Determinization of Büchi Automata: Unifying the Approaches of Safra and Muller-Schupp

Christof Löding
RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany
loeding@cs.rwth-aachen.de

Anton Pirogov
RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany
pirogov@cs.rwth-aachen.de

Abstract

Determinization of Büchi automata is a long-known difficult problem, and after the seminal result of Safra, who developed the first asymptotically optimal construction from Büchi into Rabin automata, much work went into improving, simplifying, or avoiding Safra's construction. A different, less known determinization construction was proposed by Muller and Schupp. The two types of constructions share some similarities but their precise relationship was still unclear. In this paper, we shed some light on this relationship by proposing a construction from nondeterministic Büchi to deterministic parity automata that subsumes both constructions: Our construction leaves some freedom in the choice of the successor states of the deterministic automaton, and by instantiating these choices in different ways, one obtains as particular cases the construction of Safra and the construction of Muller and Schupp. The basis is a correspondence between structures that are encoded in the macrostates of the determinization procedures – Safra trees on one hand, and levels of the split-tree, which underlies the Muller and Schupp construction, on the other hand. Our construction also allows for mixing the mentioned constructions, and opens up new directions for the development of heuristics.

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

Keywords and phrases Büchi automata, determinization, parity automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.120

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.02139`.

Funding *Anton Pirogov*: This author is supported by the German research council (DFG) Research Training Group 2236 UnRAVeL

1 Introduction

Büchi automata are finite automata for infinite words, and were initially introduced to show decidability of the logic S1S [2]. Infinite words can be used to model infinite execution traces of reactive, non-terminating systems, and serve as a translation target from logics like LTL (see, e.g., [15, 5]), which is a popular and well-understood specification formalism. For this reason, Büchi automata nowadays play a central role in formal methods like model-checking [1] and runtime-verification [6], because they can represent all ω-regular languages and are suitable for efficient algorithmic treatment. Unfortunately, the simplicity of the Büchi acceptance condition makes it crucially dependent on nondeterminism, i.e., not every ω-regular language (or LTL formula) can be accepted by a deterministic Büchi automaton (see, e.g., [16]). In some settings, this nondeterminism causes difficulties, such that algorithms require a representation of the property by a deterministic automaton, like in probabilistic model-checking (see, e.g., [1, Section 10.3]), or in synthesis (see [17] for an overview of the theory, and [9] for recent developments in practice).

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 120; pp. 120:1–120:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A first determinization procedure that translates nondeterministic Büchi automata into deterministic automata was presented in [8]. The first asymptotically optimal and most well-known determinization construction for Büchi automata is the construction of Safra [13]. It translates a nondeterministic Büchi automaton with n states into a deterministic Rabin automaton with at most $2^{\mathcal{O}(n \log n)}$ states and $\mathcal{O}(n)$ sets in the acceptance condition. In applications like synthesis, the deterministic automaton is used to build a game that inherits as winning condition the acceptance condition of the automaton. In the theory of infinite duration games, the parity condition plays a central role (see, e.g., the survey [18]). For this reason, Piterman modified Safra's construction in order to directly obtain a parity automaton [11]. This construction was reformulated in [14], where also a tighter analysis of its state complexity is given with an upper bound of $\mathcal{O}(n!^2)$ for the number of states. A similar construction is presented in [12], adapted to the translation of ω-regular expressions directly into parity automata.

It is known that the Safra construction is essentially optimal [3], so there is no hope of significantly improving the worst-case upper bounds of the known constructions. However, the data structure of Safra trees (or history trees) that is used for the states of the deterministic automata, is challenging to deal with in implementations. Therefore, alternative approaches for determinization have been studied, leading to a family of constructions that are based on a construction by Muller and Schupp, which appeared in [10] as a by-product of a translation from alternating to non-deterministic tree automata. An explicit description of the construction specifically for determinization of Büchi automata is presented in [7]. A refinement of that construction is presented in [4], in which the states of the deterministic automaton are no longer represented as trees but as ordered and labelled tuples of sets.

The two approaches of Safra and Muller-Schupp show some similarities, as pointed out in the conclusion of [4], but from the existing formulations of the constructions, their precise relationship is not clear.

In this paper, we provide a construction for transforming nondeterministic Büchi automata into deterministic parity automata that cleanly explains the connections between the approaches of Safra and Muller-Schupp. It turns out that both types of constructions can be formulated on the same data structure, which can either be understood as ordered tuples of sets in which each set has an additional rank (a natural number), or as Safra trees in which each node has an additional rank (the same structure is essentially used in the constructions from [11] and [14]). The transitions are defined in terms of a sequence of simple operations, and it turns out that the two constructions only differ in one of these operations. In summary, our contributions are the following:

- We provide a new and relatively simple formulation of a Muller-Schupp style determinization construction that yields deterministic parity automata. Compared to previous constructions from [7] and [4], we encode less information in the states, and obtain a construction that has the same worst-case upper bound as the Safra style constructions.
- We extend our Muller-Schupp style construction by introducing a degree of freedom in the choice of the successor states. This freedom can be used to make the construction correspond to Safra's construction as presented in [11] and [14]. We therefore obtain a construction that unifies the approaches of Safra and Muller-Schupp in one general construction. Furthermore, the freedom in the choice of the successors of transitions also yields new ways of obtaining deterministic parity automata, and can be used in implementations as a heuristic to reduce the state space of the resulting automaton.

This work is organized as follows. After introducing the basic notations in Section 2, we present the new variant of the Muller-Schupp construction in Section 3, and then briefly review Safra's construction in Section 4. We explain the structural relationship between

those two constructions in Section 5, and finally introduce our generalized construction as a simple extension of the presented Muller-Schupp construction in Section 6. In Section 7 we discuss and conclude.

