ω-Automata

Thomas Wilke

Christian-Albrechts-Universität zu Kiel
email: thomas.wilke@email.uni-kiel.de

March 14, 2014 18 h 48

2010 Mathematics Subject Classification: 68Q45

Key words: ω-automata.

Contents

1 Introduction

The expression "ω-automata" generally refers to automata that accept or reject ω-words—infinite sequences of letters from some alphabet. They define ω-languages—sets of ω-words—just as ordinary automata define languages of finite words; they are means for working with ω-languages, just as ordinary automata are means for working with languages of finite words.

The fundamental questions about ω-automata are similar to the fundamental questions about automata on finite words: Can operations on languages, such as intersection, complementation, and projection, be performed on automata? What is the difference between non-determinism and determinism? What is the descriptional complexity of various types of automata? Which types of automata can be transformed into other types and at which cost? Can automata be minimized efficiently? How can automata be compared? ...

Some of the techniques developed for finite words can be adopted in the infinite-word setting; some of the theory of ω-automata is very similar to ordinary automata theory. There are, however, many interesting new aspects, which, most often, have something to do with what happens "in the infinite". As this interesting behavior "in the infinite" is already present in ω-automata with a finite state space, this chapter is limited to the theory of finite-state ω-automata.

To convey the core ideas as crisp and clear as possible within the given space limits, some of the material is presented in an uncommon way, at the expense of continuity with prior work.

Excellent surveys that cover ω-languages and ω-automata in their entire breadth, especially their relationship with mathematical logic, have been written by Wolfgang Thomas, one in the late eighties [45], and one in the nineties [46]; there is also a comprehensive

monograph by Dominique Perrin and Jean-Éric Pin [32]. This chapter tries to be a concise introduction into the *theory* of ω-automata.

1.1 ω-Words

ω-Automata are devices that work on ω-words rather than finite ones. Technically, an ω-word over an alphabet A is a function $\omega \to A$, where ω stands for the set of natural numbers. In contrast, a finite word over A is a function $[n] \to A$, where n is a natural number and $[n]$ denotes the set $\{0, \dots, n-1\}$. When u is a word, finite or infinite, then $\mathrm{occ}(u)$ denotes the set of letters occurring in u; when u is an ω-word, then $\inf(u)$ denotes the set of letters occurring infinitely often in u.

There are essentially three concatenation operations involving ω-words. First, given a finite word u and an ω-word v, the ω-word which is obtained by appending v to u is denoted $u \cdot v$—we speak of ω-concatenation. Second, when $\langle u_0, u_1, \dots \rangle$ is an infinite sequence of finite non-empty words, then $u_0 \cdot u_1 \cdot \dots$ denotes the ω-word obtained by concatenating all the u_i's in the given order—we speak of ω-product. Finally, when u is a finite non-empty word, then u^ω denotes $u \cdot u \cdot \dots$—we speak of ω-power. Note that it is legitimate to use the same symbol $\cdot$ for ordinary concatenation, ω-concatenation, and ω-product, because there are various laws of associativity that hold true. As usual, the symbol "$\cdot$", representing the different forms of concatenation, is omitted in many contexts, and these operations are extended to sets of words in a straightforward fashion. An ω-word u is periodic if $u = v^\omega$ for some finite non-empty word v; it is ultimately periodic if $u = vw^\omega$ for finite words v and w with w being non-empty. The convolution of ω-words u and v over alphabets A and B, respectively, is an ω-word over the alphabet $A \times B$, denoted $u * v$ and defined by $(u * v)(i) = \langle u(i), v(i) \rangle$ for every i.

One of the reasons why ω-automata are applicable in various situations is that ω-words can represent infinite objects and therefore ω-automata can represent sets of infinite objects or even transform infinite objects into other infinite objects.

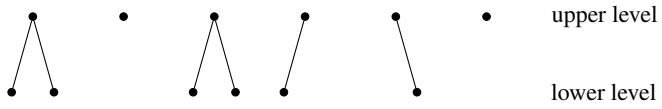
When the alphabet A is the binary alphabet, $[2]$, then an ω-word can be identified with a subset of ω, that is, with a set of natural numbers, more precisely, a word u can be identified with $\{i \in \omega \mid u(i) = 1\}$. When the alphabet is $\bigtimes_{i<k}[2]$, the k-fold cartesian product of $[2]$, then an ω-word can be identified with a k-tuple of sets of natural numbers.

Every ω-word over $[2]$ represents a real number from the interval $[0, 1]$ in a natural fashion, more precisely, $u \in [2]^\omega$ represents the number $\sum_i u(i)\, 2^{-i-1}$. Observe that some numbers are represented twice, for instance, $1/2$, which is represented by 10^ω and by 01^ω.

There are various ways to represent any real number by an ω-word. One way is to consider only words u over $[2]$ where $u(2i+2) = 0$ for almost all i and then let u represent $(-1)^{u(0)}(\sum_i u(2i+2)2^i + \sum_i u(2i+1)2^{-i-1})$. So the letters at even positions determine the integer part, including the sign, and the letter at odd positions determine the fractional part. Another way is to use a larger alphabet, for instance, $[2] \times [2]$, and represent the integer part in one dimension and the fractional part in the other dimension.

In this chapter, a binary tree is a prefix-closed subset of $[2]^*$; level i of such a tree T is the set of vertices $T \cap [2]^i$; its width is $\sup_i |T \cap [2]^i|$. Given some k, the set of trees of width at most k can be represented by ω-words over a fixed alphabet. A simple such representation is the sequence of (representations of) its slices, where a slice is two

consecutive levels together with their interconnections, that is, a slice looks like this:



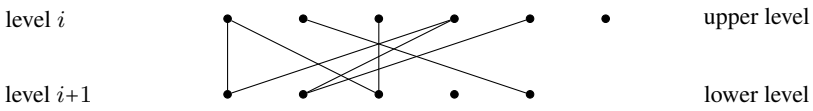
To represent this slice one could use the single "letter" $\langle 1,1\rangle\langle 0,0\rangle\langle 1,1\rangle\langle 1,0\rangle\langle 0,1\rangle\langle 0,0\rangle$.

Not all ω-words over the respective alphabet represent a tree, but every tree (of a given maximum width k) can be represented. For instance, the tree denoted by $0^*(\epsilon + 1 + 11)$, which looks like a comb, is represented by $\langle 1,1\rangle\,(\langle 1,1\rangle\langle 0,1\rangle)\,(\langle 1,1\rangle\langle 0,1\rangle\langle 0,0\rangle)^\omega$.

A labeled binary tree is a function $T \to A$ from a binary tree to an alphabet, and such trees, if restricted in their width, can also be represented by ω-words, simply by augmenting the above representation by information about the labels. It is enough to encode in the representation of one slice the labels of the vertices in the "upper" level.

What has been just said for infinite trees is also true for graphs to a certain extent. A leveled DAG is a directed acyclic graph together with a partition of its vertex set into levels, more precisely, such a graph is given by a family $\{V^{(i)}\}_{i\in\omega}$ of pairwise disjoint vertex sets and an edge set $E \subseteq \bigcup_i V^{(i)} \times V^{(i+1)}$. The set of all vertices, $\bigcup_i V^{(i)}$, is denoted V; the elements of $V^{(i)}$ are the vertices on level i. Similarly to above, the width of such a DAG is $\sup_i |V^{(i)}|$.

Given a natural number k, leveled DAG's of width at most k can be represented over a fixed alphabet, again by spelling out the sequence of its slices, where a slice represents a subgraph induced by two consecutive levels. Such a subgraph can, for instance, look like this:



This slice could be represented by the "letter" $\langle\{0,2\},\{4\},\{2\},\{0,1\},\{1\},\{\}\rangle$, an enumeration of the adjacency sets of the vertices on the the upper level. We are only interested in leveled graphs up to isomorphism; so there are, in general, many representations for the same graph. Labeled leveled DAG's of bounded width can also be represented by ω-words.

2 Types of ω-automata

A finite-state non-deterministic ω-automaton over a given alphabet A consists of

- a finite set of states, Q,
- a set of initial states, $Q_I \subseteq Q$,
- a set of transitions, $\Delta \subseteq Q \times A \times Q$, and

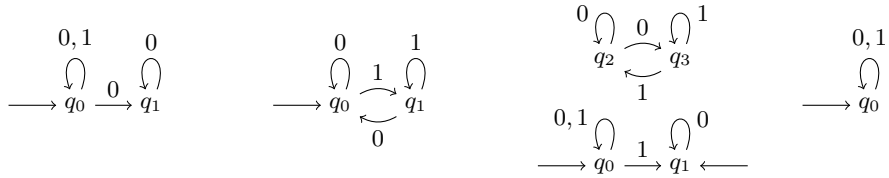


Figure 1. Different types of ω-automata for the set of all ω-words over $[2]$ with a finite number of occurrences of 1, denoted L_{fin1}: the first automaton is forward non-deterministic and, if augmented with the Büchi condition $\{q_1\}$, recognizes L_{fin1}; the second one is forward deterministic and, if augmented with the co-Büchi condition $\{q_0\}$, the parity condition $\pi\colon q_0 \mapsto 2, q_1 \mapsto 1$, the Muller condition $\{\{q_0\}\}$, or a suitable Streett or Rabin condition, recognizes L_{fin1}; the third one is backward deterministic if used with the Büchi condition $\{q_1, q_3\}$ and then recognizes L_{fin1}; the fourth one is forward deterministic and recognizes L_{fin1} if augmented with the transition-Muller condition $\{\{(q_0, 0, q_0)\}\}$.

- a representation of a recurrence condition.

The last ingredient is the one that distinguishes an ω-automaton from an ordinary finite-state automaton working on finite words; it replaces the set of final states. This is necessary, because ω-words have no end. But before recurrence conditions can be explained in detail, the notion of a run needs to be adapted to ω-words.

A run of an ω-automaton on a given word $u \in A^\omega$ is an ω-word $r \in Q^\omega$ such that $\langle r(i), u(i), r(i+1)\rangle \in \Delta$ holds for every $i \in \omega$. It is initial if $r(0) \in Q_I$ holds; it is accepting if it is initial and recurring, and what it means for r to be recurring depends on the recurrence condition, as described in what follows.

A simple type of recurrence condition is the Büchi type, which is represented as a set $B \subseteq Q$; a run r is recurring if one of the states from B occurs infinitely often in it, that is, if $\inf(r) \cap B \neq \varnothing$ holds. For instance, the language L_{fin1} defined by $L_{\text{fin1}} = \{u \in [2]^\omega \mid \inf(u) = \{0\}\}$ ("1 occurs only finitely often") is recognized by a Büchi automaton with two states, see Figure 1. (As usual, an automaton recognizes the language consisting of the words the automaton has an accepting run for; when an automaton is denoted $\mathcal{A}$, this language is denoted $\mathrm{L}(\mathcal{A})$.)

Obviously, using a Büchi condition one can neither specify that two particular states occur infinitely often nor that a specific state occurs only finitely often. A type of recurrence condition which is sufficiently expressive in this respect is the Muller type. Such a condition is represented by a set $\mathcal{M} \subseteq \mathcal{P}(Q)$; a run r is recurring if $\inf(r) \in \mathcal{M}$. In other words, one explicitly specifies which states occur infinitely often in a run and which are the ones that occur only finitely many times. The language L_{fin1} can also be recognized by a Muller automaton with two states, see Figure 1.

There are essentially five different types of recurrence conditions that have been investigated traditionally, all explained in the upper part of Table 1 and named after their originators [5, 35, 44, 29] except for the parity condition [28]. In the lower part of the table, there are three types of conditions derived from the Büchi condition [30, 19, 10].

The trivial recurrence condition, which is not mentioned in the table, considers every run recurrent. For instance, all representations of binary trees of a fixed width (over the alphabet indicated in Section 1.1) is recognized by an automaton with trivial recurrence condition.

Name		Format	Semantics
Büchi	B	$B \subseteq Q$	$\inf(r) \cap B \neq \emptyset$
parity	$\mathcal{P}$	$\pi: Q \to [2n]$, $n \in \omega$	$\min(\inf(\pi \circ r)) \bmod 2 = 0$
Rabin	$\mathcal{R}$	$\mathcal{R} \subseteq \wp(R) \times \wp(R)$	there exists $\langle L, U\rangle \in \mathcal{R}$ such that $\inf(r) \cap L = \emptyset$ and $\inf(r) \cap U \neq \emptyset$
Streett	$\mathcal{S}$	$\mathcal{S} \subseteq \wp(R) \times \wp(R)$	for every $\langle R, G\rangle \in \mathcal{S}$, if $\inf(r) \cap R \neq \emptyset$, then $\inf(r) \cap G \neq \emptyset$
Muller	$\mathcal{M}$	$\mathcal{M} \subseteq \wp(Q)$	$\inf(r) \in \mathcal{M}$
weak	W	$W \subseteq Q$, union of SCC's	$\inf(r) \subseteq W$
co-Büchi	C	$C \subseteq Q$	$\inf(r) \subseteq C$
gen. Büchi	$\mathcal{G}$	$\mathcal{G} \subseteq \wp(Q)$	$B \cap \inf(r) \neq \emptyset$ for every $B \in \mathcal{G}$

Table 1. Recurrence conditions, ordered according their expressive power; "gen. Büchi" and "SCC" are abbreviation of "generalized Büchi" and "strongly connected component", respectively.

