Complementation, Disambiguation, and Determinization of Büchi Automata Unified

Detlef Kähler and Thomas Wilke

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany

Abstract. We present a uniform framework for (1) complementing Büchi automata, (2) turning Büchi automata into equivalent unambiguous Büchi automata, and (3) turning Büchi automata into equivalent deterministic automata. We present the first solution to (2) which does not make use of McNaughton's theorem (determinization) and an intuitive and conceptually simple solution to (3).

Our results are based on Muller and Schupp's procedure for turning alternating tree automata into non-deterministic ones.

1 Introduction

As Büchi automata play a crucial role when it comes to designing decision procedures for mathematical theories (such as S1S and S2S) and solving verification problems, see, for instance, [3,16,21], much work has been devoted to fundamental constructions for Büchi automata, in particular, numerous complementation and determinization[1] procedures have been suggested and lower bounds have been proved, see [3,18,8,19,9,7,22,5] and [13,11,4,12,17,14,15], respectively. The known constructions for complementation have nothing in common with the known constructions for determinization, unless one considers the following detour: To complement a given Büchi automaton, one first determinizes it, then complements the deterministic automaton (which is, in general, straightforward), and finally reconverts the complemented deterministic automaton into a Büchi automaton. This, however, yields much more complex constructions than the ones that aim directly at complementation, not only on an informal level, but also on a formal one: When the number of states of the resulting automata are used for comparing them, then direct complementation constructions are by far superior. For turning a Büchi automaton into an equivalent unambiguous Büchi automaton, the situation is worse. The only disambiguation procedure that has been published [2] goes via determinization and therefore is complex.

In this paper, we establish a uniform framework for complementation, disambiguation, and determinization, where complementation and disambiguation avoid determinization. This framework heavily builds on the work by Muller and Schupp described in [14], where they present a procedure that turns alternating tree automata into nondeterministic ones and point out that, as a byproduct, one obtains a determinization procedure for Büchi (word) automata. From their work we identify a characteristic

[1] Recall that there are Büchi automata for which no equivalent deterministic Büchi automata exist. Determinization in the context of Büchi automata therefore means to turn a Büchi automaton into an equivalent deterministic automaton with an appropriate acceptance condition.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 724–735, 2008.

structure which is associated with a given Büchi automaton and a given word over the same alphabet—the so-called skeleton. We first show that this tree can be produced level by level by an unambiguous Büchi automaton—the so-called slice automaton. We then show that by forming a product of the slice automaton with small Büchi automata (i) an equivalent unambiguous Büchi automaton and (ii) an unambiguous Büchi automaton for the complement can be obtained. The number of states of the complement automaton obtained is at most $4(3n)^n$, which is comparable to other constructions. The upper bound for the size of the unambiguous automaton is the same, which is much smaller than anything one obtains by first determinizing the given automaton, in particular, it is much smaller than the automaton described in [2]. It should be noted that after simple optimizations, the tighter analysis from [5] applies in both cases, (i) and (ii).

We also show how the branching structure of the skeleton can be approximated by a conceptually simple deterministic automaton, which, by forming a product with a generic parity automaton (latest appearance record with hit), can be turned into an equivalent deterministic parity automaton.

Related work. The first comprehensive description of Muller and Schupp's determinization construction for Büchi word automata and a comparison with Safra's determinization construction was given by the first author in his diploma thesis [10]; further data on the relation between these two determinization constructions was presented in [1]. A preliminary version of the complementation construction presented here was described in [20]. The exposition in the present paper is more modular, in particular, [20] did not have the intermediate slice automaton, which is the basis for both complementation and disambiguation.

2 Basic Notation and Definitions

We use standard notation for alphabets and words. A *Büchi automaton* is a tuple $\mathscr{A} = (A, Q, q_I, \Delta, F)$ where A is a finite alphabet, Q is a finite state set, $q_I \in Q$ is an initial state, $\Delta \subseteq Q \times A \times Q$ is a transition relation, and $F \subseteq Q$ is a final state set. A run of $\mathscr{A}$ on a word $u \in A^\omega$ is a word $\rho \in Q^\omega$ satisfying $\rho(0) = q_I$ and $(\rho(i), u(i), \rho(i+1)) \in \Delta$ for all $i < \omega$. Such a run is *accepting* if there exist infinitely many i such that $\rho(i) \in F$. The set of all words accepted by $\mathscr{A}$ is denoted $L(\mathscr{A})$. A Büchi automaton $\mathscr{A}$ as above is *unambiguous* if for every $u \in A^\omega$ there is at most one accepting run of $\mathscr{A}$ on u.

Henceforth, we assume, without loss of generality, that every Büchi automaton is complete, that is, for every $q \in Q$, $a \in A$, there exists $q' \in Q$ such that $(q, a, q') \in \Delta$.