2 Preliminaries

First we briefly review basic definitions concerning ω-automata and ω-languages. If Σ is a finite alphabet, then Σ^ω is the set of all infinite *words* $w = w_0 w_1 \dots$ with $w_i \in \Sigma$. For $w \in \Sigma^\omega$ we denote by $w(i)$ the i-th symbol w_i. For convenience, we write $[n]$ for the set of natural numbers $\{1, \dots, n\}$. A *Büchi automaton* $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, Q_0, F)$, where Q is a finite set of states, Σ a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $Q_0, F \subseteq Q$ are the sets of initial and accepting states, respectively. When Q is understood and $X \subseteq Q$, then $\overline{X} := Q \setminus X$. We write $\Delta(p, x) := \{q \mid (p, x, q) \in \Delta\}$ to denote the set of *successors* of p on symbol x and $\Delta(P, x)$ for $\bigcup_{p \in P} \Delta(p, x)$. A *run* of an automaton on a word $w \in \Sigma^\omega$ is an infinite sequence of states $q_0, q_1, \dots$ starting in some $q_0 \in Q_0$ such that $(q_i, w(i), q_{i+1}) \in \Delta$ for all $i \geq 0$. An automaton is *deterministic* if $|Q_0| = 1$ and $|\Delta(p, x)| \leq 1$ for all $p \in Q, x \in \Sigma$, and *non-deterministic* otherwise. In this work, we assume Büchi automata to be non-deterministic and refer to them as NBA. A *transition-based deterministic parity automaton* (TDPA) is a deterministic automaton $(Q, \Sigma, \Delta, Q_0, c)$ where instead of $F \subseteq Q$ there is a *priority function* $c : \Delta \to \mathbb{N}$ assigning a natural number to each transition.

A run of an NBA is *accepting* if it contains infinitely many accepting states. A run of a TDPA is accepting if the smallest priority that appears infinitely often on transitions along the run is even. An automaton $\mathcal{A}$ *accepts* $w \in \Sigma^\omega$ if there exists an accepting run on w, and the language $L(\mathcal{A}) \subseteq \Sigma^\omega$ *recognized* by $\mathcal{A}$ is the set of all accepted words. To avoid confusion, we sometimes refer to states of TDPA that we construct as *macrostates* to distinguish them from the states of the underlying Büchi automaton.

3 A Simplified Muller-Schupp Construction

The essential idea for determinization using the Muller-Schupp construction is the following: given some Büchi automaton $\mathcal{A}$ and input word w, the resulting deterministic automaton conceptually traverses a specific run-tree of $\mathcal{A}$ on w, called reduced split-tree in [7], and tracks enough information to decide whether an infinite path with a specific shape exists in this tree. Such a path is known to exist if and only if w is accepted by $\mathcal{A}$. The construction presented in [7] uses a structure called contraction trees in order to track the relevant information. This has been simplified in [4] to macrostates that consist of an ordered tuple of disjoint sets of Büchi states, and two preorders over the states appearing in the tuple.

In this section, we further simplify the structure of the macrostates for the deterministic automaton to ordered tuples of disjoint sets of Büchi states, and a single additional linear order on these sets (formally expressed as a ranking function that assigns to each set a natural number). This also results in a relatively simple transition function on the macrostates.

The *reduced split-tree* $t^{rs}(\mathcal{A}, w)$ for NBA $\mathcal{A}$ and word $w \in \Sigma^\omega$ is an ordered infinite tree in which the nodes are labelled by state-sets, and each node has at most two successors. Formally, it is constructed as follows. The first level of the tree consists of the root node labelled by the initial states Q_0. To construct level $i + 1$ from level i, for each node at level i labelled by set S of states, let the *left child* of S be labelled by $\Delta(S, w(i)) \cap F$ and the *right child* by $\Delta(S, w(i)) \cap \overline{F}$, i.e., accepting and non-accepting successor states are separated. Then keep only the leftmost (wrt. the natural ordering of neighbors) occurrence of each state

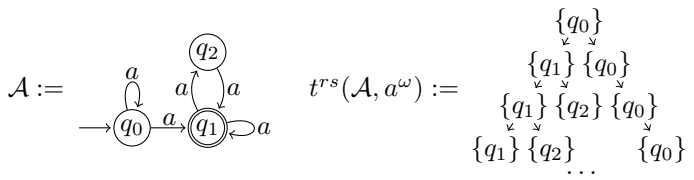

Figure 1 Example of a reduced split-tree $t^{rs}(\mathcal{A}, a^\omega)$ of an NBA $\mathcal{A}$. It has an infinite path representing the run q_0^ω and a left-path representing the run $q_0 q_1^\omega$, from which finite paths $q_0 q_1^* q_2$ branch off.

in the level and finally remove nodes labelled by $\emptyset$. Clearly, because of the normalization, the number of nodes on each level can be at most $|Q|$. An example of a reduced split-tree is shown in Figure 1. We call an infinite path in the tree that takes the left branch infinitely often a *left-path*. Reduced split-trees have the following useful property:

▶ **Lemma 1** ([7], Lemma 2). *$\mathcal{A}$ accepts $w \iff t^{rs}(\mathcal{A}, w)$ has a left-path.*

In the following, we identify nodes in the same level with their label sets. To obtain a deterministic automaton, we augment the nodes of the reduced split-tree with number tokens that we call *(age-)ranks*, which are used to infer a left-path.

The new macrostates in the deterministic automaton represent levels of reduced split-trees and consist of a tuple of disjoint non-empty sets $t := (S_1, S_2, \ldots, S_n)$ equipped with a bijection $\alpha : [n] \to [n]$ satisfying $\alpha(n) = 1$, which assigns to each set S_i the rank $\alpha(i)$. We call a pair (α, t) that satisfies these constraints *ranked slice* and we call it *pre-slice*, if t contains empty sets or α is not a bijection. Notice that all macrostates are ranked slices, whereas pre-slices occur only during intermediate steps. We introduce the following useful notations to work with ranked slices. Let $|t| := n$ and $Q_t := \bigcup_{i=1}^{|t|} S_i$. The function $\mathsf{idx} : Q_t \to [|t|]$ maps each state $q \in Q_t$ to the tuple index i such that $q \in S_i$, and by $\alpha(q)$ we denote $\alpha(\mathsf{idx}(q))$ for $q \in Q_t$.