It is convenient to give names to the elements of the recurrence conditions: a state $q \in B$ is called a Büchi state, a pair $\langle L, U\rangle \in \mathcal{R}$ is called a Rabin pair, a pair $\langle R, G\rangle \in \mathcal{S}$ is called a Streett pair, a set $M \in \mathcal{M}$ is called a Muller set, a state $q \in W$ is called a weak state, a state $q \in C$ is called a co-Büchi state, and a set $B \in \mathcal{G}$ is called a Büchi set. The function π is called priority function. Only weak, co-Büchi, and Büchi conditions have straightforward representations of size polynomial in the number of states of a given automaton.

Every type of recurrence condition is also considered in a transition variant, where states from Q are replaced by transitions from Δ and $\inf(r)$ is replaced by the set of triples $\langle q, a, q'\rangle$ for which there exist an infinite number of i such that $\langle r(i), u(i), r(i+1)\rangle = \langle q, a, q'\rangle$. Transition variants come in more handy in certain situations, for instance, L_{fin1} is recognized by a single-state transition-Muller automaton, see Figure 1.

Table 2 shows how conditions of various types can be expressed in terms of conditions of other types: every Büchi condition may be viewed as a parity condition, which, in turn, can be viewed as a Rabin or Streett condition, and these can be viewed as Muller conditions.

Generality:
Muller —> (Rabin, Streett) --> Parity —> Büchi

From	To	Conversion
Büchi B	parity π	$\pi(q) = 0$ for $q \in B$ and $\pi(q) = 1$ for $q \notin B$
parity π	Rabin $\mathcal{R}$	$\mathcal{R} = \{(\pi^{-1}(\{0, \ldots, 2i-1\}), \pi^{-1}(\{0, \ldots, 2i\})) \mid i+1 < n\}$
parity π	Streett $\mathcal{S}$	$\mathcal{S} = \{(\pi^{-1}(\{0, \ldots, 2i+1\}), \pi^{-1}(\{0, \ldots, 2i\})) \mid i+1 < n\}$
Rabin $\mathcal{R}$	Muller $\mathcal{M}$	$\mathcal{M}$ = set of all $Q' \subseteq Q$ such that there exists $\langle L, U\rangle \in \mathcal{R}$ satisfying $Q' \cap L = \emptyset$ and $Q' \cap U \neq \emptyset$
Streett $\mathcal{S}$	Muller $\mathcal{M}$	$\mathcal{M}$ = set of all $Q' \subseteq Q$ such that, for every $\langle R, G\rangle \in \mathcal{S}$, if $Q' \cap R \neq \emptyset$, then $Q' \cap G \neq \emptyset$

Table 2. Conversions between recurrence conditions

Unlike finite words, ω-words are not symmetric. So when talking about determinism the direction makes a difference. A forward deterministic automaton is one where Q_I consists of exactly one state and $|\Delta(q,a)| = 1$ for all $q \in Q, a \in A$. (As usual, $\Delta(q,a)$ is used as an abbreviation of $\{q' \in Q \mid \langle q,a,q'\rangle \in \Delta\}$.) A backward deterministic automaton[1] is one where for every ω-word over A, there is exactly one recurring run and $|\Delta(a,q')| = 1$ for all $a \in A, q' \in Q$. (Here, $\Delta(a,q')$ stands for $\{q \in Q \mid \langle q,a,q'\rangle \in \Delta\}$.) At times, when deterministic automata are used, the transition relation Δ is replaced by a transition function $\delta\colon Q \times A \to Q$ (forward automata) or $\delta\colon A \times Q \to Q$ (backward automata).

In general, a type of an ω-automaton is given by a type of recurrence condition *and* a type of mode, with the following modes being considered: non-deterministic (default), forward deterministic, backward deterministic, and alternating, defined in Section 9.2.

The most fundamental result about ω-automata compares the different types of ω-automata with respect to their expressive power. As a yardstick, non-deterministic Büchi automata are used; the ω-languages recognized by them are called regular ω-languages, see Theorem 3.1 for the origin of this terminology.

Theorem 2.1 (equivalence of types of ω-automata). *For every type of ω-automaton, consider the class of ω-languages recognized by automata of this type. Then all these classes coincide with the class of regular ω-languages except for the classes corresponding to the following types: forward deterministic generalized Büchi; forward deterministic Büchi; forward deterministic, backward deterministic, and non-deterministic co-Büchi; forward deterministic, backward deterministic, and non-deterministic weak.*

Much of ω-automata theory revolves around Theorem 2.1. The quest for good proofs of this theorem—efficient language-preserving transformations between automata of different types—has brought up many interesting results. All types of ω-automata are interesting in their own right; each one has its advantages and applications in specific contexts.

3 Basic properties of Büchi automata

Some basic insights into ω-automata can be derived from analyzing runs in a straightforward fashion, for instance, that regular languages can be defined by regular expressions of a certain type, that deterministic Büchi automata are less expressive than non-deterministic ones, and that complementation is problematic for Büchi automata.

3.1 ω-Regular expressions

An ω-regular expression [29] is of the form

$$r_0 \cdot s_0^\omega + \cdots + r_{n-1} \cdot s_{n-1}^\omega \quad , \tag{3.1}$$

[1] In [7], where these automata were introduced, they are called "complete unambiguous"; in [32], the attribute "prophetic" is used; in [34], they are referred to as "Carton-Michel automata". The terminology used in this chapter tries to be systematic.

with n being a natural number and the r_i's and the s_i's being ordinary regular expressions. The semantics is the obvious one.

Since the empty set can be denoted by an empty expression ($n = 0$) and since our definition of ω-power is only defined for sets of non-empty finite words (see Section 1.1), it is reasonable to require that the s_i's be built from the letters of the alphabet, "+" (for union), "$\cdot$" (for concatenation), and "$^+$" (for finite positive iteration). It is also reasonable to allow that individual r_i's are omitted.

Theorem 3.1 (ω-regular expression [5]). *Every ω-language recognized by a Büchi automaton is denoted by an ω-regular expression and vice versa.*

For the proof, assume a Büchi automaton is given. The insight needed is that for every accepting run r there are some state $q \in B$ and an infinite sequence $\langle i_0, i_1, \ldots \rangle$ of positions such that $r(0)$ is initial, $i_0 < i_1 < \ldots$, and $r(i_j) = q$ for every j. This motivates the following definition. For states q, q', let $L_{q,q'}$ be the language recognized by the ordinary automaton on finite words with q as initial and q' as final state. Then the language recognized by the Büchi automaton is

$$\bigcup_{q \in Q_I, q' \in B} L_{q,q'} (L_{q',q'} \setminus \{\epsilon\})^\omega \quad . \tag{3.2}$$

This representation can be turned into an ω-regular expression using techniques known from finite-state automata on finite words.

For the proof of the converse, first observe that it is enough to show that an expression of the form $r \cdot s^\omega$ (meaning $n = 1$) denotes a language recognized by a Büchi automaton, because the class of languages recognized by Büchi automata is closed under union: the disjoint union of two given Büchi automata is a Büchi automaton recognizing the union of the languages recognized by the given automata. So assume $\mathcal{A}$ and $\mathcal{B}$ are finite-state automata on finite words recognizing the languages denoted by r and s, respectively. A Büchi automaton for $r \cdot s^\omega$ is obtained by modifying the disjoint union of $\mathcal{A}$ and $\mathcal{B}$ as follows. First, an additional state q_{new} is added. Second, for every transition $\langle q, a, q' \rangle$ in $\mathcal{A}$ where q' is final, the transition $\langle q, a, q_{\text{new}} \rangle$ is added. Third, for every transition $\langle q, a, q' \rangle$ in $\mathcal{B}$ where q is initial, the transition $\langle q_{\text{new}}, a, q' \rangle$ is introduced. Fourth, for every transition $\langle q, a, q' \rangle$ in $\mathcal{B}$ where q' is final, the transition $\langle q, a, q_{\text{new}} \rangle$ is added. Finally, every final state looses its status as final state; every initial state of $\mathcal{B}$ looses its status as initial state; the state q_{new} becomes the only Büchi state; if one of the initial states of $\mathcal{A}$ was final (the empty word was accepted), then q_{new} becomes initial, too. □

From the above proof, it immediately follows:

Remark 3.2. (1) Every non-empty regular ω-language contains an ultimately periodic word. (2) The emptiness problem for Büchi automata is decidable non-deterministically in logarithmic space, by a simple graph search.

3.2 Co-Büchi and deterministic Büchi automata

Dis- and reassembling runs is a simple but powerful technique in the context of ω-automata, which can, for instance, be used to show that co-Büchi automata and deterministic Büchi automata are weaker than non-deterministic ones:

Proposition 3.3. *(1) The language denoted by* $((0+1)^*0)^\omega$ *cannot be recognized by a co-Büchi automaton. (2) The language denoted by* $(0+1)^*1^\omega$*, which is the complement of the language denoted by* $((0+1)^*0)^\omega$*, cannot be recognized by a deterministic Büchi automaton.*

For the proof of the first part, assume a co-Büchi automaton with n states recognizes the language. Then it accepts the word $(1^n0)^\omega$, say r is an accepting run. There is some k such that all letters in the segment $r[(n+1)k, (n+1)(k+1))$ are co-Büchi states. (As usual, if u denotes an ω-word, then $u[i,j)$ denotes $u(i)u(i+1)\dots u(j-1)$.) Because this segment has $n+1$ positions, there are i and j such that $i < j \leq n$ and $r((n+1)k+i) = r((n+1)k+j)$, which means $r[0,(n+1)k+i)r[(n+1)k+i,(n+1)k+j)^\omega$ is an accepting run of the automaton on $(1^n0)^k1^\omega$—a contradiction.

For the proof of the second part, assume a deterministic Büchi automaton recognizes the language denoted by $(0+1)^*1^\omega$. By complementing its Büchi set and viewing it as a co-Büchi set, one obtains a co-Büchi automaton for the language denoted by $((0+1)^*0)^\omega$—a contradiction to (1). □

Proposition 3.3 shows that the complementation procedure known from finite-state automata—first determinize, then negate the "final condition"—does not work for Büchi automata, because the following two transformations are not possible in general: from a non-deterministic Büchi automaton to an equivalent deterministic Büchi automaton; from a deterministic Büchi automaton to a deterministic Büchi automaton for the complement of the language recognized. There are fundamental differences between ω-automata and ordinary ones.

As complementation and determinization are important operations on automata in general, much of the work on ω-automata deals with them and so does this chapter. Büchi was the first to show that non-deterministic Büchi automata are closed under complementation [5]; Safra's construction [37] was the first with a worst-case state complexity of $\theta(n)^n$, which is optimal [26]. The first determinization construction for Büchi automata, transforming a non-deterministic Büchi automaton into an equivalent forward deterministic Rabin automaton, was given by McNaughton [25]; again, Safra's construction was the first with a worst-case state complexity of $\theta(n)^n$, which is, again, optimal [22]. The development with regard to complementation until the year 2007 is described very nicely in [47].

"The Büchi Complementation Saga"

4 Basic constructions

In this section of introductory technical nature, some important basic constructions are described. They exhibit parallels to the situation with finite words, but demonstrate also distinctive features of ω-automata.

4.1 Products of Büchi automata

A simple operation known from non-deterministic finite-state automata is the disjoint union of two automata, which yields a non-deterministic automaton recognizing the union of the two languages recognized by the two given automata. This works for ω-automata exactly in the same way, provided the two automata have recurrence conditions of the same type.

Another simple operation known from finite-state automata is the product of two automata: it can be used to construct an automaton recognizing the intersection or the union of the two languages recognized by the given automata. The adaptation to ω-words is possible, but not straightforward. In particular, if the recurrence condition is a Büchi condition, the problem arises that Büchi states may not be visited simultaneously, which, in the worst case, may result in an automaton not accepting a single word, while the intersection of the two languages may be the set of all ω-words over the given alphabet.

The problem can be overcome by adding one bit to the state space. More precisely, a state in the adjusted product is of the form $\langle q_0, q_1, b\rangle$, where $b \in [2]$. The transition relation is chosen in such a way that $b = 1$ if there was a prior position with a Büchi state in the second component, but no position in between with a Büchi state in the first component. The word is accepted if a state $\langle q_0, q_1, 1\rangle$ with $q_0 \in B_0$ occurs infinitely often, that is, $B_0 \times Q_1 \times \{1\}$ is the Büchi set.

More precisely, assume there is a transition $\langle q_0, a, q_0'\rangle$ in the first automaton and a transition $\langle q_1, a, q_1'\rangle$ in the second one. This gives rise to a transition from $\langle q_0, q_1, b\rangle$ to $\langle q_0', q_1', b'\rangle$, where b' is defined by: if $q_1 \in B_1$, then $b' = 1$; if $b = 1$ and $q_0 \in B_0$ and $q_1 \notin B_1$, then $b' = 0$; in all other cases, $b' = b$.

The constructed automaton is forward deterministic, provided the given automata are forward deterministic. When the transitions are reversed and the given automata are backward deterministic, then it is backward deterministic [7]. The construction can also be used to turn a generalized Büchi automaton into an ordinary one, resulting in an automaton with kn states, assuming the given automaton has n states and k Büchi sets.

4.2 Automata with output and cascades

In various situations, it is very helpful to consider ω-automata with output, which have an extra output function $\lambda\colon \Delta \to O$, where O is some alphabet. The relation defined by such an automaton is the relation between A^ω and O^ω which contains a pair $\langle u, v\rangle$ if there is an accepting run r of the automaton on u such that $v(i) = \lambda(\langle r(i), u(i), r(i+1)\rangle)$ for every i.