All trees in this paper are binary trees with ordered successors. One way to represent such a tree is by a prefix-closed non-empty subset of $\{0,1\}^*$. If t denotes such a tree and $v0 \in t$, then $v0$ is the left successor or 0-successor of v. Symmetrically, if $v1 \in t$, then $v1$ is the right successor or 1-successor of v. A vertex $v' \in t$ is a descendant of a vertex $v \in t$ if $v \leq_{\mathrm{prf}} v'$, where $\leq_{\mathrm{prf}}$ denotes the prefix order. The set of inner vertices and leafs of a tree t are denoted by $\mathrm{inner}(t)$ and $\mathrm{leafs}(t)$, respectively.

As usual, a tree t is called full binary tree if for every $v \in t$ either $\{v0, v1\} \cap t = \emptyset$ or $\{v0, v1\} \subseteq t$, that is, if every vertex is a leaf or has two successors.

A path through a tree t is a word $\pi \in t^+ \cup t^\omega$ such that $\pi(i+1) \in \{\pi(i)0, \pi(i)1\}$ for all i with $i + 1 < |\pi|$. A branch is a maximum path starting in the root, that is, starting

with ϵ, which denotes the empty word. By abuse of notation, when π is a path we will write $v \in \pi$ to denote that there exists $i < |\pi|$ such that $v = \pi(i)$.

Another way to represent a binary tree is as a tuple $T = (V, s_0, s_1)$ where V is an arbitrary set of vertices and $s_0 \colon V \to V$ and $s_1 \colon V \to V$ are partial functions describing the 0- and 1-successors of the vertices, respectively. The root of such a tree is denoted v_T. As the two representations of trees are interchangeable, in concrete circumstances we will use the representation which is most convenient. The first is referred to as *implicit notation* while the second is referred to as *explicit notation*.

A vertex v of a tree T as above is on level 0 if $v = v_T$, that is, if v is the root of T. It is on level $i + 1$ if it is a successor of a vertex on level i. The width of a tree is the supremum of the number of vertices on any level.

For a set L, an L-labeled tree in implicit notation is a function $t \colon V \to L$ where V is a tree in implicit notation as defined above. For every vertex $v \in V$, the value $t(v)$ is the label of v. By abuse of notation, we will write $v \in t$ instead of $v \in V$. An L-labeled tree in explicit notation is a tuple $T = (V, s_0, s_1, l)$ where (V, s_0, s_1) is a tree in explicit notation as defined above and $l \colon V \to L$ is the labeling function which assigns to each $v \in V$ its label $l(v)$. By abuse of notation, we will write $T(v)$ for $l(v)$. The i-th *slice* of an L-labeled tree is the word which is obtained by reading the labels of the vertices on level i from left to right, that is, it is a word over L.

3 Split Trees and Skeletons

In the rest of this paper, $\mathscr{A}$ denotes a fixed Büchi automaton as above with n states and u an infinite word over the same alphabet.

The split tree for u with respect to $\mathscr{A}$ is a tree which can be thought of as refining the power set construction. The root is labeled with the singleton set consisting of the initial state only. For any vertex on level i, we consider the set of states which can be reached from the states the vertex is labeled with by reading the letter at position i, split this set into final and non-final states, and label the left successor of the vertex with the final states and the right successor with the non-final states. If there are no final or non-final states the respective successor does not exist.

To describe this formally, we introduce more notation. For a set of states $Q' \subseteq Q$ and a letter $a \in A$, we define $\Delta(Q', a) = \{q \in Q \colon \exists q' (q' \in Q' \wedge (q', a, q) \in \Delta)\}$, and, similarly, $\Delta_F(Q', a) = \Delta(Q', a) \cap F$ and $\Delta_{\overline{F}}(Q', a) = \Delta(Q', a) \setminus F$.

The split tree for u with respect to $\mathscr{A}$, denoted t_u^{sp}, is a binary tree in implicit form with labels in 2^Q, inductively defined as follows. For the basis, $\epsilon \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(\epsilon) = \{q_I\}$. For the inductive step, assume $v \in t_u^{\mathrm{sp}}$ and let $Q' = t_u^{\mathrm{sp}}(v)$, $i = |v|$, and $a = u(i)$. Then:

- If $\Delta_F(Q', a) \neq \emptyset$, then $v0 \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(v0) = \Delta_F(Q', a)$.
- If $\Delta_{\overline{F}}(Q', a) \neq \emptyset$, then $v1 \in t_u^{\mathrm{sp}}$ and $t_u^{\mathrm{sp}}(v1) = \Delta_{\overline{F}}(Q', a)$.

If ρ is a run of $\mathscr{A}$ on u, then β inductively defined by $\beta(0) = \epsilon$ and $\beta(i+1) = \beta(i)0$ if $\rho(i+1) \in F$ and else $\beta(i+1) = \beta(i)1$ is a branch of t_u^{sp}. Conversely, using König's lemma, one can easily prove that for every infinite branch β of t_u^{sp} there exists a run ρ of $\mathscr{A}$ on u such that $\rho(i) \in t_u^{\mathrm{sp}}(\beta(i))$ for every $i < \omega$. We say that ρ is a *witness* for β.