When reading symbol $x \in \Sigma$ in macrostate (α, t), the successor macrostate (α', t') is obtained by a sequence of successive operations step, prune and normalize, where, roughly,

- step interprets t as nodes on a reduced split-tree level and calculates the next level sets,
- prune removes the empty sets produced by step, reassigning ranks in a specific way, and
- normalize just turns the ranking function obtained after prune into a bijection again.

Below, we formally define these operations (step and prune are illustrated in Figure 2).

First we describe step, which constructs the next level of the reduced split-tree and passes each existing rank on to the respective right child. Let

$$\Delta_t(q, x) := \Delta(q, x) \setminus \Delta\Big(\bigcup_{i=1}^{\mathsf{idx}(q)-1} S_i, x\Big),$$

restricting for each state $q \in Q_t$ the successors to only those which are not reached by some other state located in a set to the left of q. Then, for each node S_i let $\hat{S}_{2i-1} := \Delta_t(S_i, x) \cap F$ be the *left child* and $\hat{S}_{2i} := \Delta_t(S_i, x) \cap \overline{F}$ the *right child*, containing the accepting and non-accepting normalized successors, respectively. Let $\hat{\alpha}(2i) := \alpha(i)$ and $\hat{\alpha}(2i-1) := n+1$, i.e., the right children inherit the rank of the parent and the left children all get the same new maximal rank $n+1$, resulting in a pre-slice $(\hat{\alpha}, \hat{t})$.

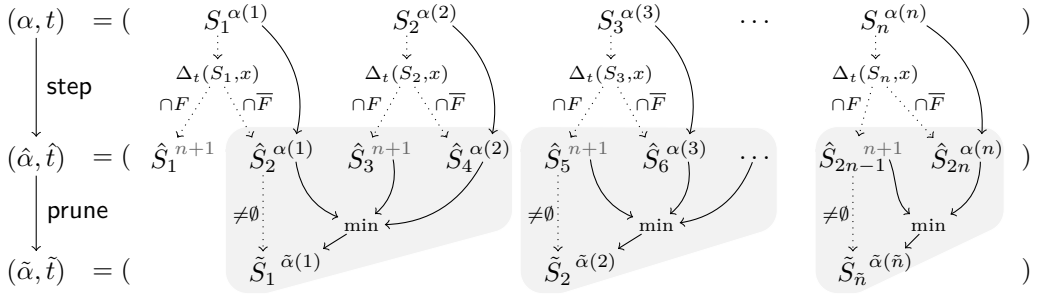

Figure 2 Abstract illustration of step and prune in a Muller-Schupp transition on some $x \in \Sigma$. The superscripts represent the assigned ranks. First, step calculates the normalized successors, separating accepting from non-accepting states and passing the parent rank on to the right child. In the illustration, we assume that the sets $\hat{S}_i \neq \emptyset$ for $i \in \{2, 5, 2n-1\}$, i.e., $x_1 = 2, x_2 = 5, x_{\tilde{n}} = x_3 = 2n-1$. Then prune keeps sets at these positions for the resulting tuple $\tilde{t}$, and $\tilde{\alpha}$ is obtained by taking the minimum of the ranks given by $\hat{\alpha}$ in the ranges spanning from one x_i up to the position before x_{i+1}. Finally $t' := \tilde{t}$ and $\tilde{\alpha}$ is normalized to α', while preserving strict ordering between positions wrt. $\tilde{\alpha}$. The dotted edges connect parent sets (in the top row) and resulting left/right children sets (bottom row) in the conceptual reduced split-tree, the solid edges show the movements of the rank values assigned to the sets.

Intuitively, in the prune operation, all ranks that mark empty sets after step are relocated onto the closest non-empty set to the left (or removed, if no such set exists). When multiple ranks occupy the same set, then the smallest one is preserved. Ranks that moved to the left in this way and are not removed, indicate a good (green) event, whereas ranks which were removed indicate a bad (red) event.

Formally, let $x_1 < x_2 < \ldots < x_{\tilde{n}}$ be the increasing sequence of all indices such that $\hat{S}_{x_j} \neq \emptyset$. Then prune returns $(\tilde{\alpha}, \tilde{t})$ with the tuple $\tilde{t} := (\hat{S}_{x_1}, \ldots, \hat{S}_{x_{\tilde{n}}})$ without empty sets, where $\tilde{\alpha}$ is defined as $\tilde{\alpha}(i) := \min\{\hat{\alpha}(j) \mid x_i \leq j < x_{i+1}\}$ with $x_{\tilde{n}+1} := |\hat{t}| + 1$.

The set of *green* ranks is given by $G := \mathsf{img}(\tilde{\alpha}) \cap \{\hat{\alpha}(j) \mid \hat{S}_j = \emptyset\}$, where $\mathsf{img}(\tilde{\alpha})$ denotes the image of $\tilde{\alpha}$. These are the ranks that mark empty sets after step and are not removed by prune. The set of *red* ranks given by $R := \mathsf{img}(\hat{\alpha}) \setminus \mathsf{img}(\tilde{\alpha})$ contains the ranks that were not preserved during prune. The set of *active ranks* is $A := G \uplus R$. Let $k := \min A$ (or $k := |Q| + 1$ if $A = \emptyset$) denote the *dominating* rank of the transition, i.e., the smallest active rank. We define the priority p of the transition as $2k$ if $k \in G$ and $2k - 1$ otherwise.