A simple example is the relation which holds between a binary tree and an ω-word if, and only if, the prefixes of the ω-word form an infinite rooted path in the tree. When the trees considered are of width at most k, then the states of a suitable automaton can be chosen to be elements of $[k]$, representing the vertex on the current level which is chosen to be part of the path. There is a transition $\langle i, a, j\rangle$ if vertex j is a successor of vertex i in slice a, and $\lambda(\langle i, a, j\rangle) = 0$ if j is a left successor and else $\lambda(\langle i, a, j\rangle) = 1$.

Just as in the theory of finite-state automata on finite words, an ω-automaton with output can be composed with an ω-automaton (with or without output) by using the output

of the first automaton as input for the second automaton—one speaks of a cascade or of cascading. For instance, if the above automaton is cascaded with a Büchi automaton recognizing $(1^*0)^\omega$, then the resulting automaton recognizes the trees that have a rooted path with infinitely many left successors.

When cascading two finite-state automata on finite words a state of the resulting automaton is a pair consisting of a state of the first automaton and a state of the second one; when cascading ω-automata one needs to be careful about the recurrence condition. For instance, if two Büchi automata are cascaded, then the result can be chosen to be a generalized Büchi automaton or a Büchi automaton with a third component, consisting of a single bit, for combining the two Büchi recurrence conditions as described in Section 4.1.

4.3 The breakpoint construction

A more important example for a Büchi automaton with output is an automaton which defines the function that maps each leveled DAG to the subgraph which is composed of the finitary vertices of the DAG, or, dually, the infinitary vertices. A vertex is called finitary if it has only a finite number of descendants, else it is called infinitary. For DAG's of finite width, which we only consider, being infinitary is equivalent to being on an infinite path.

A Büchi automaton can guess which vertices on a level of a given DAG are finitary and which are not, and it can check that vertices guessed infinitary are indeed infinitary ones by forcing, via the transition relation, each infinitary vertex to have a successor. The problem is to verify that every vertex guessed finitary is indeed finitary. All successors of a vertex guessed finitary must be guessed finitary and this can be enforced by the transition relation, but this condition is only necessary and not sufficient.

To solve the problem a construction referred to as breakpoint construction can be used. A "breakpoint automaton" works in phases. When a phase starts, all vertices on the current level guessed to be finitary are stored in some set, the verification set. During a phase, the verification set is updated from level to level by replacing the vertices in it by their successors. If the guesses were correct, the verification set becomes empty at some point and the phase ends successfully. During a phase, all vertices newly guessed finitary are stored in some other set—they are put on hold. When a new phase starts, the vertices put on hold previously are moved into the verification set. For all guesses to be correct, every phase has to end successfully.

A state of a breakpoint automaton is of the form $\langle V, H\rangle$, where V is the current verification set and H is the set of states currently put on hold. A transition is of the form $\langle\langle V, H\rangle, a, \langle V', H'\rangle\rangle$ and must satisfy the following conditions, which are all phrased with respect to the slice a being read:

- The sets V and H are disjoint sets of vertices of the upper level.
- The sets V' and H' are disjoint sets of vertices of the lower level.
- Every vertex on the upper level not in $V \cup H$ has a successor on the lower level not in $V' \cup H'$.
- If $V \neq \varnothing$, then V' is the set of all successors of the vertices in V, and H' contains at least all the successors of the vertices in H which do not belong to V'.
- If $V = \varnothing$, then V' is the set of all successors of the vertices in H.

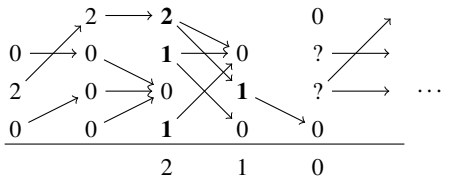


Figure 2. Beginning of a leveled DAG with vertices labeled by their extinction ranks and critical values at the bottom. Lifts are in bold; question marks indicate vertices with ranks which cannot be determined from the visible part of the DAG.

A state is initial if $H = \emptyset$; it is a Büchi state if $V = \emptyset$.

To a transition as above, the output function assigns the slice which is obtained from a by restricting it to the vertices from $V \cup V' \cup H \cup H'$ or, dually, to the other vertices.

Theorem 4.1 (breakpoint construction [27]). *For every k, the breakpoint construction yields a Büchi automaton with 3^k states outputting, for every leveled DAG of width at most k, the subgraph of its finitary [infinitary] vertices.*

The breakpoint construction is used in Sections 5.1, 6.1, 8.4, and 9.5.

4.4 The lift construction

For backward deterministic automata, the previous task—computing the finitary or infinitary vertices in a leveled DAG—can be solved using a construction here referred to as lift construction. If one knows the finitary vertices of a leveled DAG on one level, one can determine the finitary vertices on the previous level in a deterministic fashion: a vertex on the previous level is finitary if, and only if, all its successors are finitary (in particular, a vertex without successors is finitary). The naive approach for constructing a backward deterministic automaton for determining the finitary vertices is to use states which have one bit for every vertex on the current level, indicating whether the vertex is finitary or not, and use the above rule as a backward transition function. The problem is that this construction may overapproximate, because when in a run on a graph with only infinitary vertices all vertices are assumed to be finitary the above rule is obeyed.

To overcome the problem it is important to realize that for every finitary vertex v, say on level l, there is some smallest level $i > l$ without a descendant of v. We call this level the extinction level of v, denote it by $\mathrm{el}(v)$, and use it to rank v.

To this end, assume a vertex v is on some level l. The rank of v measures how difficult it is to get from v to its extinction level, more concretely, how "wide" the part of the run DAG is which one needs to pass by while moving from v to its extinction level. For every level i between l and the extinction level of v, that is, for every i with $l \leq i < \mathrm{el}(v)$, let W_i contain all extinction levels of vertices on level i, but only the ones which are before the extinction level of v, that is, $W_i = \{\mathrm{el}(w) \mid w \in V^{(i)} \text{ and } \mathrm{el}(w) < \mathrm{el}(v)\}$. The maximum of the cardinalities of the W_i's is the extinction rank of v and denoted $\mathrm{er}(v)$. Formally, $\mathrm{er}(v) = \max\{|W_i| \mid l \leq i < el(v)\}$. For an illustration, see Figure 2.

Remark 4.2. (1) The extinction rank of a vertex in a leveled graph of width at most k is at most $k - 1$. (2) On every maximum path starting in a finitary vertex the extinction rank is monotone descending and eventually reaches 0.

In the lift construction, the backward deterministic automaton determines, for each vertex, its extinction rank, that is, a state of the automaton maps each vertex on the current level to a value in $[k]\cup\{\infty\}$, where ∞ is used for infinitary vertices. It is possible to define a backward deterministic transition function accordingly, because the extinction ranks of the vertices on one level can be determined from the structure of the respective slice and the extinction ranks of the vertices on the next level, as described in what follows.

Let U be the set of vertices on the upper level of a slice a. For every $v \in U$, let m_v be the maximum of all values $\mathrm{er}(v')$ where v' is a successor of v in the slice a; by convention, $m_v = -1$ if v has no successor. If there is no $v \in U$ with $m_v = -1$, then $\mathrm{er}(v) = m_v$ for all $v \in U$. If there is such a v, then the largest number c such that $[c] \subseteq \{m_v \mid v \in U\}$ is called the critical value of the upper level. For every $v \in U$ with $m_v \geq c$, the equation $\mathrm{er}(v) = m_v$ still holds. For every other $v \in U$, the values are "lifted": $\mathrm{er}(v) = m_v + 1$.

If extinction ranks are used, then an overapproximation as described above can be avoided by adding an appropriate recurrence condition, more precisely, a generalized transition-Büchi condition. For every rank i, there is a transition-Büchi set B_i which includes all transitions in which i does not occur as a value on the upper level or i is less than the critical value (and thus lifted), see Remark 4.2(2) and Figure 2.

Theorem 4.3 (lift construction [7]). *The lift construction yields, for every k, a backward deterministic generalized transition-Büchi automaton with at most $(k+1)^n$ states outputting, for every leveled DAG of width at most k, the subgraph of its finitary [infinitary] vertices.*

Note that the above approach is very versatile. If, for instance, one wants to determine the vertices which have at least one descendant with no successors, which one could call weakly finitary vertices, then one can take the same approach, replacing maximization by minimization.

The lift construction is used for different purposes in Section 5.2.

4.5 Latest appearance records

Given an alphabet A and a special symbol \$ not in A, the latest appearance automaton (LAA) is a forward deterministic automaton with states being words over $A \cup \{\$\}$ where every letter from A occurs at most once and \$ occurs exactly once. One such word is called a latest appearance record (LAR) and the part to the right of "\$" is its frame.

The initial state of the LAA is the one-letter word \$; the recurrence condition is trivial; the transition function δ is defined as follows. When u is a state of the form $v\$v'$ and a is a letter of the alphabet occurring in vv', say $vv' = waw'$, then $\delta(u,a) = w\$w'a$. When a does not occur in vv', then $\delta(u,a) = vv'\$a$. So the order in which the letters occur in the current state of the automaton is the order of their latest appearance in the prefix of the given word read so far, with all letters in the frame of the current state being the ones that have occurred since the previous occurrence of the letter just read. From this, the following can be derived.

Remark 4.4. [6, 17] Consider the frames of maximal length among all frames occurring

infinitely often in the run of the LAA on a given word. Then all theses frames contain the same letters and these are exactly the ones occurring infinitely often in the given word.

An interesting application of the latest appearance record is the transformation of a given Muller automaton into an equivalent parity automaton. First, the Muller condition is removed (and replaced by the trivial recurrence condition). Second, the automaton is augmented by the trivial output function, which simply outputs the current state. Third, the generated automaton is cascaded with the LAA over the state set of the Muller automaton. Finally, assuming the automaton has n states, a priority function is added that assigns each state $\langle q, v\$v'\rangle$ the priority $2n - 2|v'|$ if $\mathrm{occ}(v')$ is a Muller set and else $2n - 2|v'| + 1$.

Theorem 4.5 (Muller to parity). *[28] For every forward deterministic [non-deterministic] Muller automaton with n states there is an equivalent forward deterministic [non-deterministic] parity automaton with $(n+1)!$ states and at most $2n$ priorities.*

A refined construction, saving priorities if possible, is presented in Section 8.2.

5 Run DAG's of Büchi automata

Büchi automata, in general, are non-deterministic automata, in other words, there may be several runs of a given Büchi automaton on a given word. These runs have to be considered at the same time if, for instance, one wants to turn a Büchi automaton into a Büchi automaton for the complement of the language recognized, because not to accept means *all* initial runs are not recurrent.

There are essentially two global structures that have been investigated for arranging all runs of a Büchi automaton in a concise way: DAG's and trees. The former are treated in this section, the latter in the next one. Applications are complementation, determinization, and disambiguation (defined in Section 6.1).

Assume a Büchi automaton is given. The run DAG of a given ω-word u is the leveled graph with levels $\{Q \times \{i\}\}_{i\in\omega}$ and edges $\langle\langle q, i\rangle, \langle q', i+1\rangle\rangle$ for $\langle q, u(i), q'\rangle \in \Delta$. Its width is the number of states of the given automaton.

Often, it is useful to think of a run DAG as a graph labeled with elements from Q; in this section, it is sufficient to think of it as providing only information about whether the state component of a vertex is an initial or a Büchi state. Technically, the DAG is labeled with elements from $\wp(\{I, B\})$ and we say it is $\{I, B\}$-tagged; if a vertex is labeled with a letter a and $I \in a$, we say it is I-tagged, and, analogously, if $B \in a$, we say it is B-tagged.

A vertex of an $\{I, B\}$-tagged DAG is called B-recurring if a path with an infinite number of B-tagged vertices starts in it; it is called B-free if none of its descendants (including itself) is B-tagged. The ultimate width of such a DAG is the limes inferior of the number of non-B-recurring infinitary vertices on a given level.

Remark 5.1. An ω-word is accepted by a Büchi automaton if, and only if, there is an I-tagged B-recurring vertex on level 0 of the run DAG of the word.

The main insight needed about $\{I, B\}$-tagged DAG's (or simply $\{B\}$-tagged DAG's) of finite width is that they can be decomposed in a simple manner. Consider the following operation, here called peeling. First, remove all finitary vertices; second, remove all B-free vertices. Peeling does not remove any B-recurring vertex, and if it does not change the DAG at all, then all vertices are B-recurring, because every vertex has a strict B-tagged descendant. Moreover, if there are non-B-recurring infinitary vertices, then peeling decreases the ultimate width by at least one, as explained in what follows.

Consider a non-B-recurring infinitary vertex. By König's lemma [21], there is an infinite path starting in it. Assume that every B-tagged strict descendant of the vertex is finitary. Then, after removing the finitary vertices, each successor of the vertex is B-free, but the infinite path is still there and all of its vertices (except, maybe, the first one) are removed in the second step, decreasing the ultimate width by one. If there is a strict B-tagged infinitary descendant of the vertex, apply the same argument to it. This cannot go ad infinitum, because a path with an infinite number of B-tagged vertices would be constructed.

This all implies:

Lemma 5.2 (peeling [20]). *For every Büchi automaton with n states, peeling the run DAG of any ω-word n times yields the subgraph induced by the B-recurring vertices.* □

This can be used in various ways, in particular, it can be used for complementing Büchi automata, see Section 5.1, determinizing them backward, see Section 5.2, and showing that alternating Büchi automata can easily be converted into weak alternating automata, see Section 9.5.