Using the next definition, it is straightforward to express in terms of t_u^{sp} whether $u \in L(\mathscr{A})$. We say a branch β of a tree is *left-recurring* if it is infinite and $\beta(i)$ is a left successor for infinitely many i. This implies:

Remark 1. $u \in L(\mathscr{A})$ iff in t_u^{sp} there exists a left-recurring branch.

Assume t_u^{sp} has a left-recurring branch. Consider the following inductive process for constructing the "left-most left-recurring branch", denoted λ_u. For the basis, let $\lambda_u(0) = \epsilon$. For the inductive step, assume $\lambda_u(i)$ is defined and let $v = \lambda_u(i)$.

- If $v0$ is on a left-recurring branch of t, then $\lambda_u(i+1) = v0$.
- If $v0$ is not on a left-recurring branch, then $\lambda_u(i+1) = v1$.

That this process yields indeed a left-recurring branch is stated in the following lemma, but it also states another interesting property of λ_u. The lemma uses the following notation: $v \leq_{\mathrm{lft}} v'$ if $|v| = |v|'$ and $v \leq_{\mathrm{lex}} v'$, where $\leq_{\mathrm{lex}}$ denotes the lexicographical ordering.

Lemma 1. *Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Assume t_u^{sp} has a left-recurring branch. Then:*

1. *λ_u is left-recurring.*
2. *$\lambda_u \leq_{\mathrm{lex}} \beta$ for every left-recurring branch β of t_u^{sp}.*
3. *Let ρ be a witness for λ_u. Then $\rho(i) \notin t_u^{\mathrm{sp}}(v)$ for all $i < \omega$ and $v \in t_u^{\mathrm{sp}}$ with $v <_{\mathrm{lft}} \lambda_u(i)$.*

One property of the split tree is that, in general, it has unbounded width, which makes it difficult to handle. Since we know from the previous lemma that there is a left-most left-recurring path, it suggests itself to remove a state from a labeling of a vertex if it occurs in a vertex on the same level which is to the left of it. This leads to the following inductive definition of the *reduced split tree* for a word u with respect to $\mathscr{A}$, denoted t_u^{rs}, where the induction is on the lexicographic ordering of the vertices. For the basis, $\epsilon \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(\epsilon) = \{q_I\}$. For the inductive step, assume $v \in t_u^{\mathrm{rs}}$ and let $Q' = t_u^{\mathrm{rs}}(v)$, $i = |v|$, and $a = u(i)$. Further, let $Q'' = \bigcup\{\Delta(t_u^{\mathrm{rs}}(v'), a) \colon v' \in t_u^{\mathrm{rs}} \wedge v' <_{\mathrm{lft}} v\}$. Observe that Q'' is the set of states which have been assigned to vertices to the left of $v0$. Then:

- If $\Delta_F(Q', a) \setminus Q'' \neq \emptyset$, then $v0 \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(v0) = \Delta_F(Q', a) \setminus Q''$.
- If $\Delta_{\overline{F}}(Q', a) \setminus Q'' \neq \emptyset$, then $v1 \in t_u^{\mathrm{rs}}$ and $t_u^{\mathrm{rs}}(v1) = \Delta_{\overline{F}}(Q', a) \setminus Q''$.

From Remark 1 and Lemma 1, we can conclude:

Lemma 2. *Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Then $u \in L(\mathscr{A})$ iff in t_u^{rs} there exists a left-recurring branch.*

The other important property of t_u^{rs} is:

Remark 2. The tree t_u^{rs} has width at most n.

As we are only interested in the infinite branches of t_u^{rs}, we prune away the finite branches. The *skeleton* for u with respect to $\mathscr{A}$, denoted t_u^{sk}, is the subtree of t_u^{rs} which contains a vertex v if there exists an infinite branch β of t_u^{rs} with $v \in \beta$. By König's lemma, this is equivalent to having an infinite number of descendants.

Remark 3. The skeleton t_u^{sk} has an infinite left-recurring branch iff $u \in L(\mathscr{A})$.

4 Disambiguation and Complementation

The complementation and disambiguation construction to be presented make use of the fact that one can construct an unambiguous automaton that produces the slices of the skeleton. We first describe what exactly we mean by this and how this automaton can be used for disambiguation and complementation. We then turn to the construction of this automaton.

4.1 Slice Automaton and Its Applications

The *i-th skeleton slice* of u with respect to $\mathscr{A}$, denoted $\mathrm{skelslice}_i(u)$, is the i-th slice (see Sect. 2) of the skeleton for u. So $\mathrm{skelslice}_i(u) \in (2^Q)^+$. We say that a skeleton slice $Q_0 \dots Q_{s-1}$ is accepting if there exists $i < s$ such that $Q_i \subseteq F$.

Using the above definition and Remark 3 we can state:

Remark 4. 1. $u \in L(\mathscr{A})$ iff there are infinitely many i such that $\mathrm{skelslice}_i(u)$ is accepting.
2. $u \notin L(\mathscr{A})$ iff there is some i such that $\mathrm{skelslice}_j(u)$ is not accepting for all $j \geq i$.