The function $\tilde{\alpha}$ might assign the same rank to several sets, and it might have gaps (unused rank values between used ones). So finally, normalize returns (α', t') with $t' := \tilde{t}$ and a final bijective ranking function $\alpha' : [|t'|] \to [|t'|]$ such that $\tilde{\alpha}(j) < \tilde{\alpha}(k) \Rightarrow \alpha'(j) < \alpha'(k)$ for all $j, k \in \{1, \ldots, |t'|\}$, i.e., a total order which is compatible with the preorder induced by $\tilde{\alpha}$. If there are several such ranking functions α', then any of these works.

A TDPA $\mathcal{B}$ is obtained by taking the initial state (α_0, t_0) with $t_0 := (Q_0), \alpha_0(1) := 1$ and a transition function that picks for each state a valid successor that satisfies the description above, and assigns the corresponding priority p to the edge. Observe that by construction, the sequence of states visited along some word $w \in \Sigma^\omega$ from the initial state represents exactly the levels of $t^{rs}(\mathcal{A}, w)$, marked with ranks.

▶ **Theorem 2.** *For a given NBA with n states, the TDPA obtained by the Muller-Schupp construction accepts the same language as the NBA, and its number of states is in $\mathcal{O}(n!^2)$.*

The correctness follows from the correctness of the generalized construction presented in Section 6. The claim on the state complexity directly follows from the upper bound given in [14, Proposition 2], and the bijection between the set of ranked slices and the set of ranked Safra trees presented in Section 5.

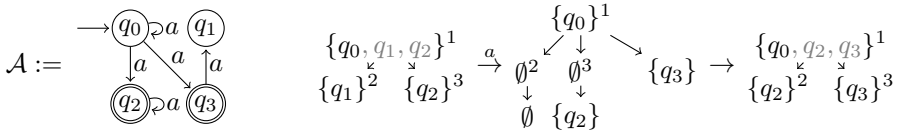

Figure 3 Example of a Safra-tree transition on letter a, based on NBA $\mathcal{A}$. The LIR position of nodes is depicted as superscript of the sets. The "redundant" states that are implicit in our definition are depicted in gray in the initial and resulting tree. In the intermediate step, the tree is depicted after calculating and pruning successor state sets. In the final tree the remaining actions are performed and LIR positions are updated. The transition has a red event for LIR position 2 and a green event for position 3. Because of the removal of the node at position 2 in the LIR, the node that originally was at position 3 moved up, whereas the fresh node labelled by $\{q_3\}$ comes last.

4 Sketch of the Safra Construction

In this section, we roughly illustrate the used structures and operations of the Safra construction along the lines of [11, 14], so that we can demonstrate its relationship with the Muller-Schupp construction in the next section. As before, $\mathcal{A}$ is an NBA with the usual components.

A *Safra tree* is a finite ordered tree with non-empty state-sets as labels. Usually, it is required that a parent is labelled by a strict superset of all states in its subtree and siblings are labelled by pairwise disjoint sets. We use the equivalent requirement that *all* labels in the tree are pairwise disjoint, i.e., refrain from listing states in the parent label which are already present in some descendant. One can easily reconstruct the "full" label set of a node wrt. the classical definition by taking the union of all the labels in its subtree. To obtain parity automata, each node of the Safra tree is associated with a number from $\{1, \ldots, n\}$, where n is the number of nodes in the Safra tree [11]. These numbers satisfy the property that parent nodes have smaller numbers than their children, and a node has a smaller number than its right sibling. The numbers correspond to the ranks that we use in Section 3, and we therefore refer to Safra trees in combination with these numbers as *ranked Safra trees*. Two ranked Safra trees are shown in Figure 3 (and an intermediate tree in the middle).

In [14], a slightly different representation is used based on a *later introduction record* (LIR), which just lists the tree nodes in their introduction order, i.e., nodes appear in this list after parents and older siblings (in this representation, nodes have canonical names depending only on their position in the tree). Safra trees with LIR directly correspond to ranked Safra trees by annotating each tree node with its position in the LIR.

A transition on symbol $x \in \Sigma$ is constructed as follows (see Figure 3 for an example). First, for each label set S, the set $S' := \Delta(S, x)$ of successor states is calculated. After this, each node gets a fresh right-most child, and the accepting states in S', that is $S' \cap F$, are moved into the label of this child. Then, disjointness is ensured by keeping of each state only the copy which is located at the deepest node along the leftmost branch where that state occurs (this stage is represented by the middle tree in Figure 3). If now some internal node has an empty label, but a non-empty subtree (a good event for the node), its subtree is collapsed into a single node by removing all descendants and moving the states in their labels into the parent label. Finally, all remaining sets that are labelled by $\emptyset$ are removed (being removed is a bad event for a node). In the following, we refer to good and bad events as *green* and *red*, respectively. The priority for the transition is derived from the green and red events, which are associated with the relative position of the corresponding nodes in the LIR. The LIR for the new tree is obtained by deleting removed nodes from the LIR and appending fresh nodes that remain in the resulting tree in arbitrary order.

5 From Safra-trees to ranked slices and back

In this section we state the key observation that was the starting point of this work: there is a bijection between the set of ranked slices and the set of ranked Safra trees. From a ranked Safra tree, one obtains the ranked slice by simply listing the nodes of the Safra tree by a depth-first post-order traversal (i.e., a parent processed after all its children). We formalize this relationship below, and then explain that the transitions defined in the Muller-Schupp construction and in the Safra construction are very similar, which then leads to the unified construction.