To describe these applications, it is useful to have some notation and terminology at hand. By the above, each vertex v in a $\{B\}$-tagged DAG of finite width can be assigned a value in $\omega \cup \{\infty\}$ according to when the vertex is removed by peeling the DAG successively. More precisely, when i is a natural number and all vertices with value $< 2i$ are removed from the given DAG, the finitary vertices in the remaining DAG get assigned $2i$; when all vertices with value $< 2i + 1$ are removed, the B-free vertices in the remaining DAG get assigned $2i + 1$. The B-recurring vertices get assigned ∞. The number assigned to a vertex v is called its canonical rank, it is denoted $c(v)$, and, according to the above, it is ∞ or $< 2n$, when n is the width of the DAG.

Corollary 5.3. *For a Büchi automaton with n states, let c be the canonical rank function of the run DAG of some ω-word. (1) The word is accepted if, and only if, $c(v) = \infty$ for some I-tagged vertex v on level 0. (2) Equivalently, the word is not accepted if, and only if, $c(v) < 2n$ for every I-tagged vertex v on level 0.*

5.1 Complementation via canonical ranks

The idea of using ranks or "progress measures" for complementing ω-automata goes back to [19] and has been improved and refined over the years, especially in [20]. The basic idea is to implement Corollary 5.3(2). The starting point is a compilation of properties of the canonical rank function of a given $\{I, B\}$-tagged leveled DAG.

Klarlund, 1991 Kupferman, Vardi, 2001

Property 5.4. Let v be any vertex. If v does not have any successor, let $M = 0$, else let M be the maximum of all values $c(v')$ for successors v' of v. If v is not B-tagged or if M is even, then $c(v) = M$; if v is B-tagged and M is odd, then $c(v) = M + 1$.

Property 5.5. For any vertex with an even rank, the number of its descendants with the same rank is finite.

In general, a rank function of a leveled DAG of width n with vertex set V is a function $f\colon V \to [2n]$ satisfying Properties 5.4 and 5.5 with f instead of c.

Remark 5.6. Any rank function is pointwise greater or equal to the canonical rank function.

A complementation construction for Büchi automata can now be based on Corollary 5.3(2) and the following observations. First, there is a forward deterministic automaton with trivial recurrence condition that outputs the part of the $\{I, B\}$-tagged run DAG of a given word which is reachable from the I-tagged vertices on level 0. Second, there exists a non-deterministic Büchi automaton that produces for every $\{I, B\}$-tagged leveled graph of width at most n the same graph, but with any labeling with numbers from $[2n]$ such that Property 5.4 is satisfied. Third, using a variant of the breakpoint construction, see Theorem 4.1, a Büchi automaton can be constructed that checks Property 5.5 for a $[2n]$-labeled DAG. In other words, a suitable cascade yields a Büchi automaton for the complement of the language recognized by a given Büchi automaton.

Theorem 5.7 (complementation via ranks [20]). *Complementation via canonical ranks yields, for every Büchi automaton with n states, a Büchi automaton with at most $(6n)^n$ states.*

Friedgut, Kupferman, Vardi, 2006

In [15], the above approach is improved, resulting in an asymptotic upper bound of $(0.96\,n)^n$ for the number of states, and in [39] a further improvement leads to a construction which is optimal within a factor of $O(n^2)$ and has an asymptotic upper bound of $(0.76n)^n$.

Schewe, 2009

5.2 Backward determinization via canonical ranks

A second application of canonical ranks is the conversion of a given non-deterministic Büchi automaton into an equivalent backward deterministic generalized transition-Büchi automaton. The idea, which is due to [7], is to use use Corollary 5.3(1) and to construct an automaton which labels the run DAG in a backward deterministic fashion with the values of the canonical rank function.

The key to designing such an automaton is the fact that the canonical rank function is the only function on an $\{I, B\}$-tagged DAG satisfying Property 5.4 and the following one, Property 5.8.

Property 5.8. (1) For every even number i, the set $c^{-1}(\{i\})$ is exactly the set of finitary vertices in the sub-DAG without the vertices in $c^{-1}([i])$. (2) For every vertex $v \in V$, if $c(v) > 1$ and $c(v)$ is odd, then v has a descendant v' with $c(v') = c(v) - 1$. (3) In the sub-DAG consisting of the vertices in $c^{-1}(\{\infty\})$, every vertex has a strict B-tagged descendant.

This means a backward deterministic generalized transition-Büchi automaton computing the rank function for a run DAG can be constructed as a cascade of two automata: (i) an automaton with a backward deterministic transition function and a trivial recurrence condition outputting the run DAG of a given word and an assignment to the vertices satisfying Property 5.4; (ii) a backward deterministic automaton checking Property 5.8 using adaptations of the lift construction, see Theorem 4.3 and also the subsequent remark on weakly finitary vertices. An automaton equivalent to the given Büchi automaton is obtained when the states which assign ∞ to an I-tagged vertex are chosen to be initial.

Theorem 5.9 (backward determinization via canonical ranks [7]). *Backward determinization via canonical ranks yields, for every Büchi automaton with n states, a generalized transition-Büchi automaton with at most $(3n)^n$ states.*

6 Run trees of Büchi automata

Run DAG's are one way to represent the set of all runs of a Büchi automaton on an ω-word. A different approach, which can serve as a basis for complementation, disambiguation, and forward determinization, is to use compressed run trees.

In a first step towards the definition of the compressed run tree for a given word u with respect to a given Büchi automaton, a labeled binary tree t is defined, using a refined subset construction. Just as in the subset construction, all the states reachable from the initial states are tracked at the same time. The difference is that in each step the set of states reachable by reading the next letter is split into the Büchi states and the non-Büchi states: a binary tree emerges. To keep this tree compact, only one occurrence of each state—more precisely, its leftmost occurrence—is kept, that is, the tree is pruned in a straightforward fashion.

—> Split tree, this is what Nitsche's construction

—> Reduced split tree

In the following, when a vertex v is said to be to the left of another vertex v', then this means that v and v' are on the same level, that is, $|v| = |v'|$, and there exists $i < |v|$ such that $v(j) = v'(j)$ for all $j < i$, $v(i) = 0$, and $v'(i) = 1$. The corresponding ordering is denoted by $<_{\mathrm{lft}}$.

The definition of t is by induction on the levels. The root (level 0) of t is labeled with Q_I. Assume all vertices on level l have already been constructed and assigned labels, say these vertices form the set W, and let a stand for the next letter, $u(l)$. For each $v \in W$, let $R_v = \bigcup\{\Delta(q,a) \mid q \in t(v)\}$, which means R_v is the set of states reached from any state in the label of v by reading a. Set

$$Q_v^0 = (R_v \cap B) \setminus \bigcup_{w <_{\mathrm{lft}} v} R_w \ , \qquad Q_v^1 = R_v \setminus (B \cup \bigcup_{w <_{\mathrm{lft}} v} R_w) \ . \tag{6.1}$$

accepting states

non-accepting states

In other words, Q_v^0 is the set of Büchi states reached from $t(v)$ by reading letter a, but not including the states that are reached from any state in a label of a vertex to the left of v. Similarly, Q_v^1 is the set of non-Büchi states reached from $t(v)$ by reading letter a, but not including the states that are reached from any state in a label of a vertex to the left of v. The definition of the tree t now says that, for $v \in W$ and $i < 2$, if $Q_v^i \neq \emptyset$, then $vi \in t$ and $t(vi) = Q_v^i$. For an illustration, see Figure 3.

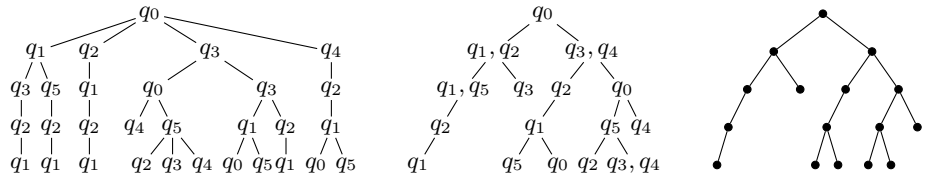


Figure 3. The first five levels of the ordinary run tree of a word; the corresponding part of the labeled compressed run tree; the corresponding part of the compressed run tree. For ease in notation, curly brackets to denote sets are omitted.

The tree t is called the labeled compressed run tree of u, while the unlabeled tree, that is, the domain of t, is called the compressed run tree of u and denoted T.

Lemma 6.1. *A Büchi automaton accepts an ω-word if, and only if, its compressed run tree has a path with infinitely many left successors, also called left-recurring path.*

For the proof, assume u is a word which is accepted by a given Büchi automaton. We construct an infinite path $\langle v_0, v_1, \dots \rangle$ in t in such a way that for every l the following conditions hold. (i) There exists a recurring run on $u[l, \infty)$ starting in some $q \in t(v_l)$. (ii) There is no vertex $v <_{\text{lft}} v_l$ satisfying the same condition. For the induction base, we choose $v_0 = \epsilon$, which obviously works. Assume v_l has already been defined. By (i), there is a recurring run on $u[l, \infty)$ starting in some state of $t(v_l)$. If there is such a run r with $r(1) \in B$, we set $v_{l+1} = v_l 0$ or else $v_{l+1} = v_l 1$. To show $v_{l+1} \in T$, we fix a state $q \in t(v_l)$ and a recurring run r on $u[l+1, \infty)$ such that qr is a recurring run on $u[l, \infty)$. By way of contradiction, assume $v_{l+1} \notin T$. Then there are a vertex $v <_{\text{lft}} v_l$ and a state $q' \in t(v)$ such that $r(0) \in \Delta(q', u(l))$. This means $q'r$ is a recurring run on $u[l, \infty)$—a contradiction to (ii).

Assume the constructed path is not left-recurring. Then there exists i such that $v_j \in (0+1)^*1$ for all $j \geq i$. Let r be a recurring run on $u[i, \infty)$ starting with a state in $t(v_i)$. Then there is some $k > 0$ such that $r(k)$ is a Büchi state. By adjusting i, we can assume $k = 1$. If, on one hand, $r(1) \notin t(v_{i+1})$, we can obtain a contradiction similar to above. If, on the other hand, $r(1) \in v_{i+1}$, then, by definition, $v_{i+1} = v_{i+1-1}0$—a contradiction.

For the converse, assume $\langle v_0, v_1, \dots \rangle$ is a left-recurring path in T. For every $i < \omega$ and every $q \in t(v_i)$, there is an initial run of the automaton on $u[0, i)$ such that $r(i) = q$ and $r(j) \in t(v_j)$ for every $0 < j < i$. All these runs can be organized in a straightforward fashion in an infinite tree with branching degree at most the number of states of the given automaton. By König's lemma, this tree has an infinite rooted path, and, by construction, the labeling of this path is an initial run of the automaton on u. Further, for every $i > 0$, the state in position i of this run belongs to B if, and only if, v_i is a left successor. □

From an automata-theoretic point of view the important observation is that compressed run trees of a given Büchi automaton have width at most n and can be constructed in a forward deterministic fashion by an ω-automaton with output and trivial recurrence condition. One way to realize such an automaton is to use states of the form $\langle Q_0, \dots, Q_{m-1} \rangle$ with the Q_i's being pairwise disjoint, non-empty subsets of Q, representing the labeling of the current level of the labeled compressed run tree. An upper bound on the number of such states can be obtained using ordered Bell numbers.

Remark 6.2. For every Büchi automaton with n states, there is a forward deterministic automaton that outputs, for every ω-word, a representation of its compressed run tree, and has a number of states which is asymptotically bounded from above by $(0.54\,n)^n$.

6.1 Complementation and disambiguation via compressed run trees

Compressed run trees can be used for complementation. To see this, consider the subtree of a compressed run tree which contains only the infinitary vertices and call it the core of the run tree. From Lemma 6.1 and the fact that the run tree has finite width it follows that an ω-word is not accepted if, and only if, in the core of its run tree there are only finitely many slices with a left successor. In other words, a Büchi automaton for the complement is obtained as a cascade of the following automata: (i) the automaton from Remark 6.2; (ii) an automaton based on the breakpoint construction removing the finitary vertices; (iii) a two-state automaton checking that from some level onward, no slice with a left successor occurs anymore. A careful implementation leads to state spaces similar in size to those described in [39].

Schewe, 2009

An interesting observation is that the cascade of the first and the second automaton from above yields a non-deterministic Büchi automaton which, for every ω-word over the given alphabet, outputs the core of its compressed run tree and has exactly one accepting run. In general, an automaton which has at most one accepting run for each word is called an unambiguous automaton. In other words, the above automaton is an unambiguous automaton for the set of all ω-words over the given alphabet. Moreover, it can be modified in two ways. (i) By cascading it with a two-state deterministic automaton checking that there are infinitely many slices with left successors, one obtains an unambiguous automaton for the language recognized by the given automaton. (ii) By cascading it with a two-state unambiguous automaton checking that there are only finitely many slices with left successors, one obtains an unambiguous automaton for the complement. In the terminology of [8], such an automaton could be called strongly unambiguous.

6.2 Forward determinization via compressed run and history trees

Theorem 2.1 states in particular that every non-deterministic Büchi automaton is equivalent to a forward deterministic parity, Rabin, Streett, or Muller automaton. The quest for good constructions establishing this—determinization constructions—has resulted in different approaches. The approach followed in this section is motivated by [31], but it is also closely related to Safra-like constructions, as explained towards the end.