We say a Büchi automaton $\mathscr{S} = (A, S, s_I, \Delta', F')$ is a *slice automaton* for $\mathscr{A}$ with respect to some function $h\colon S \to (2^Q)^+$ if for every $u \in A^\omega$ there exists exactly one accepting run ρ_u of $\mathscr{S}$ on u and this run has the property that $h(\rho(i)) = \mathrm{skelslice}_i(u)$ for every $i < \omega$.

Proposition 1. *For every Büchi automaton $\mathscr{A}$ there exists a slice automaton with at most $(3n)^n$ states.*

Before we sketch the construction, we explain how this automaton can be used for disambiguation and complementation.

The first part of Remark 4 tells us how to disambiguate: We simply check—using a Büchi condition and in a deterministic fashion—that infinitely accepting slices are produced by the slice automaton. The second part tells us how to complement: We simply guess a point from which onwards all skeleton slices produced by the slice automaton are not accepting.

To turn this into a construction, we use the following product definition. Let $\mathscr{A}$ be as above, $h\colon Q \to B$ a function, and $\mathscr{B} = (B, S, s_I, \Delta', F')$ a Büchi automaton. Then the Büchi automaton $\mathscr{A} \times_h \mathscr{B} = (A, Q \times S \times \{0,1\}, (q_I, s_I, 0), \Delta'', F \times S \times \{0\})$ is defined by: $((q,s,b), a, (q',s',b')) \in \Delta''$ for $(q,a,q') \in \Delta$ and $(s, h(q), s') \in \Delta'$, and where

$$b' = \begin{cases} 0 & \text{if } b = 0 \text{ and } q \notin F, \text{ or } b = 1 \text{ and } s \in F, \\ 1 & \text{if } b = 1 \text{ and } s \notin F', \text{ or } b = 0 \text{ and } q \in F. \end{cases}$$

Remark 5. 1. For every $u \in A^\omega$, $u \in L(\mathscr{A} \times_h \mathscr{B})$ iff there exists an accepting run ρ of $\mathscr{A}$ on u and an accepting run of $\mathscr{B}$ on $h(\rho(0))h(\rho(1)) \dots$.
2. If $\mathscr{A}$ and $\mathscr{B}$ are unambiguous, then so is $\mathscr{A} \times_h \mathscr{B}$.

Let $\mathscr{A}$ be a Büchi automaton, and $\mathscr{S}$ a slice automaton for $\mathscr{A}$ with respect to h. Let $\mathscr{C}^{\mathrm{dis}}$ and $\mathscr{C}^{\mathrm{cpl}}$ be defined by

where S_a and S stand for accepting and non-accepting slices, respectively. The automata $\mathscr{A}^{\mathrm{dis}}$ and $\mathscr{A}^{\mathrm{cpl}}$ are defined by

$$\mathscr{A}^{\mathrm{dis}} = \mathscr{S} \times_h \mathscr{C}^{\mathrm{dis}} \qquad\qquad \mathscr{A}^{\mathrm{cpl}} = \mathscr{S} \times_h \mathscr{C}^{\mathrm{cpl}} \ ,$$

respectively.

Fig. 1. Disambiguation and Complementation Construction

From this remark and the above observations, see Remark 4, it is now easy to derive disambiguation and complementation constructions. These are described in Figure 1, where, for complementation, w. l. o. g., we assume that every skeleton has at least one accepting slice.

Theorem 1. *Let $\mathscr{A}$ be a Büchi automaton.*

1. *The automaton $\mathscr{A}^{\mathrm{dis}}$ is an equivalent unambiguous Büchi automaton with at most $4(3n)^n$ states.*
2. *The automaton $\mathscr{A}^{\mathrm{cpl}}$ is a Büchi automaton with at most $4(3n)^n$ states satisfying $L(\mathscr{A}^{\mathrm{cpl}}) = A^\omega \setminus L(\mathscr{A})$ where A is the alphabet of $\mathscr{A}$.*

4.2 Construction of a Slice Automaton

The slice automaton we present produces, in a deterministic fashion, the slices of the reduced split tree. In addition, it guesses which vertices of each level are vertices of the skeleton and which are not.

The problem is that the automaton needs to verify that its guesses are correct. To achieve this, the automaton proceeds in phases as follows. At the beginning of a phase there are certain vertices which have been guessed to be skeleton vertices, and the others have been guessed to be non-skeleton vertices. The automaton follows the descendants of the vertices which have been guessed to be skeleton vertices and for each descendant it decides whether it is a skeleton vertex or not. The latter are put on hold for the next phase. Recall that by definition the non-skeleton vertices are the ones which have only finitely many descendants. So the automaton follows these descendants and ends the current phase as soon as there are no more such descendants. At this point, it reaches a final state and the states on hold take over the role of the states guessed to be non-skeleton vertices. If infinitely many phases are gone through, the guesses were correct.