Let (α, t) be a ranked slice with $t = (S_1, \ldots, S_n)$. The tuple index of the *parent* of S_i is the closest index to the right of i that has a smaller rank and is formally defined as $\uparrow(i) := \min_{i<k\leq n}\{k \mid \alpha(k) < \alpha(i)\}$. As we require by definition of ranked slices that the rightmost position in the tuple always has rank 1, this is the only position in the tuple for which the parent is undefined. The ordered tree induced by $\uparrow$, with siblings in tuple index order, is called the *rank tree* of (α, t). The tuple index of the *left subtree boundary* of S_i is the closest index to the left with a smaller rank, and is denoted by $\leftarrow(i) := \max_{1\leq k<i}\{k \mid \alpha(k) < \alpha(i)\}$ or 0 if no such index exists. It points to either the direct left sibling of i, or the left sibling of the closest ancestor, if one exists. Effectively, $\leftarrow(i)$ is the closest neighbor to the left which is not a descendant of i. As children by definition are always to the left of their parents, every node at indices $\leftarrow(i)+1, \ldots, i$ is in the subtree of i.

For an example, consider the tuple $(\{q_3\}^4, \{q_1\}^2, \{q_2\}^3, \{q_0\}^1)$, where the superscripts denote the assigned rank (e.g., $\alpha(1) = 4$). The rightmost position 4 of the tuple is the root of the tree. For the positions 2 and 3, which have rank 2 and 3 respectively, the next position to the right with a smaller rank is in both cases position 4, i.e., $\uparrow(2) = \uparrow(3) = 4$. Finally, position 1 in the tuple has position 2 as parent, i.e., $\uparrow(1) = 2$. The discussed tuple is depicted with the parent edges at the bottom right of Figure 4. There is also one non-trivial left subtree boundary in this tuple, assigned by $\leftarrow(3) = 2$, i.e. index 2 is not in the subtree of index 3, and in this case is an actual left sibling of index 3.

We use the notation $\uparrow_\alpha := \alpha \circ \uparrow \circ \alpha^{-1}$ to denote the parent rank of another rank directly, without mentioning the indices in the tuple. In the previous example, we have $\uparrow_\alpha(4) = 2$, and $\uparrow_\alpha(2) = \uparrow_\alpha(3) = 1$. We identify the age-ranks $\alpha(i)$ as nodes of the tree, while each set S_i determines the label of the node $\alpha(i)$, called *hosted set*. We write $S_i^{\downarrow} := \bigcup_{k=\leftarrow(i)+1}^{i} S_k$ for the *subtree set* of node $\alpha(i)$.

▶ **Definition 3.** *Let* safra2slice *be the mapping which takes a ranked Safra tree and returns* (α, t)*, with* $t := (S_1, \ldots, S_n)$ *being the label sets of the nodes in depth-first post-order (i.e., a parent processed after all its children) traversal order and ranking* α *defined by the ranks of the corresponding Safra tree nodes.*

Let slice2safra *be the mapping which takes a ranked slice* (α, t) *and returns the ranked Safra tree given by the rank tree of* (α, t)*, i.e. the tree structure defined by* $\uparrow$ *and the ordering of siblings given by the order of the corresponding sets in* t*.*

It is easy to see from the definitions that safra2slice and slice2safra are injective and return a valid ranked slice and ranked Safra tree, respectively. This implies that there exists a bijection between the sets of ranked Safra trees and ranked slices. It is also not very hard to see that the following holds:

▶ **Lemma 4.** safra2slice *and* slice2safra *are inverses of each other and provide a bijection between ranked Safra trees and ranked slices.*

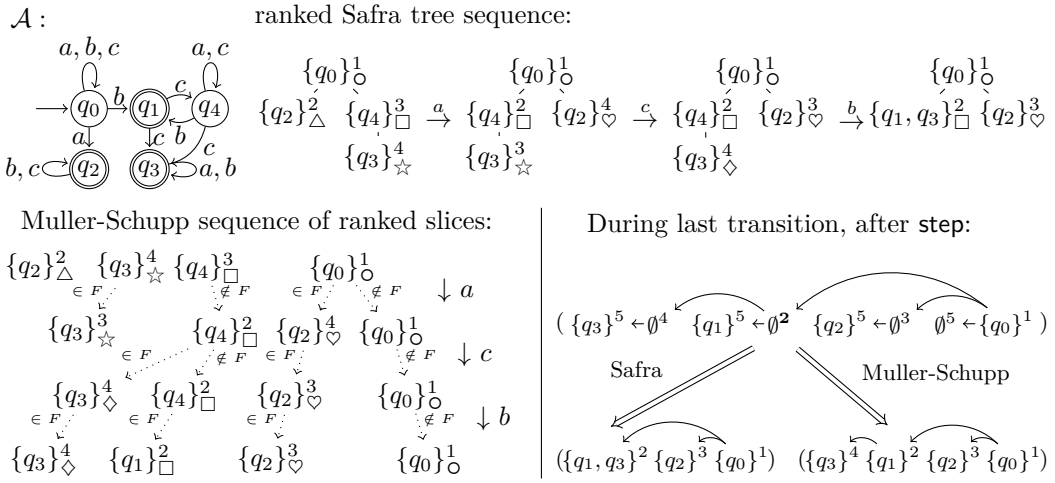

Figure 4 Transitions based on NBA $\mathcal{A}$ using both constructions. The superscripts denote the ranks of tree nodes / sets in the slice tuple. The subscripts are added for illustration purposes and conceptually track nodes throughout time, i.e., the same symbol marks the "same" node at different times. The algorithms agree on all but the last transition, where they differ due to different handling of green nodes/ranks, in this case rank 2 that marks an empty set after calculating and splitting the successors (illustrated on the bottom right). In the Muller-Schupp case, the rank is moved left during prune, resulting in a child being pulled into the parent in the rank tree, whereas in the Safra construction the whole subtree is collapsed. The solid edges between sets depict the rank tree induced by $\uparrow$, dotted edges depict the edges in the conceptual split-tree. In the bottom right the slices are shown together with their tree interpretation.