Müller, Schupp, 1987

In view of Lemma 6.1 and Remark 6.2, a determinization construction has been established once it has been shown that the set of all binary trees of width at most k which have a left-recurring path is recognized by a forward deterministic automaton. Therefore, the objective in what follows is exactly to describe such an automaton.

To understand the mechanics of infinite trees of finite width better, we associate with every vertex v in a binary tree its origin. This is the earliest ancestor of v (shortest prefix of v) with the property that no vertex to the right of v has the same ancestor; it is denoted $\mathrm{or}(v)$. For an illustration, see Figure 4.

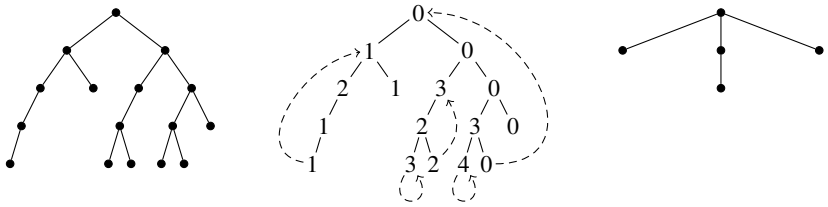


Figure 4. The first five levels of a compressed run tree; the same tree with the origin numbers of each vertex and the origins of the vertices on level 4; the tree induced by the origins of the vertices on level 4. There are two origins that move left in the last slice depicted.

Observe, for instance, that (i) the root is the origin of the rightmost vertex on each level; (ii) vertices on the same level have distinct origins; (iii) if a vertex has a left and a right successor, then the left successor is its own origin.

The important definition specifies that an origin moves left in one slice if it is the origin of a vertex v on the upper level of the slice and of a vertex v' on the lower level of the slice and v' is not the right successor of v, that is, v' is to the left of $v1$. (Note that, by definition, v' cannot be to the right of $v1$.) For an illustration, see Figure 4.

The key for the construction to be presented is:

Lemma 6.3. *A binary tree of finite width contains a left-recurring path if, and only if, there is some origin which moves left in infinitely many slices.*

To prove the lemma, assume $\langle v_0, v_1, \dots \rangle$ is a left-recurring rooted path. Let i be minimal such that there is no other infinite path $\langle v_0, \dots, v_j, v_j 1, \dots \rangle$ for any $j \geq i$. (This number i exists because there are at most k rooted infinite paths in any tree of width k.) For every $j \geq i$, consider the rightmost vertex w_j on level j such that $v_j \leq_{\text{lft}} w_j$ and there is no infinitary vertex v' with $v_j <_{\text{lft}} v' \leq_{\text{lft}} w_j$. Then v_i is the origin of every w_j and moves left in infinitely many slices.

For the converse, let v be an origin which moves left in infinitely many slices. Consider, for every level $i \geq |v|$, the vertex w_i with origin v. By König's lemma, there is a rooted path $\langle v_0, v_1, \dots \rangle$ in the tree which consists of all vertices w_i and their ancestors. This path is left-recurring, because otherwise there would be some j with $v_k \in v_j 1^*$ for all $k \geq j$, which is a contradiction to v moving left in infinitely many slices. □

There are several ways for an automaton to check whether there is an origin which moves to the left in infinitely many slices. One is explained in what follows, another one is sketched later.

We use the notion of military ordering, denoted $<_{\text{mil}}$, and defined by $v <_{\text{mil}} v'$ if, and only if, either $|v| < |v'|$ or $|v| = |v'|$ and $v <_{\text{lft}} v'$.

From level to level, the automaton determines, for each vertex, its origin number, which is defined as follows. For a given level l, let $v_0, \dots, v_{r-1}$ be an enumeration of the vertices on level l, ordered according to the military order of their origins, that is, $\text{or}(v_0) <_{\text{mil}} \cdots <_{\text{mil}} \text{or}(v_{r-1})$. The index i is the origin number of v_i and denoted $\text{on}(v_i)$. The index i is said to refer to the origin $\text{or}(v_i)$ on level l. For an illustration, see Figure 4.

When a forward deterministic automaton computes the origin numbers, it can be augmented by a transition-Rabin condition to check for a left-recurring path. To see this, as-

sume v is an origin on some level l that moves infinitely often to the left and let $v_0, v_1, \ldots$ be the list of vertices such that v_i is on level $l+i$ and $\mathrm{or}(v_i) = v$, in particular, $v_0 = v$. Then there is some i such that $\mathrm{on}(v_i) = \mathrm{on}(v_{i+1}) = \ldots$. So an appropriate transition-Rabin recurrence condition can be chosen to have, for each $i < k$ (maximum width of the trees considered), a Rabin pair $\langle L_i, U_i \rangle$ as follows. The set L_i contains all transitions where for every $j \leq i$ the index j does not refer to the same origin in the upper and the lower level of the current slice. The set U_i contains all transitions where the origin which i refers to on the upper level moves left in the slice.

As states of a forward deterministic automaton computing the origin numbers one can choose bijections $g\colon [m] \to [m]$ where $m \leq k$. The meaning of such a state g would be that if v is vertex i on the current level in the order from left to right, then $\mathrm{on}(v) = g(i)$. The actual definition of the transition function is somewhat technical and omitted.

In the determinization constructions presented in [37, 33, 40], states are trees. For instance, in [40], so-called history trees are used and defined as trees where (i) each node is labeled by a non-empty set of states, (ii) the label of every node is a strict superset of the union of the labels of its children, and (iii) the labels of siblings are disjoint. Observe that, alternatively, one could require that the vertices of a history tree are labeled with pairwise disjoint, non-empty sets of states. Such a tree is obtained from the labeled compressed run tree of a given word for each level in a straightforward fashion: move every label of a vertex on the respective level to its origin and then remove all vertices except for these origins and contract edges accordingly. For an illustration, see Figure 4.

If one constructs a deterministic automaton which only keeps track of the history trees just described, one arrives at a fundamental construction, which is also known to be optimal in a certain sense [9]:

Theorem 6.4 (determinization via history trees [40]). *Determinization via history trees yields, for every Büchi automaton with n states, an equivalent deterministic transition-Rabin automaton with an asymptotic upper bound of $(1.65\,n)^n$ for the number of states and $2^n - 1$ Rabin pairs.*

The transformations on different types of ω-automata discussed in this section and the previous one are fundamental transformations, but not all one can consider. Optimal solutions for most of the basic transformation tasks can be found in [37] and a later paper by the same author [38]. Here, "optimal" means with respect to a rough measure of complexity: polynomial, exponential, doubly exponential. The "optimal" results stated subsequent to Theorem 5.7 and before Theorem 6.4 are with respect to much finer measures and based on very good lower bounds. A breakthrough with regard to lower bounds on ω-automata is [51].

7 Congruence relations

Congruence relations are a useful tool for working with finite-state automata on finite words. For instance, the minimum-state deterministic finite-state automaton for a given regular language of finite words can be derived from the Myhill-Nerode congruence rela-

tion for the language—it merely is this congruence relation.

For ω-automata, congruences are also useful, but the situation is more complex.

7.1 Right (and left) congruence relations

The straightforward adaptation of the Myhill-Nerode congruence relation ??? to ω-languages is the initial syntactic congruence relation. For a given ω-language L, it considers finite words u and v equivalent if, and only if, for every ω-word w, either $\{uw, vw\} \subseteq L$ or $\{uw, vw\} \cap L = \varnothing$.

On the automata-theoretic side, there is a corresponding notion. Given an ω-automaton of any type, its initial congruence relation considers finite words u and v equivalent if, and only if, for every initial run of the automaton on u ending in some state q there is such a run on v, and vice versa.

The analogy to the Myhill-Nerode congruence relation is as follows.

Remark 7.1. (1) The initial syntactic congruence relation for an ω-language is a right congruence relation. (2) For an ω-automaton of any type, the initial congruence relation is a right congruence relation and equal to or finer than the initial syntactic congruence relation for the language recognized and has a finite number of congruence classes.

(Here, as usual, an equivalence relation on words over a given alphabet is a right congruence relation if uw and vw are equivalent whenever u and v are and w is a finite word over the alphabet. A relation is finer than another one if it is a subset of it.)

For a simple language such as the set of all ultimately periodic words over a given alphabet, which is not regular, the initial syntactic congruence relation has only one equivalence class. Hence, it cannot serve as a vehicle to define regularity. It can neither be used for classifying regular ω-languages: the languages denoted by $(0 + 1)^\omega$ and by $(0^*1)^\omega$ have the same initial syntactic congruence relation, but they are completely different in nature.

The initial syntactic congruence relation can, however, take over the role of the Myhill-Nerode congruence relation for the small class of ω-languages which are recognized by forward deterministic weak automata (see also Section 8.4):

Theorem 7.2 (minimization of forward deterministic weak automata [42, 23]). *Let L be an ω-language recognized by a forward deterministic weak automaton $\mathcal{A}$ and let $\mathcal{D}$ be the DFA (without final state set) corresponding to the initial syntactic congruence relation for L.*

(1) The automaton $\mathcal{D}$ can be augmented by a weak acceptance condition in such a way that the resulting automaton recognizes L.

(2) The automaton from (1) is, up to isomorphism, the smallest forward deterministic automaton recognizing L and can be computed from $\mathcal{A}$ by DFA minimization ???.

The role that the initial syntactic congruence relation of a language L plays in the context of forward deterministic automata is taken over by the final syntactic congruence relation in the context of backward deterministic automata. This relation, which is a left congruence relation, considers ω-words v and w congruent if, and only if, for every finite word u, either $\{uv, uw\} \subseteq L$ or $\{uv, uw\} \cap L = \varnothing$.

7.2 Two-sided congruence relations

There are essentially two straightforward adaptations of the two-sided syntactic congruence relation for languages of finite words ??? to ω-languages.

Let L be an ω-language over some alphabet A. In the first adaptation, finite words u and v are congruent if, and only if, for all $x \in A^*$ and $y \in A^\omega$, either $\{xuy, xvy\} \subseteq L$ or $\{xuy, xvy\} \cap L = \varnothing$. In the second adaptation, non-empty finite words u and v are congruent if, and only if, whenever $u_0, u_1, \ldots$ and $v_0, v_1, \ldots$ are sequences of non-empty finite words over the given alphabet such that $u_i = v_i$ or $\{u_i, v_i\} \subseteq \{u, v\}$, then either $\{u_0u_1\ldots, v_0v_1\ldots\} \subseteq L$ or $\{u_0u_1\ldots, v_0v_1\ldots\} \cap L = \varnothing$.

The two adaptations try to capture what it means for two finite words to behave equally in the same context, and they both yield two-sided congruence relations. The first one is finer or equal to the initial syntactic congruence relation; the second one is finer or equal to the first one and called the syntactic congruence relation of L.

In the following, the syntactic congruence relation is further discussed, because it provides a finer means of characterization.

Corresponding to the syntactic congruence relation one can define, for every ω-automaton of any type, a suitable two-sided congruence relation. For a Büchi automaton, this relation considers non-empty finite words u and v congruent if the following two conditions hold for all states $q, q' \in Q$. (i) There is a run of the automaton on u starting with q and ending in q' if, and only if, this is true for v. (ii) There is a run r of the automaton on u starting with q, ending in q', and passing through an element of B, that is, $r(i) \in B$ for some $i < |r|$, if, and only if, this is true for v.

Just as before, the syntactic congruence relation of an ω-language does not characterize regularity. Consider[2] the language L_{ub0} of all ω-words of the form $0^{i_0}10^{i_1}\ldots$ where $\limsup_{j\to\infty} i_j = \infty$. The language L_{ub0} and the one denoted by $(0^*1)^\omega$ have the same syntactic congruence relation, and this has just two equivalence classes.

Still, the syntactic congruence relation has interesting properties. One is phrased in terms of saturation, where an equivalence relation on finite words is said to saturate an ω-language L if, for all sequences $\langle V_0, V_1 \ldots\rangle$ of equivalence classes, either $V_0V_1\cdots \subseteq L$ or $V_0V_1\cdots \cap L = \varnothing$ holds.

Theorem 7.3 (saturation [5, 1]). *(1) The two-sided automaton congruence relation of a Büchi automaton with n states has at most 2^{2n^2} congruence classes and saturates the language recognized by the automaton.*

(2) The syntactic congruence relation of a given regular ω-language is the coarsest congruence relation saturating the language.

(3) An ω-language is regular if, and only if, there exists a congruence relation saturating it and having a finite number of congruence classes.

The proofs of (1) and (2) are straightforward; one direction of (3) follows from (1). The other direction of (3) can be proved on the basis of Ramsey's Theorem A [36], which says that, for a given equivalence relation on finite words with a finite number of equivalence classes and an infinite sequence of non-empty finite words $u_0, u_1, \ldots$, there is a strictly monotone infinite sequence $\langle i_0, i_1, \ldots\rangle$ of natural numbers such that all finite

[2] Slides of a presentation given by Mikołaj Bojańczyk.

words of the form $u_{i_j} u_{i_j+1} \dots u_{i_k-1}$ with $j < k$ are equivalent. In the context of ω-languages, this means the following.

Remark 7.4. [5] Given an alphabet A, a congruence relation on A^+ having a finite number of congruence classes, and $u \in A^\omega$, there are congruence classes U and V satisfying $UV \subseteq U$, $V^2 \subseteq V$, and $u \in UV^\omega$.