We first study how the slices of the reduced split tree evolve. So assume $S = Q_0 \dots Q_{s-1}$ is slice i of the reduced split tree for u and $a = u(i)$. For every $j < s$, consider Q'_{2j} and Q'_{2j+1} defined by

$$Q'_{2j} = \Delta_F(Q_j, a) \ , \qquad Q'_{2j+1} = \Delta_{\overline{F}}(Q_j, a) \ .$$

Further, for every $j < s$, let $\tilde{Q}_j = \bigcup_{k<2j} Q'_k$ and set

$$Q''_{2j} = Q'_{2j} \setminus \tilde{Q}_j \ , \qquad Q''_{2j+1} = Q'_{2j+1} \setminus \tilde{Q}_j \ .$$

Slice $i+1$ of the reduced split tree is obtained by removing from $Q''_0 \dots Q''_{2s-1}$ all occurrences of $\emptyset$. We write $\Delta(S, a)$ for this word and $f_{S,a}$ for the partial function $\{0, \dots, 2s-1\} \to \{0, \dots, 2s-1\}$ which tells us where the Q''_j's are moved to because of removing $\emptyset$: If Q''_j is non-empty, then $f_{S,a}(j)$ is the number of non-empty sets in the sequence $Q''_0, \dots, Q''_{j-1}$, and undefined otherwise.

We now describe the state set of our slice automaton. A *decorated slice* is a word $(Q_0, b_0) \dots (Q_{s-1}, b_{s-1})$ where the Q_j's are pairwise disjoint, non-empty subsets of Q and where $b_j \in \{0, *, 1\}$ for $j < s$. The b_j's are meant to indicate whether the corresponding vertex of the respective level is guessed to belong to the skeleton (1), is being checked to be a non-skeleton vertex (0), or is put on hold ($*$). We say such a decorated slice is a reset slice if $b_j \neq 0$ for all $j < s$.

Let $D = (Q_0, b_0) \dots (Q_{s-1}, b_{s-1})$ be a decorated slice, $a \in A$, $S = Q_0 \dots Q_{s-1}$, $f = f_{S,a}$, and $P_0 \dots P_{t-1} = \Delta(S, a)$. An a-successor of D is a decorated slice of the form $(P_0, c_0) \dots (P_{t-1}, c_{t-1})$ where the c_j's satisfy the four conditions below. In what follows, assume $j < s$ and let $b = *$ if D is a reset slice and else $b = 0$.

[D1] If $b_j = 1$, then $f(2j)$ is defined and $c_{f(2j)} = 1$, or $f(2j+1)$ is defined and $c_{f(2j+1)} = 1$.

[D2] If $b_j = 1$, then $c_{f(2j)} \in \{*, 1\}$ and $c_{f(2j+1)} \in \{*, 1\}$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

[D3] If $b = *$ and $b_j = *$, then $c_{f(2j)} = 0$ and $c_{f(2j+1)} = 0$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

[D4] If $b = 0$ and $b_j \in \{0, *\}$, then $c_{f(2j)} = b_j$ and $c_{f(2j+1)} = b_j$, provided $f(2j)$ and $f(2j+1)$, respectively, are defined.

Observe that these conditions exactly reflect the informal description above. For instance, [D3] says that when a phase is over—we have a reset slice—then the successors of all states on hold are marked as non-skeleton vertices and will be checked to have only a finite number of descendants.

The entire automaton is described in Figure 2. The function h postulated by Proposition 1 is defined by $h((Q_0, b_0) \dots (Q_{s-1}, b_{s-1})) = Q_{i_0} \dots Q_{i_{t-1}}$ where $i_0 < \dots < i_{t-1}$ and $\{i_0, \dots, i_{t-1}\} = \{j < s \colon b_j = 1\}$.

5 Determinization

To motivate our determinization construction, we start with a few definitions, assuming, w. l. o. g., that each skeleton has at least two branches.

We write $\mathrm{branches}(u)$ for the set of branches of the skeleton t_u^{sk}. A *fork* of a binary tree is a vertex which has exactly two successors. The set of all forks of a tree t is denoted by $\mathrm{forks}(t)$.

Our determinization construction is based on the fact that u is accepted by $\mathscr{A}$ iff there exists a left-recurring $\beta \in \mathrm{branches}(u)$. Observe that for each $\beta \in \mathrm{branches}(u)$

Let $\mathscr{A}$ be a Büchi automaton. The *slice automaton* for $\mathscr{A}$ is the Büchi automaton denoted $\mathscr{A}^{\mathrm{slc}}$ and defined by

$$\mathscr{A}^{\mathrm{slc}} = (A, Q', q'_I, \Delta', F')$$

where
- Q' is the set of all decorated slices over Q,
- q'_I is the decorated slice $(\{q_I\}, 1)$,
- Δ' contains all transitions (D, a, D') such that D' is an a-successor of D, and
- F' contains all reset slices.