As we have established that both constructions, Muller-Schupp and Safra, operate on essentially the same structures, from now on we talk about ranked slices and trees interchangeably. Using this relationship, one can take the same tree/slice and apply both the successor calculation of the Safra construction and of the Muller-Schupp construction to it. What one first notices, is that the resulting tree/slice is very similar or equal in many cases. This is owed to the fact that most operations in one construction have an equivalent operation in the other, just formulated for the other representation.

For example, moving accepting successor states into a fresh child node in Safra's construction corresponds to splitting accepting successors from non-accepting ones during step in the Muller-Schupp construction, as in the successor tuple the new child (in the conceptual split-tree) gets a fresh, larger rank and by definition becomes the rightmost child in the rank tree of the resulting new slice. The normalization steps that make the successor sets pairwise disjoint also yield the same results. The ranks of nodes with green events in the Safra construction coincide with ranks of sets that signal green in the ranked slices, and ranks of Safra nodes with red events with ranks of sets that signal red. The removal of empty sets by prune and renumbering the ranks with normalize is the same as the removal of the corresponding nodes in the Safra tree and updating the LIR, i.e., the ranks of Safra nodes.

In fact, *the only difference* between the constructions is what happens with a tree node in case of a green event. Recall that in Safra's construction, the whole subtree of a green node is collapsed to a single node. In the Muller-Schupp construction, the green ranks are those that end up on an empty set after step, and that survive the prune operation, in which the ranks are moved to the next non-empty set to the left, and only the minimal ones are kept on each non-empty set. In the view of ranked trees, this corresponds to a green node absorbing its rightmost, uppermost child node into it, while keeping the rest of the subtree unchanged. See Figure 4 for an illustration.

After observing that both constructions differ in only a minor step and noticing that both yield correct (but possibly different) automata, it becomes apparent that the exact step performed for green events is not essential and there must be a more general mechanism to uncover. The construction we present in Section 6 results from this line of thought.

On the practical side, it is worth mentioning that the cost of switching between the representations using the presented bijection is negligible – the traversal of a ranked Safra tree to obtain a ranked slice is obviously possible in linear time. For the other direction there also exists a simple linear time algorithm that calculates the parent and left subtree boundary relation from the ranking α.

6 The unified construction

In this section, we present a construction that builds on the Muller-Schupp construction from Section 3, and unifies it with Safra's construction by adding another operation, called merge, between prune and normalize: $(\alpha, t) \xrightarrow{\text{step}} (\hat{\alpha}, \hat{t}) \xrightarrow{\text{prune}} (\tilde{\alpha}, \tilde{t}) \xrightarrow{\text{merge}} (\check{\alpha}, \check{t}) \xrightarrow{\text{normalize}} (\alpha', t')$.

This new operation is nondeterministic, and can be instantiated in different ways. In particular, it can be instantiated trivially and thus corresponds to the Muller-Schupp construction, and it can be used to emulate the Safra construction.

We first describe the idea of merge, and then give a formal definition. Assume that, after step and prune have been applied to some ranked slice (α, t), we have the pre-slice $(\tilde{\alpha}, \tilde{t})$, and the dominating (minimal active) rank k (determined by prune, see Section 3). Then merge can collapse groups of neighbouring sets in the tuple, and preserves the minimum rank from each collapsed range, similar to prune. In contrast to prune, which "merges" one non-empty set with multiple empty sets in a deterministic manner, merge may actually take the union of multiple adjacent non-empty sets, depending on the ranks currently assigned to them.

The non-overlapping intervals of sets that are collapsed together are not uniquely determined in general. They only have to satisfy the constraints that no sets with rank smaller than the dominating rank k are merged with anything else, and that the set with rank k is not merged with anything to the right of it. These constraints are important for the correctness, and ensure that in the ranked Safra tree perspective, the nodes with rank smaller than k do not change, and that the node with the dominating rank k is not merged with sets outside of its subtree.

Formally, merge returns a pre-slice $(\check{\alpha}, \check{t})$ obtained in the following way (see Figure 5 for an illustration). Let $I_1, I_2, \ldots, I_{n'}$ be a sequence of sets partitioning the set of indices $\{1, \ldots, \tilde{n}\}$ in $\tilde{t}$ into adjacent groups, i.e., $\min I_1 = 1$, $\max I_{n'} = \tilde{n}$ and for all $j > 1$ we have $\min I_j = \max I_{j-1} + 1$. This grouping should satisfy the following property for all $1 \leq j \leq n'$ and $l \in I_j$: if $\tilde{\alpha}(l) < k$, then $|I_j| = 1$, and if $\tilde{\alpha}(l) = k$, then $\max I_j = l$. Then the pre-slice $(\check{\alpha}, \check{t})$ is defined by the sets $\check{S}_i := \bigcup_{j \in I_i} \tilde{S}_j$ and the ranking function $\check{\alpha}(i) := \min\{\tilde{\alpha}(j) \mid j \in I_i\}$ for all $i \in \{1, \ldots, n'\}$, i.e., for each interval, the union of the sets and the smallest rank is taken.

As in the Muller-Schupp construction, normalize is applied to $(\check{\alpha}, \check{t})$ to obtain the successor macrostate (α', t'). This extended transition relation is used to obtain the transition-based deterministic parity automaton, as before.

An example showing how the choice of different merge strategies leads to different successor states is illustrated in Figure 6. Observe that we can recover the Muller-Schupp construction by using the identity function for merge, or in other words, putting each index into its own interval, which is the finest partitioning of indices that satisfies the requirements on merge. On the other hand, we can also take the coarsest compatible partitioning, i.e., minimize the number of intervals. We call this kind of update *maximal collapse*.