To prove the other direction of Theorem 7.3(3), observe that from the previous remark it follows that if a congruence relation with a finite number of equivalence classes saturates a given ω-language L, then $L = \bigcup UV^\omega$ with U and V ranging over equivalence classes with $UV^\omega \subseteq L$. Based on this, a construction such as the one described in the proof of Theorem 3.1 can be used to arrive at a Büchi automaton recognizing L. □

The procedure just described is far from being as natural as the procedure that turns the Myhill-Nerode congruence relation for a given regular language of finite words into the minimum-state DFA for the language. In fact, nothing which would come close to this is known for regular ω-languages. Still, two-sided congruence relations for ω-languages are useful in several contexts, for instance, when it comes to classifying regular ω-languages, see [32]. Another application, described in what follows, is complementation.

7.3 Complementation via two-sided congruence relations

When a two-sided congruence relation saturates a given language, then, by definition, it also saturates the complement of the language, which establishes once again that the class of ω-languages recognized by Büchi automata is closed under complementation. In fact, the first proof of this fact was along these lines.

In view of the bound stated in Theorem 7.3(1), using congruences for complementation leads to much larger automata than the ones described in Sections 5.1 and 6.1. To obtain smaller automata with this approach, the following modification suggests itself.

Write the complement of the language recognized by a given Büchi automaton again in the form $\bigcup_{i<k} U_i V_i^\omega$, but choose the U_i's and V_i's to be unions of congruence classes in a way such that the resulting Büchi automaton is small. Observe that if m is the number of states of an automaton recognizing all U_i's (with a different set of final states for each i) and m' is an upper bound on the number of states needed in automata recognizing the V_i's, then $m + k(m'+1)$ is an upper bound for the number of states in the resulting Büchi automaton.

To see how the indicated approach works, assume a Büchi automaton is given as usual, with n states and recognizing a language L. For every set $P \subseteq Q$, let U_P be set of words u such that every run of the automaton on u starting in some initial state ends in some state of P. Then each set U_P is a union of equivalence classes of the automaton congruence relation and can be recognized by a deterministic automaton with 2^n states.

For every non-empty sequence $\sigma = \langle P_0, \dots, P_{k-1} \rangle$ of non-empty, pairwise disjoint sets of states, let the set $P(\sigma)$ be defined by $P(\sigma) = P_0 \cup \dots \cup P_{k-1}$, and let V_σ be the set of all finite words u satisfying the following two conditions for every run r of the automaton on u. (i) If $r(0) \in P_i$ for some i, then $r(|u|) \in P_j$ for some j with $i \leq j < k$. (ii) If $r(0) \in P_i$ and r contains a Büchi state, then $r(|u|) \in P_j$ for some j with $i < j < k$.

Breuers, Improved Ramsey-Based Büchi Complementation, 2012

Theorem 7.5 (complementation by saturation [4]). *For every Büchi automaton with n states, the language $\bigcup_\sigma U_{P(\sigma)}V_\sigma^\omega$ is the complement of the language recognized by the Büchi automaton, and a conversion of this expression into a Büchi automaton yields an automaton with $2^{\theta(n \log n)}$ states.*

For the proof, first observe that each set V_σ is a union of equivalence classes of the two-sided automaton congruence relation: compare (i) and (ii) above with (i) and (ii) in the definition of the two-sided automaton congruence relation.

Next, let σ and $P(\sigma)$ be as above. To see that $U_{P(\sigma)}V_\sigma^\omega \cap L = \varnothing$ holds, let r be an initial run on any word $u \in U_{P(\sigma)}V_\sigma^\omega$ and $i_0 < i_1 < \dots$ be such that $u[0, i_0) \in U_{P(\sigma)}$ and $u[i_j, i_{j+1}) \in V_\sigma$ for every j. From the definition of V_σ we can conclude that $r(i_0) \in P(\sigma)$ holds and that there are i and k such that $r(i_j) \in P_i$ holds for every $j \geq k$. This implies that, for every $j \geq k$, there is no Büchi state in $r[i_j, i_{j+1})$, which means $u \notin L$.

Conversely, if an ω-word u is not accepted by the given Büchi automaton, then, by Remark 7.4, there are classes U and V of the automaton congruence relation such that $u \in UV^\omega$, $UV \subseteq U$, and $V^2 \subseteq V$. Let P be the set of states which can be reached by reading some word from U from some initial state. Consider the graph with vertex set P and an edge between q and q' if, and only if, there is a run of the automaton on some word $v \in V$ starting in q and ending in q'. Let $\sigma = \langle P_0, \dots, P_{k-1}\rangle$ be a list of the SCC's of this graph in topological order. Then $V \subseteq V_\sigma$, which means $u \in U_{P(\sigma)}V_\sigma^\omega$.

To prove the claim about the size of the resulting ω-automaton, we describe how to construct a deterministic automaton of size $(k+1)^n$ for a language V_σ as above. The states are functions $Q \to \{-\infty\} \cup [k]$. The transition function is defined in a way such that if by reading a finite word v the automaton reaches state f, then the following holds for every $q \in Q$. If in the Büchi automaton there is no run on v starting in $P(\sigma)$ and ending in q, then $f(q) = -\infty$; if there are such runs, then $f(q)$ is the greatest index i such that a run on v starting in some state from P_i ends in q. □

The construction described above can be generalized so as to improve the construction of a Büchi automaton from a saturating congruence relation.

8 Loop structure

As the set of states visited infinitely often in a run of an ω-automaton determines whether the run is recurring, it is only natural to investigate the structure of the strongly connected subsets in a given ω-automaton.

A loop at some state q is a word $q_0a_0q_1a_1 \dots a_nq_{n+1}$ where $\langle q_i, a_i, q_{i+1}\rangle \in \Delta$ for every $i \in [n]$ and $q_0 = q_{n+1} = q$. The word $a_0 \dots a_n$ is the label of the loop, the set $\{q_0, \dots, q_n\}$ is the loop set. The loop is positive if it satisfies the recurrence condition of the given automaton (for a Büchi condition, this means $\{q_0, \dots, q_n\} \cap B \neq \varnothing$), it is negative if it does not—we speak of the sign of the loop. In a deterministic automaton (forward or backward), q and the label determine the loop.

In forward deterministic automata, the nesting depth of positive and negative loops sets is an interesting measure for the complexity of the language recognized, explained in Section 8.1, whereas in backward deterministic automata, the distribution of the labels of

positive loops is interesting, as explained in Section 8.4.

In the following, we assume, without loss of generality, that in forward deterministic automata every state is reachable from the initial state.

8.1 Alternating loops in forward deterministic automata

A tower is a non-empty sequence $\langle C_0, \ldots, C_{m-1} \rangle$ of loop sets such that $C_0 \supseteq \cdots \supseteq C_{m-1}$ and the signs alternate; the sign of the last loop is the sign of the tower, the number m is the height of the tower. A maximal tower is one of maximal height. A wall is a sequence of maximal towers where each one is reachable from the previous one and the signs alternate; the sign of the wall is the sign of the first tower, the number of towers in the sequence is the length of the wall.

The types of towers and walls in a given forward deterministic ω-automaton are invariants of the language recognized:

Theorem 8.1 (towers and walls [49]). *All forward deterministic ω-automata recognizing the same language have the same types of towers and walls in the sense that if one of them has a tower of a certain height and sign or a wall of a certain length and sign, then the other has so, too.*

To illustrate this theorem we prove the claim for towers and start with a useful remark.

Remark 8.2. Consider a forward deterministic automaton with n states over an alphabet A. For $u \in A^*$, $v \in A^+$ and $k \geq n$, some power of v^k is the label of a loop at $\delta^*(q_I, uv^k)$, and this loop is positive if, and only if, uv^ω is accepted. (As usual, δ^* stands for the extended transition function defined by $\delta^*(q, \epsilon) = q$ and $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$ for all $q \in Q$, $u \in A^*$, and $a \in A$.)

For the proof of the claim on towers, assume equivalent forward deterministic ω-automata $\mathcal{A}$ and $\mathcal{A}'$ are given and consider any tower $\langle C_0, \ldots, C_{m-1} \rangle$ in $\mathcal{A}$, say a positive one; the argument is symmetric for a negative one. Let q be a state in C_{m-1} and, for every $i < m$, let the word v_i be a label for a loop at q with loop set C_i. Further, let u be a word such that $\delta(q_I, u) = q$. Then $\delta(q_I, uw) = q$ for every word $w \in (v_0 + \cdots + v_{m-1})^*$. Moreover, whether a word $uv_{i_0}v_{i_1}\ldots$ is accepted is determined by the least index occurring infinitely often among $i_0, i_1, \ldots$. Let k be greater than the number of states of $\mathcal{A}'$ and consider the words w_i defined inductively by $w_0 = \epsilon$ and $w_{i+1} = (v_{m-i-1}w_i)^k$, for $i < m$. Then, using Remark 8.2, we find that for each $i < m$, some power of w_{i+1} is the label of a loop at $\delta^*(q'_I, uw_m)$ and has the same sign as C_{m-i-1}. So the reverse sequence of the loop sets forms a positive tower in $\mathcal{A}'$ of height m. □

There is a strong relationship between towers and walls on one side and topological aspects of ω-languages on the other side, see [32].

8.2 The parity index

From Theorem 8.1 it follows that, in particular, the greatest height of a tower in a forward deterministic automaton is characteristic for the language recognized. This number is in-

timately connected with the number of priorities needed by a forward deterministic parity automaton to recognize the same language. To make this more precise we say a parity automaton uses n priorities if n is the maximum of the number of priorities occurring in any strongly connected component of the automaton. Given a regular ω-language the smallest number of priorities used in any forward deterministic parity automata recognizing the language is its parity index.

Corollary 8.3 (parity index). *The greatest height of a tower in a given forward deterministic ω-automaton is exactly the parity index of the language recognized.*

That the parity index is at least the greatest height of a tower follows from Theorem 8.1. For the converse, reconsider the construction from Section 4.5 that turns a Muller automaton into an equivalent parity automaton. Essentially, the Muller automaton without recurrence condition is cascaded with the LAA (latest appearance automaton) and augmented by a parity condition. It is enough to adjust the latter as follows. A state $\langle q, v\$v' \rangle$ is assigned the value $n - l + o$ where (i) the number l is the greatest height of a tower ending in a loop with loop set $\mathrm{occ}(v')$ and (ii) the number $o < 2$ is chosen in a way such that $n - l + o$ is even if the loop is positive and else odd. □

The Rabin index of a regular ω-language [49] is a similar but somewhat coarser measure.

8.3 Forward deterministic weak automata

In terms of the above complexity measure—parity index—the simplest forward deterministic automata that can be considered are the ones with parity index 1; these automata are exactly the forward deterministic weak automata.

On one hand, weak automata are indeed weak in the sense that the class of languages recognized by them is small, for instance, $(01)^\omega$ cannot even be recognized by such automata. In fact, there is a simple characterization of languages recognized by forward deterministic weak automata.

Remark 8.4. [43] An ω-language can be recognized by a forward deterministic weak automaton over some alphabet A if, and only if, it is a boolean combination of languages of the form UA^ω where U is a regular language of finite words.

On the other hand, weak automata have some properties which general ω-automata are lacking. One interesting property is described in Theorem 7.2. Another property has to do with their determinization:

Theorem 8.5 (conditional determinization [2]). *If an ω-language is recognized by some forward deterministic weak automaton, then a variant of the breakpoint construction can be used to transform a forward non-deterministic weak automaton recognizing the language into an equivalent forward deterministic weak automaton.*

8.4 Loops in backward deterministic automata

The requirement that in a backward deterministic automaton there is exactly one recurring run for every ω-word over the given alphabet is a very strong one, which has interesting implications.

Proposition 8.6. *[7] An ω-automaton is backward deterministic if, and only if, its transition relation is backward deterministic and every non-empty finite word is the label of a positive loop at exactly one state.*

For the proof, assume a backward deterministic automaton is given and let u be a non-empty finite word. If it is the label of a positive loop at two distinct states q and q', then there are at least two recurring runs of the automaton on u^ω—a contradiction. So u can only be the label of a positive loop at at most one state. Since there is a recurring run of the automaton on u^ω, there is some k such that u^k is the label of a loop at state q. If $\delta(u,q) \neq q$, then u^k would also be the label of a loop at $\delta(u,q)$ and there would be two recurring runs for u^ω—a contradiction. So u is the label of a positive loop at at least one state.

For the converse, assume every non-empty finite word is the label of a positive loop at exactly one state. Then every periodic word over the given alphabet has a recurring run and hence every ultimately periodic word over the same alphabet has so, too. In other words, the set of all words without recurring run is a regular ω-language without ultimately periodic words. From Remark 3.2(1), we can conclude this set is empty. This shows that for every ω-word there is at least one recurring run. By way of contradiction, assume there are two distinct recurring runs on a given ω-word u, say r and r'. Because of the backward deterministic transition relation there must be some i such that $r(j) \neq r'(j)$ for all $j \geq i$. As a consequence, there are positions i and j such that (i) $i < j$, $r(i) = r(j)$, $r'(i) = r'(j)$, and $r(i) \neq r'(i)$, and (ii) $r(i)u(i)r(i+1)\dots r(j)$ as well as $r'(i)u(i)r'(i+1)\dots r'(j)$ are positive loops at different states with the same label. This is a contradiction to the assumption. □

The above proposition, in combination with the final syntactic congruence (defined subsequent to Theorem 7.2), can be used to classify regular ω-languages using backward deterministic ω-automata, see [34].

9 Alternation

ω-Automata are often used in the context of two-player games of infinite duration played on graphs, and results on such games are useful tools for obtaining results on ω-automata.