Fig. 2. Definition of Slice Automaton

there exists a unique $v \in \mathrm{forks}(t_u^{\mathrm{sk}})$ and $d \in \{0,1\}$ such that $vd \in \beta$ and $vd \notin \beta'$ for every $\beta' \in \mathrm{branches}(u) \setminus \{\beta\}$. That is, v is the last fork in t_u^{sk} lying on β. We write y_β and d_β for v and d, respectively. Our deterministic automaton tries to identify y_β and d_β for each β and to check that a left-recurring path of t_u^{sk} starts in some $y_\beta d_\beta$. To this end, the automaton approximates the skeleton t_u^{sk} as explained in what follows.

Approximation i of u is the tree consisting of all vertices of t_u^{rs} which potentially belong to t_u^{sk} given only the prefix of length i of u. Formally, *approximation i (of the skeleton) of u* with respect to $\mathscr{A}$ is the tree $a_u^i \subseteq \{0,1\}^*$ defined by $v \in a_u^i$ iff there exists $v' \in t_u^{\mathrm{rs}} \cap \{0,1\}^i$ such that $v \leq_{\mathrm{prf}} v'$.

We prove that the forks of the skeleton can be deduced from the sequence of the approximations of t_u^{sk}:

Lemma 3. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $v \in \{0,1\}^*$. Then $v \in \mathrm{forks}(t_u^{\mathrm{sk}})$ iff $v \in \mathrm{forks}(a_u^i)$ for all but finitely many i.*

To be able to handle approximations by a finite automaton, we use $\mathrm{forks}(a_u^i) \cup \mathrm{leafs}(a_u^i)$ as the vertex set of a tree—called contraction—which represents the branching structure of a_u^i. When t is a tree, v and v' are vertices of t, $v_0 \dots v_s$ is the path from v to v', $v_i \notin \mathrm{forks}(t)$ for $i < s$, and $v_s \in \mathrm{forks}(t) \cup \mathrm{leafs}(t)$, we write $v \rightsquigarrow v'$. In other words, $v \rightsquigarrow v'$ if v' is the next descendant of v which is a fork or a leaf.

The *contraction* of a tree t, denoted $C(t)$, is the tree $(\mathrm{forks}(t) \cup \mathrm{leafs}(t), s_0, s_1)$ where for $v, v' \in \mathrm{forks}(t) \cup \mathrm{leafs}(t)$ and $d \in \{0,1\}$, $s_d(v) = v'$ if $vd \rightsquigarrow v'$. The contraction of approximation i of u is denoted C_u^i and called *contraction i of u*. Similarly, $C(t_u^{\mathrm{sk}})$, the contraction of t_u^{sk}, is denoted C_u.

Corollary 1. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $v \in \{0,1\}^*$. Then $v \in C_u$ iff $v \in C_u^i$ for all but finitely many i.*

Further, there exists a number i_u such that $C_u \subseteq C_u^j$ for all $j \geq i_u$.

Next we derive a criterion that tells us whether a given $\beta \in \mathrm{branches}(u)$ is left-recurring. Observe that the vertex $y_\beta d_\beta$ is, in general, not an element of the C_u^j's. But, clearly, for $j > i_u$ the vertex y_β is an element of C_u^j and it has a d-successor in C_u^j, which we denote by w_j^β. This vertex is on β. The next lemma tells us that the w_j^β's move along β as j increases, and how one can check whether β is left-recurring or not. It uses the terminology introduced in the following definition.

A sequence $v_0, v_1, \dots$ of vertices of a branch β *covers* this branch if $v_0 \leq_{\mathrm{prf}} v_1 \leq_{\mathrm{prf}} v_2 \leq_{\mathrm{prf}} \dots$ and if, for every $i < \omega$, there exists some j such that $|v_j| \geq i$. For $v, w \in \{0,1\}^*$, we say that w is a *left descendant* of v and write $v \to^l w$ if there exists $v' \in \{0,1\}^*0\{0,1\}^*$ such that $w = vv'$, that is, if on the way from v to w there is a "left turn".

Using Corollary 1, we prove:

Lemma 4. *Let $\mathscr{A}$ be a Büchi automaton, u an infinite word over the same alphabet, and $\beta \in \mathrm{branches}(u)$.*

1. *The sequence $w^\beta_{i_u+1}, w^\beta_{i_u+2}, w^\beta_{i_u+3}, \dots$ covers β.*
2. *The branch β is left-recurring iff there are infinitely many j such that $w^\beta_j \to^l w^\beta_{j+1}$.*

This motivates our determinization construction. The automaton produces isomorphic copies of the contractions, uses them to identify the forks of the skeleton, and checks whether there is one fork whose left or right successors in the approximations cover an infinite branch which is left-recurring. The latter is checked according to the second part of Lemma 4 by using an appropriate acceptance condition.

Note that a finite automaton cannot produce contractions directly, because they have vertices which are vertices of the skeleton t^{sk}_u and as such they are strings of unbounded length. Once an automaton works with isomorphic copies of contractions though, it is not obvious anymore how to identify vertices that cover a left-recurring branch. We solve this by (i) using the same vertex in two consecutive isomorphic copies if and only if it represents the same vertex of the original contractions and (ii) storing in each inner vertex of the isomorphic copies whether or not its right successor represents a left descendant of the vertex it represents. Observe that a left successor always represents a vertex which is a left descendant.