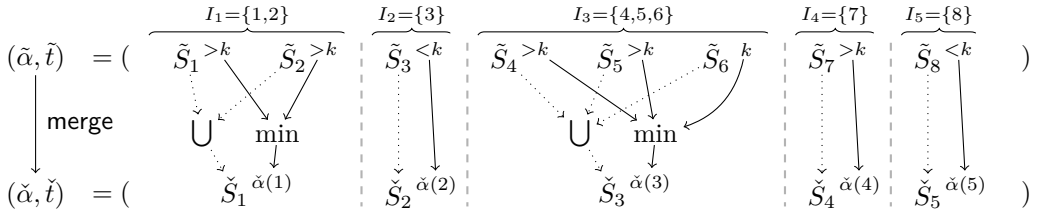

Figure 5 Illustration of the general merge operation that comes after prune and before normalize, with the minimal active rank k and ranks depicted as set superscripts. The illustrated intervals are the coarsest partitioning of indices in $\tilde{t}$ satisfying the constraints.

We can emulate a Safra-update by imposing some additional constraints on the intervals, ensuring that only the complete subtrees of nodes with green ranks are merged. More concretely, we require that intervals that are not singletons span exactly the nodes of the complete subtree that is rooted in a green rank in the view of the slice as ranked Safra tree. Note that for an index ℓ in the tuple, the subtree of the corresponding node in the ranked Safra tree corresponds to the interval that starts one step right of the left subtree boundary of ℓ, and ends in ℓ, that is, the interval $\leftarrow(l)+1, \ldots, \ell$. Thus, for imitating the Safra merge rule, the intervals $I_1, I_2, \ldots, I_{n'}$ from merge are the unique smallest intervals satisfying

$$\forall i \in [n'], l \in I_i : \quad \tilde{\alpha}(l) \in G \implies \leftarrow(l)+1 \in I_i \quad \text{(complete subtrees collapsed)}.$$

▶ **Proposition 5.** *The operation* merge *can be instantiated such that the transitions of the constructed TDPA correspond to the transitions of the Muller-Schupp construction or to the transitions of the Safra construction.*

Notice that for all merge rules except for the Muller-Schupp update, the relationship of ranked slices and consecutive levels of the reduced split-tree (see Section 3) breaks down. One can, however, reflect the merges also in the reduced split-tree by doing the merges of the corresponding sets on each level, which leads to an acyclic graph instead of a tree. This view is helpful in the correctness proof of the construction.

▶ **Theorem 6.** *Let $\mathcal{A}$ be an NBA. Then a deterministic parity automaton $\mathcal{B}$, obtained by the described determinization construction, has at most $\mathcal{O}(n!^2)$ states and recognizes the same language as $\mathcal{A}$.*

The upper bound holds because the same macrostates are used as in the presented Muller-Schupp construction in Section 3. The correctness can be shown by a refinement of the original correctness proof of the Safra construction [13].

7 Discussion and Conclusion

We have presented a new variant of the Muller-Schupp construction for determinization of Büchi automata into parity automata, reducing the information stored in the macrostates to ordered tuples of disjoint sets annotated with ranks. These ranked slices are in bijection with ranked Safra trees, which leads to a general construction that can emulate the Muller-Schupp construction and the Safra-construction. This answers, in some sense, the question from [4] on the relation between the two types of constructions.

