentation of the constructions using tokens, bells, and buzzers.

We tackled the former by answering the elemental decision problem of whether the number of infinite paths in a tree with both finite and infinite paths is smaller than a given k.

Acknowledgements. We would like to thank Nir Piterman for his helpful comments on a draft of this paper.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *In Proc. 38th Symp. on Foundations of Computer Science*, pages 100–109, 1997.

2 J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

3 A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume I, pages 3–50, 1957.

4 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

5 J. Esparza and J. Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In *In Proc 26th Int. Conf. on Comp. Aided Verif.*, pages 192–208, 2014.

6 S. Fogarty, O. Kupferman, M.Y. Vardi, and T. Wilke. Profile trees for büchi word automata, with application to determinization. In *GandALF*, pages 107–121, 2013.

7 S. Fogarty, O. Kupferman, T. Wilke, and M.Y. Vardi. Unifying büchi complementation constructions. *ogical Methods in Computer Science*, 9, 2013.

8 E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17:851–868, 2004.

9 S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *In Proc. 12th Conf. on CHARME*, volume 2860 of *LNCS*, pages 96–110. Springer, 2003.

10 T.A. Henzinger and N. Piterman. Solving games without determinization. In *Annual Conf. of the Eur. Asso. for Comp. Sci. Log.*, volume 4207 of *LNCS*, pages 394–410, 2006.

11 D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP08*, volume 5125 of *LNCS*, pages 724–735. Springer, 2008.

12 N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *In Proc. 32nd Symp. on Found. of Comp. Sci.*, pages 358–367, 1991.

13 D. Kozen. *Theory of Computation.* Texts in Computer Science. Springer, 2006.

14 O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc 18th of CAV*, volume 4144 of *LNCS*, pages 31–44. Springer, 2006.

15 O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158, 1997.

16 O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *In Proc. 16th IEEE Symp. on Logic in Computer Science*, pages 389–398, 2001.

17 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

18 L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

19 W. Liu and J. Wang. A tighter analysis of piterman's büchi determinization. *Inf. Process. Lett.*, 109(16):941–945, 2009.

20 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

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

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

23 D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata. *TCS*, 141:69–107, 1995.

24 Jean-Pierre Pécuchet. On the complementation of büchi automata. *Theoretical Computer Science*, 47:95–98, 1986.

25 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, 2006.

26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *In Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

27 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

28 S. Safra. On the complexity of ω-automata. In *In Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

29 S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, Victoria, 1992.

30 S. Schewe. Tighter bounds for the determinisation of büchi automata. In *Found. of Soft. Sci. and Comp. Struc. 12th Inter. Conf. FOSSACS*, pages 167–181, 2009.

31 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.

32 W. Thomas. Complementation of büchi automata revised. *Jewels are Forever*, pages 109–120, 1999.

33 M.-H. Tsai, S. Fogarty, M.Y. Vardi, and Y.-K. Tsay. State of büchi complementation. In *15th Int. Conf. on Impl. and Appl. of Aut.*, volume 6482, pages 261–271, 2010.

34 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *n Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 327–338, 1985.

35 M.Y. Vardi. The büchi complementation saga. In *In Proc. 24th Symp. on Theoretical Aspects of Computer Science*, pages 12–22, 2007.

36 Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd ICALP*, volume 4052 of *LNCS*, pages 589–600. Springer, 2006.

A Recap – The Construction using Tokens, Buzzers and Bells

Below we give a description of our construction, using tokens, bells and buzzers in the style of the representation of Safra's construction in [13]. The tokens are a visual means to describe the states of the automaton. For a parity automaton with 3 colors $\{0, 1, 2\}$ the bell corresponds to a state colored 1 and the buzzer to a state colored 0, so that a word is rejected if the buzzer buzzed infinitely many times, and otherwise, accepted if the bell rang infinitely many times. For a parity automaton with $2m$ colors, we use m buzzers and m bells. Suppose during a run on the word the minimal buzzer to buzz infinitely often is k_{buzz} and the minimal bell to ring infinitely often is k_{bell}. Then $k_{buzz}, k_{bell} \in [1..m] \cup \{\infty\}$ and if $k_{bell} < k_{buzz}$ the word is accepted.