The details of our construction are as follows. Since each contraction has at most $2n-1$ vertices and in each step at most n leafs are added, the vertices for the isomorphic copies of the contractions are taken from the set $\{0, \dots, 3n-2\}$, which we denote by U. A *contraction tree* is a full binary tree T with at most $2n-1$ vertices from U and labels from $2^Q \cup \{0,1\}$ such that $T(v) \in \{0,1\}$ for each inner vertex and $T(v) \subseteq Q$ for each leaf. Each inner vertex carries information about its right successor (see (ii) above), and each leaf has the same label as the vertex it represents.

We will use the following notation. Assume T is a tree where each inner vertex is labeled by 0 or 1, v' is a descendant of v, and $v_0 \dots v_s$ the path from v to v'. We write $v \to^0 v'$ if $T(v_i) = 1$ for some $i < s$ or if v_{i+1} is the left successor of v_i for some $i < s-1$. This is the analogue of $\to^l$ for contraction trees.

The crucial part is to describe the transition function. To this end, let T be a contraction tree, assume $v_0, \dots, v_{s-1}$ is a listing of $\mathrm{leafs}(T)$ in infix order (from left to right), let $S = T(v_0) \dots T(v_{s-1})$, $a \in A$, $Q'_0 \dots Q'_{t-1} = \Delta(S, a)$, and $f = f_{S,a}$. We want to define the *a-successor* of T, which we denote by T_a. This tree is obtained from T by a series of transformations modeling (i) extending the corresponding approximation by one level, (ii) removing superfluous vertices, (iii) contracting the tree, and (iv) adjusting the labeling.

We assume that $w_0, w_1, \dots$ is the sequence of the elements of W defined by $W = U \setminus T$, say in increasing order. Note that W is the set of vertices which are not "used" in T.

Let $\mathscr{A}$ be a Büchi automaton. The *contraction automaton* for $\mathscr{A}$ is the deterministic automaton (without acceptance condition) denoted $\mathscr{A}^{\mathrm{ctr}}$ and defined by

$$\mathscr{A}^{\mathrm{ctr}} = (A, Q', T_I, \delta)$$

where
- Q' is the set of all contraction trees,
- T_I is the one-vertex tree with $v_T = 0$ and $T(v_T) = \{q_I\}$, and
- $\delta(T, a)$ is the a-successor of T for every $T \in Q'$ and $a \in A$.

Fig. 3. Definition of Contraction Automaton

1. *Extend T by one level.* For every $j < s$, first set $T(v_j) = 0$. Then,
   - if $f(2j)$ is defined, make $w_{f(2j)}$ the 0-successor of v_j and set $T(w_{f(2j)}) = Q'_{f(2j)}$, and
   - if $f(2j+1)$ is defined, make $w_{f(2j+1)}$ the 1-successor of v_j and $T(w_{f(2j+1)}) = Q'_{f(2j+1)}$.
2. *Remove superfluous vertices and relabel.* The tree T' is obtained from T by removing all vertices which do not have a descendant in W and changing the labeling as follows. If $v \in \mathrm{forks}(T)$, $d \in \{0, 1\}$, v' is the d-successor of v, and $v' \rightsquigarrow v''$, then $T'(v) = 1$ iff $v \to^0 v''$.
3. *Contract T'.* Finally, T_a is obtained by contracting T' to T_a. (Observe that the definition of contraction given above for trees in implicit notation can easily be adapted to trees in explicit notation.)

The full construction of the deterministic automaton which produces the isomorphic copies of the contractions, the so-called *contraction automaton*, is depicted in Figure 3.

Proposition 2. *Let $\mathscr{A}$ be a Büchi automaton. The contraction automaton $\mathscr{A}^{\mathrm{ctr}}$ has $\lessapprox (4.3n)^{4n}$ states and for every u over the same alphabet the run ρ_u of $\mathscr{A}^{\mathrm{ctr}}$ on u has the following properties:*

1. *For every i, the tree $\rho_u(i)$ is isomorphic with C^i_u, say via $\pi_i \colon \rho_u(i) \to C^i_u$.*
2. *For every $v \in \rho_u(i)$, the vertex v belongs to $\rho_u(i+1)$ iff the vertex $\pi_i(v)$ belongs to C^{i+1}_u, and if this is the case, then $\pi_i(v) = \pi_{i+1}(v)$.*
3. *For every $v \in \mathrm{inner}(\rho_u(i))$, if v' is the right successor of v in $\rho_u(i)$, then $\pi_i(v) \to^l \pi_i(v')$ iff $\rho_u(i)(v) = 1$.*

From the previous proposition we can conclude:

Corollary 2. *Let $\mathscr{A}$, u, and ρ_u as in Proposition 2. Then $u \in L(\mathscr{A})$ iff there is some $v \in U$ such that*

(a) $v \in \rho_u(i)$ for all but finitely many i and

(b) for infinitely many i, the vertex v has a d-successor v'' in $\rho_u(i+1)$ such that v'' is a leaf or $v' \to^0 v''$ in $\rho_u(i)$ for the d-successor v' of v.