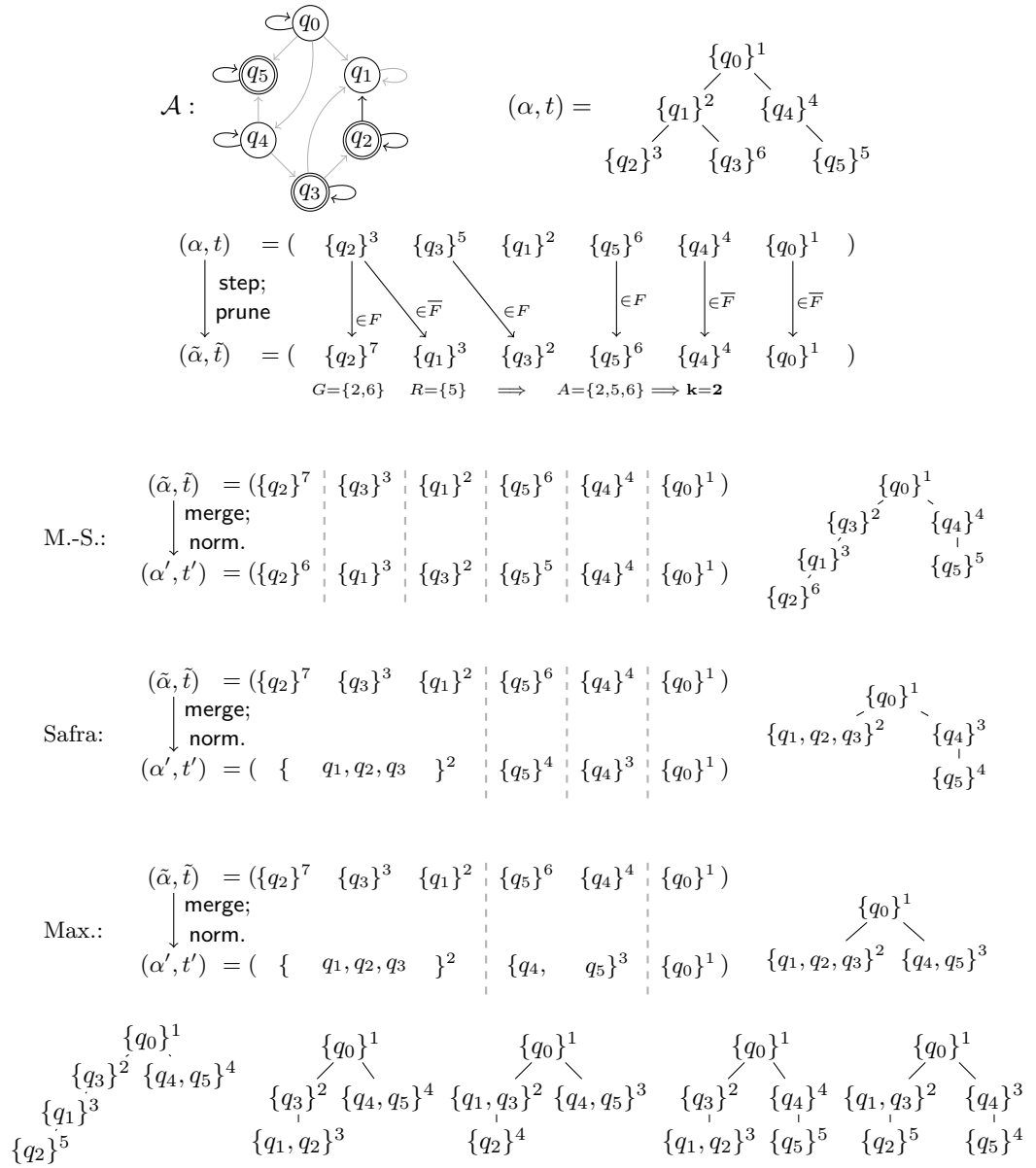

Figure 6 $\mathcal{A}$ illustrates the relevant part of an NBA during a transition on some symbol $x \in \Sigma$, that is, the arrows correspond to the x-transitions of $\mathcal{A}$. The gray edges are the ones pruned in the reduced transition relation Δ_t. The current macrostate (α, t) is represented as the rank tree to the right of $\mathcal{A}$, and as ranked slice below $\mathcal{A}$. The step and prune operations (see Fig. 2 for details) result in ranks 1,3 and 4 being passed down along the right child. Ranks 2 and 6 are moved to the left and hence are green. Rank 5 is overwritten by 2 and hence is red. Rank 7 is a fresh rank which is larger than the others. The dominating rank k is 2. The choice of different merge intervals (as shown in Fig. 5) results in different successors. The successors for the three discussed variants, Muller-Schupp, Safra, and maximal collapse, are shown as rank trees on the right. The 5 other permitted successors are depicted at the bottom.

In general, one can obtain many different valid deterministic automata by choosing different deterministic transition functions that are compatible with the described successor relation. One can also imagine this as constructing a non-deterministic automaton with *all* permitted successors, and then pruning the edges arbitrarily, while preserving for each state only one outgoing transition for each symbol, to "carve" out a valid deterministic automaton.

This non-determinism comes from two sources. One degree of freedom in our construction comes from the different ways of assigning ranks (to new nodes, and when closing gaps resulting from deleted ranks). This freedom is already mentioned in [14]. But here the flexibility is just in the choice of the specific permutation, which still describes structurally the same tree in any case. The novel and in our opinion powerful degree of freedom in our construction is the possibility for different valid merge operations, which allows for a vastly larger pool of possible successors, as the results may describe structurally different trees. Furthermore, the smaller the smallest active rank, the more different a permitted successor may look like.

We have explicitly mentioned the merge strategies that lead to the Muller-Schupp and Safra constructions, and also have mentioned a third strategy, the maximal collapse rule that merges as many sets as possible (as shown in e.g. Figure 6). We also want to point out that, while fixing one such merge-rule for the whole construction is the simplest implementation, the construction permits using *any* valid successor without the need to disambiguate the merge operation beforehand, i.e., picking the successor of a state from the set of permitted ones is a *local* choice. One may think of schemes where the successor is chosen dynamically, depending on the input or already computed information. For example, one can check whether a valid successor has already been constructed, and only add a new state according to a fixed policy if this is not the case. We have already implemented a prototype making use of such an optimization (among others) with encouraging results.

We also want to point out that the presented construction works equally well with transition-based Büchi automata as input, in which case the step operation separates states which are reached by at least one accepting transition from those that are not. One can easily verify that this does not impact the reasoning in the proofs.

It is also possible to adapt the construction to yield Rabin automata, such that the corresponding Safra construction as presented in [14] is subsumed. In this setting, however, the presentation of macrostates as ordered tuples of sets is less natural. Furthermore, in this setting the merges of sets needs to be restricted to subtrees of green nodes, because there is no total order of importance of nodes as provided by the ranks.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

2 J Richard Büchi. On a decision method in restricted second order arithmetic. In *Studies in Logic and the Foundations of Mathematics*, volume 44, pages 1–11. Elsevier, 1966.

3 Thomas Colcombet and Konrad Zdanowski. A Tight Lower Bound for Determinization of Transition Labeled Büchi Automata. In *ICALP 2009*, volume 5556 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2009.

4 Seth Fogarty, Orna Kupferman, Moshe Y Vardi, and Thomas Wilke. Profile trees for Büchi word automata, with application to determinization. *Information and Computation*, 245:136–151, 2015.

5 Paul Gastin and Denis Oddoux. Fast LTL to Büchi Automata Translation. In *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, pages 53–65, 2001. doi:10.1007/3-540-44585-4_6.

6 Dimitra Giannakopoulou and Klaus Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *16th IEEE International Conference on Automated Software Engineering (ASE 2001), 26-29 November 2001, Coronado Island, San Diego, CA, USA*, pages 412–416, 2001. doi:10.1109/ASE.2001.989841.

7 Detlef Kähler and Thomas Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP 2008*, pages 724–735. Springer, 2008.

8 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966.

9 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.

10 David E Muller and Paul E Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

11 Nir Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 255–264. IEEE, 2006.

12 Roman R Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundamenta Informaticae*, 119(3-4):393–406, 2012.

13 Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, 1988.*, pages 319–327. IEEE, 1988.

14 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FOSSACS*, pages 167–181. Springer, 2009.

15 Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 248–263, 2000. doi:10.1007/10722167_21.

16 Wolfgang Thomas. Languages, Automata, and Logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

17 Wolfgang Thomas. Church's Problem and a Tour through Automata Theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 635–655. Springer, 2008. doi:10.1007/978-3-540-78127-1.

18 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and automata - history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.