For width one. The construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with *violet* and *orange* tokens (for the MH bits v and o, resp). On the root of the tree, put violet if it is accepting, and orange otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a violet token, put a violet token. For a son of an orange token, put a violet token if it is accepting and an orange token otherwise. If all tokens in the slice are violet, ring the bell. Then replace the tokens on all the non-accepting nodes with orange tokens.

For width k. The construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with *violet* and *orange* tokens that are numbered 1 to n (where n is the number of states of the given NBW). That is, $\mathit{tokens} = \{\mathit{violet}(i) \mid i \in [1..n]\} \cup \{\mathit{orange}(i) \mid i \in [1..n]\}$. We use $\mathit{token}(i)$ for $\mathit{violet}(i) \vee \mathit{orange}(i)$.

On the root of the tree, put $\mathit{violet}(1)$ if it is accepting, and $\mathit{orange}(1)$ otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a $\mathit{violet}(i)$ token, put a $\mathit{violet}(i)$ token. For a son of an $\mathit{orange}(i)$ token, put a $\mathit{violet}(i)$ token if it is accepting and an $\mathit{orange}(i)$ token otherwise. If the number of i's for which $\mathit{token}(i)$ is in the current slice is less than k, buzz the buzzer. Then put on each node of the slice a new token, with i's increasing from left to right, and with $\mathit{orange}(i)$ placed on a non-accepting node and $\mathit{violet}(i)$ on an accepting node. If for some i, all $\mathit{token}(i)$ are violet, ring the bell. Then replace all the $\mathit{violet}(i)$ tokens that are on a non-accepting node with $\mathit{orange}(i)$ tokens.

Overall construction.[6] Again, the construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with tokens. Here we use n sets of violet and orange tokens, numbered 1 to n. That is, $\mathit{tokens} = \{J\text{-}\mathit{violet}(i) \mid J, i \in [1..n]\} \cup \{J\text{-}\mathit{orange}(i) \mid J, i \in [1..n]\}$. We use $J\text{-}\mathit{token}(i)$ for $\{J\text{-}\mathit{violet}(i), J\text{-}\mathit{orange}(i)\}$ and $J\text{-}\mathit{token}()$ for $\{J\text{-}\mathit{violet}(i) \mid i \in [1..n]\} \cup \{J\text{-}\mathit{orange}(i) \mid i \in [1..n]\}$.

On the root of the tree, for every $J \in [1..n]$, put $J\text{-}\mathit{violet}(1)$ if it is accepting, and $J\text{-}\mathit{orange}(1)$ otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a $J\text{-}\mathit{violet}(i)$ token, put a $J\text{-}\mathit{violet}(i)$ token. For a son of a $J\text{-}\mathit{orange}(i)$ token, put $J\text{-}\mathit{violet}(i)$ if it is accepting and $J\text{-}\mathit{orange}(i)$ otherwise. If for some J the number of used $J\text{-}\mathit{token}()$'s on the current slice is less than J, buzz the J-buzzer. Then, for all $J' \geq J$, put on all nodes of the slice a new $J'\text{-}\mathit{token}()$, with i's increasing from left to right, and with $J'\text{-}\mathit{orange}(i)$ placed on a non-accepting node and $J'\text{-}\mathit{violet}(i)$ on an accepting node. If for some i and J, all $J\text{-}\mathit{token}(i)$ are violet, ring the J-bell. The for every non-accepting node with $J\text{-}\mathit{violet}(i)$ on it, for every $J' \geq J$, if the current $J'\text{-}\mathit{token}()$ on it is $J'\text{-}\mathit{violet}(i')$, replace it with a $J'\text{-}\mathit{orange}(i')$ token.

[6] The construction here incorporates the modification required to get the complexity of $n^{O(n)}$.