To conclude, we explain how (a) and (b) from the corollary can be checked using appropriate acceptance conditions. We first use a custom acceptance condition. Let M be a finite set. A pair (α, β) of functions $Q \times A \to 2^M$ is called an *$\alpha\beta$-condition* over M

if $\alpha(q,a) \supseteq \beta(q,a)$ for all $q \in Q$ and $a \in A$. A run ρ on a word u is accepting with respect to such a condition if there exists some $m \in M$ such that $m \in \alpha(\rho(i), u(i))$ for all but finitely many i and $m \in \beta(\rho(i), u(i))$ for infinitely many i. The order of such a condition is the maximum of the values $|\alpha(q,a)|$.

For the contraction automaton, we choose $M = U$, let $\alpha(T,a)$ be the set of vertices of $\delta(T,a)$ (cf. (a)), and let $\beta(T,a)$ be the set of vertices v of $\delta(T,a)$ satisfying: for some $d \in \{0,1\}$, the vertex v has a d-successor v'' in $\delta(T,a)$ such that v'' is a leaf or $v' \to^0 v''$ in T for the d-successor v' of v (cf. (b)).

Any $\alpha\beta$-condition can be converted into ordinary acceptance conditions:

Lemma 5. *Let $\mathscr{A}$ be a deterministic automaton with n states and an $\alpha\beta$-acceptance condition over a set with s elements and of order m. Then there exists*

1. *an equivalent Muller automaton with $n3^s$ states,*
2. *an equivalent Rabin automaton with $n3^s$ states and s Rabin pairs, and*
3. *an equivalent parity automaton with at most $n(s+1)!$ states and $2m+1$ priorities.*

The proof of the last part of the lemma follows Piterman's construction [15], which itself uses the latest appearance record with hit. The first and second part simply add a function $M \to \{0,1,2\}$ to each state.

We can finally state:

Theorem 2. *Let $\mathscr{A}$ be a Büchi automaton. The deterministic automaton $\mathscr{A}^{\mathrm{ctr}}$ augmented by the above acceptance condition can be converted into a deterministic Muller, Rabin, or parity automaton equivalent to $\mathscr{A}$ with $2^{O(n \log n)}$ states (and n Rabin pairs or $2n+1$ priorities).*

We conclude with two open problems:

1. Give a lower bound for disambiguating Büchi automata.
2. Generalize our determinization construction to Streett automata.

References

1. Schulte-Althoff, C., Thomas, W., Wallmeier, N.: Observations on determinization of Büchi automata. Theor. Comput. Sci. 363(2), 224–233 (2006)
2. Arnold, A.: Rational omega-languages are non-ambiguous. Theor. Comput. Sci. 26, 221–223 (1983)
3. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Nagel, E., Suppes, P., Tarski, A. (eds.) Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress, pp. 1–11. Stanford University Press, Stanford (1962)
4. Emerson, E.A., Sistla, A.P.: Deciding full branching time logic. Information and Control 61(3), 175–201 (1984)
5. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. Int. J. Found. Comput. Sci. 17(4), 851–868 (2006)
6. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: 14th ACM Symposium on the Theory of Computing, San Francisco, Calif, pp. 60–65. ACM, New York (1982)
7. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 96–110. Springer, Heidelberg (2003)

8. Klarlund, N.: Progress measures for complementation of ω-automata with applications to temporal logic. In: 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, pp. 358–367. IEEE, Los Alamitos (1991)
9. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Logic 2(3), 408–429 (2001)
10. Kähler, D.: Determinisierung von ω-Automaten. Diploma thesis, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel (2001)
11. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
12. Michel, M.: Complementation is more difficult with automata on infinite words (unpublished notes) (1988)
13. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the 4th Annual IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 3–16 (1963)
14. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theor. Comput. Sci. 141(1&2), 69–107 (1995)
15. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic Parity automata. In: 21th IEEE Symposium on Logic in Computer Science, Seattle, WA, USA, Proceedings, pp. 255–264. IEEE, Los Alamitos (2006)
16. Ozer Rabin, M.: Decidability of second-order theories and finite automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
17. Safra, S.: On the complexity of ω-automata. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, pp. 319–327. IEEE, Los Alamitos (1988)
18. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with appplications to temporal logic. Theor. Comput. Sci. 49, 217–237 (1987)
19. Thomas, W.: Complementation of Büchi automata revised. In: Karhumäki, J., Maurer, H.A., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 109–120. Springer, Heidelberg (1999)
20. Vardi, M.Y., Wilke, Th.: Automata: from logics to algorithms. In: Flum, J., Grädel, E., Wilke, Th. (eds.) Logic and Automata: History and Perspectives. Texts in Logic and Games, vol. 2, pp. 629–736. Amsterdam University Press, Amsterdam (2007)
21. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37
22. Yan, Q.: Lower bounds for complementation of ω-automata via the full automata technique. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 589–600. Springer, Heidelberg (2006)