9.1 Games of infinite duration with regular winning conditions

In this section, the fundamentals of games of infinite duration with regular winning conditions are recalled. Remark 9.2 is one of the prime applications of *forward deterministic*

ω-automata; in combination with Theorem 9.3, it explains why the parity condition is so important.

The players of a two-player game of infinite duration played on graphs are called Zero and One; a game is given by a set V of vertices, a set $E \subseteq V \times V$ of edges, a set V_0 of vertices owned by Zero, and a winning condition $W \subseteq V^\omega$. A play of such a game starting in some vertex v_I is a maximal path through the graph starting with the vertex v_I; the idea is that a pebble is moved over the edges of the graph from one vertex to the next, starting with the pebble on vertex v_I, and Zero moving in her vertices and One moving in the vertices owned by him, which are the ones in $V \setminus V_0$. A play is winning for Zero if the path is either finite and its last vertex belongs to One (that is, One cannot move anymore) or infinite and belongs to W; else it is winning for One.

When a player has a strategy for winning the plays starting in a particular vertex, the player is said to win the game starting in this vertex. The set of such vertices is called his or her winning region.

Often, a winning condition is a regular ω-language. More precisely, a coloring function $c\colon V \to C$ into a finite set of so-called colors and a regular ω-language L over C are given; the winning condition W is determined by $W = \{u \in V^\omega \mid c \circ u \in L\}$. One speaks of a regular winning condition.

A consequence of Martin's theorem [24] and McNaughton's result [25] is:

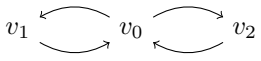
Theorem 9.1 (regular determinacy). *Given a game with a regular winning condition and a vertex in this game, either Zero or One wins the game starting in this vertex. The game is said to be determined in the vertex.*

A game is called a parity game if there is a function $\pi\colon V \to [n]$ such that $u \in W$ if, and only if, $\liminf_i \pi(u(i))$ is even. Hence, parity games can be viewed as games with a regular winning condition. From the fact that every regular ω-language is recognized by some forward deterministic parity automaton, one can derive:

Remark 9.2. Every game with a regular winning condition can be embedded into a game with a parity winning condition.

To understand what exactly this means assume a game with a regular winning condition as described above and a forward deterministic parity automaton $\mathcal{A}$ recognizing the language L are given. Consider the modified game with vertex set $V \times Q$, Zero's vertex set $V_0 \times Q$, edge set $\{\langle\langle v, q\rangle, \langle v', \delta(q, c(v'))\rangle\rangle \mid \langle v, v'\rangle \in V \text{ and } q \in Q\}$, and priority function $\langle v, q\rangle \mapsto \pi(q)$. Because the automaton $\mathcal{A}$ is forward deterministic, playing in the original game starting from a vertex v is exactly the same as playing in the new game starting from the vertex $\langle v, q_I\rangle$. In particular, Zero wins the former game in a vertex v if, and only if, she wins the latter game in $\langle v, q_I\rangle$. In other words, without loss of generality, only games with parity winning conditions need to be considered when regular winning conditions are used.

In general, regular winning conditions may require a player to remember a certain amount of information in order to win. For instance, if the winning condition demands that Zero visits the vertices v_1 and v_2 infinitely often in the graph



,

where v_0 is her vertex, then she cannot base her decision what to do in vertex v_0 only on the fact that she is in that vertex. (Of course, when she remembers where she moved previously, she can alternate and win.) Opposed to this, if the winning condition demands that Zero visits v_2 infinitely often, she only needs to follow the rule "if in vertex v_0, go to vertex v_2"—her decision what to do next is only based on the current vertex.

A uniform positional winning strategy for Zero is a function $(W_0 \cap V_0) \to W_0$, where W_0 is Zero's winning region, such that no matter where in W_0 a play starts, if Zero moves as determined by the function, then the resulting play is winning for Zero. For One, the definition is symmetric.

Theorem 9.3 (positional strategies [13]). *In every parity game, both players have a uniform positional winning strategy.*

9.2 State- and transition-controlled alternating automata

In general, an alternating automaton is an automaton where acceptance depends on the full computation tree on a given word, more precisely, such an automaton provides means for specifying that a given word is accepted if, and only if, a certain subgraph of the full computation tree exists. At one extreme, when this subgraph is required to be a rooted path, then the automaton is nothing else than a conventional automaton.

For ω-automata, essentially two variants of alternating automata have been studied: in one variant, alternation is specified by partitioning the state space [27]; in the other variant, alternation is specified by complex transition formulas [31].

In the state-controlled variant the state space Q is partitioned into a set E of existential states and a set U of universal states (where either set could be empty) and the set of initial states is either a subset of E or of U. A run of the automaton on a word u is a prefix-closed set $T \subseteq Q^*$, which should be thought of as a tree satisfying the following properties for every vertex $vq \in T$:

- if $q \in E$, then there exists a state q' such that $vqq' \in T$ and $\langle q, u(|v|), q' \rangle \in \Delta$;
- if $q \in U$, then $vqq' \in T$ for every q' such that $\langle q, u(|v|), q' \rangle \in \Delta$.

The run is initial if either $Q_I \subseteq E$ and $Q_I \cap T \neq \varnothing$ or $Q_I \subseteq U$ and $Q_I \subseteq T$; it is recurring if every word $r \in Q^\omega$ whose finite prefixes all belong to T (every infinite rooted path through T) is recurring in the sense of the given transition condition. This means, in particular, if $E = Q$ and the set of initial states is existential, then the automaton can be viewed as an ordinary ω-automaton. It is said to be a universal automaton if $U = Q$.

From the closure under complementation of the class of ω-languages recognized by Büchi automata, one obtains immediately:

Remark 9.4. Every regular ω-language is recognized by a universal co-Büchi automaton.

In the transition-controlled variant, the transition relation is replaced by a transition function $\delta\colon Q \times A \to M(Q)$, where $M(Q)$ is the set of all expressions built from states, the connectives $\vee$ ("or") and $\wedge$ ("and"), and the boolean constants 0 ("false") and 1 ("true"). For instance, $q \wedge (q' \vee q'')$ could be a value of the transition function. The set of initial states is replaced by an expression from $M(Q)$. Again, a run is a prefix-closed set $T \subseteq Q^*$, but

this time satisfying the following condition. For each vertex $vq \in T$, the set $\{q' \mid vqq' \in T\}$ satisfies the expression $\delta(q, u(|v|))$, where satisfaction is defined in the obvious way. A run is initial if $\{q \in Q \mid q \in T\}$ satisfies the initial condition. Being recurring is defined as above.

Remark 9.5. A state-controlled alternating ω-automaton can be viewed as a transition-controlled alternating ω-automaton.

More precisely, for every existential state q one sets $\delta(q, a) = \bigvee\{q' \mid \langle q, a, q'\rangle \in \Delta\}$, and for every universal state q one sets $\delta(q, a) = \bigwedge\{q' \mid \langle q, a, q'\rangle \in \Delta\}$; the set of initial states is converted into an initial condition in the same way; the recurrence condition does not need to be changed.

9.3 Alternating automata and games

Given a state-controlled alternating automaton as above and an ω-word u over the same alphabet, the question whether u is accepted by the automaton can be viewed as the question whether Zero wins a certain game, the so-called automaton game for u. The vertices of this game are pairs of the form $\langle q, u'\rangle$, where u' is a suffix of u; such a vertex belongs to Zero if, and only if, q is existential; there is an edge from $\langle q, u'\rangle$ to $\langle q', u''\rangle$ if $\langle q, u'(0), q'\rangle \in \Delta$ and $u'' = u'(1)u'(2)\ldots$; the recurrence condition is adapted in the straightforward fashion, based on the state in the first component.

Remark 9.6. A state-controlled alternating ω-automaton with existential [universal] initial states accepts a word u if, and only if, Zero wins the automaton game for u in some [every] vertex in $Q_I \times \{u\}$.

From Theorem 9.1, which states that the games that occur in this fashion are determined, one can derive that a word is not accepted if, and only if, One has a winning strategy. This is equivalent to saying that the dual automaton accepts the word, where dualizing an automaton has the obvious meaning: existential and universal states exchange their roles and the recurrence condition is replaced by its negation. In other words, complementation is a trivial problem for alternating automata.

Proposition 9.7 (complementing alternating automata). *The dual of a state-controlled alternating ω-automaton recognizes the complement of the language recognized by the given automaton.*

Remark 9.6 and Proposition 9.7 hold true for transition-controlled alternating ω-automata as well, but the definition of the automaton game and the dualization process need to be adapted. The vertices of the automaton games are of the form $\langle \varphi, u'\rangle$ where φ is a subformula of some value of the transition function. In the dualization process, the values of the transition function and the initial condition are dualized.

9.4 From alternating automata to non-deterministic ones

Alternating ω-automata can be exponentially more concise than ordinary ones, just as in the finite-word setting [12], but with regard to expressive power there is no difference. This is a major application of complementing ω-automata.

Theorem 9.8 (from alternating to non-deterministic [27]). *For every alternating ω-automaton there exists an equivalent non-deterministic Büchi automaton.*

To prove this, first observe that it is enough to consider alternating parity automata, because any Muller condition can be turned into a parity condition as described in the proof of Theorem 4.5.

By Theorem 9.3, parity games have uniform positional winning strategies. It follows that if there is an accepting run (recall that runs are trees) of an alternating parity automaton on a given word, then there is also an accepting subgraph of the run DAG, where this is defined in the obvious way. Checking that in a subgraph of a run DAG *all* rooted paths are recurring can be done using an appropriate ω-automaton, as explained in what follows.

Consider the non-deterministic parity automaton over the alphabet $\wp(Q \times Q)$, with state set Q, initial set Q_I, transition relation $\{\langle q, a, q' \rangle \mid \langle q, q' \rangle \in a\}$, and parity condition $\pi + 1$. This automaton accepts a word u if the DAG which is obtained by collating the letters of u contains *some* initial rooted path starting in an initial state and not satisfying the parity condition of the original automaton. Any ω-automaton recognizing the complement of the language recognized by this automaton is one that can check the DAG's.

To sum up, cascading (i) an automaton producing a subgraph of a run DAG of a given ω-word satisfying the transition relation and (ii) the above automaton yields the desired automaton. □

9.5 Weak alternating automata

Remark 8.4 states that weak deterministic ω-automata only recognize fairly simple ω-languages. This is different for alternating automata:

Theorem 9.9 (from alternating to weak alternating). *[20] For every alternating Büchi automaton with n states there exists an equivalent weak alternating automaton with $2n^2$ states.*

By dualization, it is enough to consider alternating co-Büchi automata. Theorem 9.3 says that runs of alternating co-Büchi (and Büchi) automata can be thought of as run DAG's. The use of rank functions from Section 5 leads to the following characterization of when an ω-word u is accepted by a transition-controlled co-Büchi alternating automaton with n states. There exists a tree $T \subseteq (Q \times [2n])^*$ satisfying the following conditions. (i) The set $\{q \in Q \mid \langle q, c \rangle \in T \text{ for some } c < 2n\}$ satisfies the initial condition. (ii) Whenever $v\langle q, c \rangle \in T$, then $\{q' \in Q \mid v\langle q, c \rangle\langle q', c' \rangle \in T \text{ for some } c' < 2n\}$ satisfies $\delta(q, u(|v|))$. (iii) There is no vertex $v\langle q, 2j + 1 \rangle \in T$ with $q \in B$. (iv) When $v\langle q, c \rangle\langle q', c' \rangle \in T$, then $c \geq c'$. (v) For every rooted path $\langle q_0, c_0 \rangle\langle q_1, c_1 \rangle \ldots$ there exists some i such that

$c_i, c_{i+1}, \ldots$ are all odd. This can be used to construct a transition-controlled weak alternating automaton with state set $Q \times [2n]$; the initial condition and the transition function are adapted from the given automaton in a straightforward fashion; the Büchi set consists of all states with an odd second component. □

It should be noted that the breakpoint construction can be used to convert a weak alternating Büchi automaton into an equivalent non-deterministic one.

9.6 Simulation relations and simulation games

One way to compare automata with each other, more precisely, to compare their internal structure, is to use simulation relations, or, more generally, simulation games.

A simple approach is to say that a Büchi automaton $\mathcal{A}'$ forwardly simulates a Büchi automaton $\mathcal{A}$ if there is a relation $\sigma \subseteq Q \times Q'$ such that the following three conditions are satisfied. (i) For every $q \in Q_I$ there is some $q' \in Q'_I$ such that $\langle q, q' \rangle \in \sigma$. (ii) For all $\langle q, q' \rangle \in \sigma$ and $\langle q, a, r \rangle \in \Delta$ there is some $r' \in Q'$ such that $\langle r, r' \rangle \in \sigma$ and $\langle q', a, r' \rangle \in \Delta'$. (iii) For all $\langle q, q' \rangle \in \sigma$, if $q \in B$, then $q' \in B'$.

The important observations concerning this definition are:

Theorem 9.10 (direct simulation [11]). *(1) If a Büchi automaton $\mathcal{A}'$ simulates a Büchi automaton $\mathcal{A}$, then the language recognized by $\mathcal{A}$ is a subset of the language recognized by $\mathcal{A}'$.*

(2) Whether a Büchi automaton $\mathcal{A}'$ simulates a Büchi automaton $\mathcal{A}$ can be determined in time linear in the product of the sizes of $\mathcal{A}$ and $\mathcal{A}'$.

As a consequence, simulation relations can be used for efficient (but incomplete) inclusion tests.

The requirement that a Büchi state in the simulating automaton match a Büchi state in the simulated automaton right away is very strong. For inclusion to hold, it would be enough if a Büchi state in the simulated automaton is matched by a Büchi state in the simulating automaton at a later position. This is captured by the notion of delayed simulation, which is best phrased in terms of a certain two-player game, where one of the players is called Duplicator and tries to show that simulation is given, whereas the other is called Spoiler and tries to show that this is not the case.

More precisely, the game determines whether a state in a Büchi automaton $\mathcal{A}'$ delayed simulates a state in a Büchi automaton $\mathcal{A}$. When a play of the game starts, there is a pebble on each of the two states in question. In every round of the game, first Spoiler is required to move the pebble on $\mathcal{A}$ over some transition and then Duplicator is required to move the other pebble (the pebble in $\mathcal{A}'$) over some transition with the same label. If one of the players cannot move anymore, this player looses early. If an infinite play emerges, then Duplicator wins if, and only if, the following holds: whenever Spoiler visits a Büchi state in some round, Duplicator visits a Büchi state in the same or in a later round. The state in $\mathcal{A}'$ delayed simulates the state in $\mathcal{A}$ if Duplicator has a winning strategy in the game just described. The automaton $\mathcal{A}'$ delayed simulates the automaton $\mathcal{A}$ if every initial state of $\mathcal{A}$ is simulated by some initial state of $\mathcal{A}'$. Observe that the above game can be viewed as a game of infinite duration with a regular winning condition as described in Section 9.1.

Theorem 9.10 carries over to delayed simulation, only the complexity of computing delayed simulation is higher [14].

For purposes of state-space reduction, it useful to study simulation in both directions: if one state [delayed] simulates another one and vice versa, the states are said to mutually [delayed] simulate each other. These relations are, indeed, equivalence relations and have a useful property:

Theorem 9.11 (quotienting [14]). *If, in a quotient of a Büchi automaton with regard to the mutual [delayed] simulation relation, initial and Büchi states are chosen appropriately, then the resulting automaton is equivalent to the given one.*

This gives, in effect, two polynomial-time algorithms for reducing the state space of Büchi automata, one less efficient than the other, but producing smaller automata. Finding and even approximating minimum-size Büchi automata is PSPACE-hard, in fact, this is independent of the type of the automaton, because results from finite-state automata on finite words [16] carry over in a straightforward fashion.

In principle, one could also work with bisimulation rather than mutual simulation, but this gives, in general, worse reductions.

Much effort has gone into finding coarser relations for state-space reductions, and there are various ways of approaching this: letting Duplicator match with more than just one pebble, relaxing the winning condition for Duplicator further, considering backward simulation, and so on.

10 Applications in logic

ω-Automata were introduced in the late fifties in the context of mathematical logic, more precisely, Büchi automata first showed up in [5] (in disguise) and were used there as a tool for proving that theories of specific structures are decidable. From a modern point of view, Büchi showed that the structures are ω-automatic [18].

10.1 ω-Automatic structures

Assume a first-order structure $\mathfrak{S}$ consisting of a universe U and a family $\{R_i\}_{i\in I}$ of relations, say R_i having arity n_i, are given; the question is whether the theory of this structure is decidable. A good example are the real numbers with the ternary relation "addition", the predicate "is positive", and the predicate "is power of 2".

An ω-automatic presentation of a structure $\mathfrak{S}$ as above is given by an alphabet A, an ω-automaton $\mathcal{U}$ over A, an ω-automaton $\mathcal{E}$ over $A\times A$, and, for each $i \in I$, an ω-automaton $\mathcal{R}_i$ over $\bigtimes_{i<n_i} A$. It is required that there exists an onto function $f\colon \mathrm{L}(\mathcal{U}) \to U$ such that the following conditions are satisfied:

- For all $u, v \in L(\mathcal{U})$, $f(u) = f(v)$ if, and only if, $u * v \in L(\mathcal{E})$.
- For all $i \in I$ and $u_0, \ldots, u_{n_i-1} \in L(\mathcal{U})$, $u_0 * \cdots * u_{n_i-1} \in L(\mathcal{R}_i)$ if, and only if, $\langle f(u_0), \ldots, f(u_{n_i-1})\rangle \in R_i$.

To extend the above example, one can start with an ω-automaton $\mathcal{U}$ that accepts exactly the ω-words representing real numbers as described in Section 1.1. Then $\mathcal{E}$ must be constructed in a way such that it identifies representations of identical numbers. Finally, ω-automata representing the three respective relations must be found. A simple automaton is the "is power of 2" automaton, which only checks that there is exactly one occurrence of 1 and that this occurrence is not in position 0 (because otherwise the number represented would be 0, more precisely, -0, which is not a power of 2).

The fundamental result about ω-automatic structures is:

Theorem 10.1 (ω-automatic structures [18]). *The first-order theory of every ω-automatic structure is decidable.*

The reason for this is that, by induction, one can show that for every first-order formula in the respective vocabulary one can construct an ω-automaton that recognizes the representations of the satisfying assignments. When $\varphi = \varphi(x_0, \ldots, x_{n-1})$ is a formula with all of its free variables among $x_0, \ldots, x_{n-1}$, then a word of the form $u_0 * \cdots * u_{n-1}$ represents a satisfying assignment if $u_i \in L(\mathcal{U})$ for every $i < n$ and $\mathfrak{S}, f(u_0), \ldots, f(u_{n-1}) \models \varphi(x_0, \ldots, x_{n-1})$.

For the base case, there is almost nothing to show, because this is part of the definition of ω-automatic structure. For the induction itself it should be noted that disjunction can essentially be viewed as union, negation as complementation, and existential quantification as projection. All these operations can easily be implemented effectively on ω-automata. In other words, there is an effective procedure that, given a closed formula, constructs an ω-automaton over the unary alphabet, $\times_{i<0} A$, which accepts some word, more precisely, the word $\langle\rangle^\omega$, if, and only if, the formula is true in the given structure. Non-emptiness can be verified effectively for ω-automata, see Remark 3.2(2). □

An important example for this theorem, already mentioned in Büchi's seminal paper, is the one described above: the real numbers with addition and the "is positive", "is power of two", and "is an integer" predicates.

Another example from Büchi's original work is the monadic second-order theory of the natural numbers with successor, more precisely: the structure is the set of natural numbers endowed with the successor predicate; in the vocabulary of the logical language there are, in addition to what is part of a suitable first-order language (symbols for disjunction, negation, existential quantification, the binary successor relation, variables for natural numbers), variables for sets of natural numbers, a symbol for "is element of", and a symbol for existential quantification of set variables. At first glance, this does not look like a situation where Theorem 10.1 can be applied, but it actually can: a formula in the above logic can be translated in a straightforward fashion into a first-order formula for the structure with the power set of the natural numbers as universe and endowed with the "is singleton" predicate, the binary relation "is subset of", and the binary relation "every element of … has a successor in …".

Theorem 10.2 (decidability of S1S [5]). *The monadic second-order theory of the natural numbers with the successor predicate is decidable.*

From the point of view of ω-automata theory, there are several applications in logic which are of particular interest. Two of them are discussed in what follows.

10.2 Temporal logic

Temporal logic comes in many different flavors. The version that is most often considered and also most amenable to being dealt with using ω-automata is the one where the temporal operators used are future operators (next, X; eventually in the future, F; always in the future, G; until, U; release, R) and the time domain is discrete, more precisely, where the time domain is ω, the set of natural numbers. In such a context, a temporal variable, here denoted p_i, is assigned a set of natural numbers, the points in time where the variable is true. So if the variables occurring in a given formula φ are among $p_0, \ldots, p_{n-1}$, then the models of this formula can be viewed as ω-words over the alphabet $\wp(\{p_0, \ldots, p_{n-1}\})$.

For instance, the set of models of the formula $\mathrm{G}(p_0 \to \mathrm{F}p_1)$, which is read "now and always in the future, if p_0, then p_1 at the same time or some point later", can be viewed as the set of ω-words over $\wp(\{p_0, p_1\})$ with the property that whenever p_0 is an element of a letter at some position, then p_1 belongs to the same letter or some other letter in a position to the right.

The starting point for constructing an ω-automaton recognizing the set of models of a given formula is the observation that whether a formula of temporal logic is true in some point in time only depends on (i) which of its strict subformulas hold true in this and the next point in time and (ii) whether the formula itself holds true in the next point in time. So a suitable automaton can guess, for each point in time, which subformulas are true and then verify its guessing locally in a backward deterministic fashion. For some temporal operators, it is important though to also verify certain conditions globally. For instance, it is true that the formula $\mathrm{F}\varphi$ holds true in position i if, and only if, φ holds true in position i or $\mathrm{F}\varphi$ holds true in position $i+1$, and $\mathrm{F}\varphi$ holds true in position $i+1$ if, and only if, φ holds true in position $i+1$ or $\mathrm{F}\varphi$ holds true in position $i+2$, and so on, but, clearly, the formula φ must become true at some point. Such a global condition can be captured by an appropriate recurrence condition. The initial states are the ones where the automaton guesses the entire formula to be true.

The general theorem is as follows:

Theorem 10.3 (from temporal logic to automata [50, 48]). *Every temporal formula with n subformulas can be translated into an equivalent backward deterministic generalized Büchi automaton with at most 2^n states and as many Büchi sets as there are subformulas with leading temporal operator* F *or* U.

This implies, in particular, that satisfiability and validity of temporal formulas as well as model checking temporal formulas over finite-state system with fairness conditions are problems in PSPACE [41].

Future linear-time temporal formulas can be translated directly into weak alternating automata (see Section 9.5) of a very specific structure; the resulting number of states is the number of subformulas.

10.3 The additive theory of the reals

Regarding the aforementioned example of the real numbers one can show (by other means than automata-theoretic ones) that the relations definable by formulas in the underlying

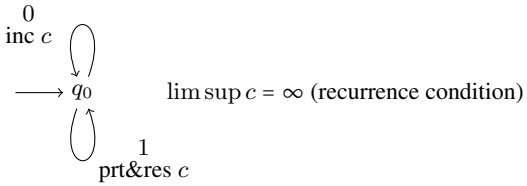
first-order language are all recognizable by forward deterministic weak automata as introduced in Section 8.1 (when real numbers are represented as described in Section 1.1). In view of Theorem 8.5, this means that a recursive procedure for constructing automata representing such relations can take advantage of conditional determinization, which is much less complicated than ordinary determinization, and of simple and fast minimization procedures as pointed out in Theorem 7.2. This, in the end, leads to feasible decision procedures [2].

11 More complex recurrence conditions

Much effort can and has been put into extending ω-automata like the ones dealt with in this chapter, that is, the ones with a finite state space and a recurrence condition based on the states occurring infinitely often in a run. There are finite-state automata working on other infinite objects: other ordinals, the integers, linear orderings in general, and, most notably, infinite trees; there are ω-automata using more complex storage, for instance, ω-automata with stacks; there are probabilistic ω-automata, that is, ω-automata where transitions are taken with certain probabilities; there are timed ω-automata, using clocks and processing infinite sequences of events having a duration; and so on. ???

Beside all this, Bojańczyk and Colcombet suggest in [3] to strengthen the models discussed in this chapter by a more powerful mechanism for defining recurrence, allowing a finer analysis of what happens "in the infinite". In their model, every automaton has a finite number of counters. A transition is of the form $\langle q, a, \alpha, q' \rangle$ where α is a function assigning to each counter no action or one of the following two: "inc"—increment the counter by one; "prt&res"—output (print) the counter value and then reset the counter. So, for every counter, a finite or infinite sequence of natural numbers, its recurrence sequence, is produced in each run. The recurrence condition is a boolean combination of conditions of the form $\liminf c = \infty$ and $\limsup c = \infty$, with c standing for a counter. A run of such an automaton is recurring if every recurrence sequence is infinite and they all satisfy the recurrence condition.

A good example for a non-regular ω-language which can be recognized by such an automaton is the language L_{ub0} (see Section 7.2) of all ω-words of the form $0^{i_0}10^{i_1}1\ldots$ where $\limsup_{j\to\infty} i_j = \infty$. This language is recognized by an automaton with one counter, say c:



Acknowledgment I am grateful to Christof, Sebastian, and my master students for insightful comments, to Wolfgang for his constant support, and to the other contributors to this volume and, in particular, Jean-Éric for their endless patience.

References

[1] André Arnold, *A syntactic congruence for rational ω-languages*, Theor. Comput. Sci. **39** (1985), 333–335.

[2] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper, *An effective decision procedure for linear arithmetic over the integers and reals*, ACM Trans. Comput. Log. **6** (2005), no. 3, 614–633.

[3] Mikołaj Bojańczyk and Thomas Colcombet, *Bounds in ω-regularity*, Proc. of the 21th IEEE Symposium on Logic in Computer Science (Seattle, WA), IEEE Computer Society, 2006, pp. 285–296.

[4] Stefan Breuers, Christof Löding, and Jörg Olschewski, *Improved Ramsey-based Büchi complementation*, 2012, pp. 150–164.

[5] J. Richard Büchi, *On a decision method in restricted second order arithmetic*, Logic, Methodology and Philosophy of Science: Proc. of the 1960 Internat. Congr. (Ernest Nagel, Patrick Suppes, and Alfred Tarski, eds.), Stanford Univ. Press, Stanford, California, 1962, pp. 1–11.

[6] J. Richard Büchi and Lawrence H. Landweber, *Solving sequential conditions by finite-state strategies*, Trans. Amer. Math. Soc. **138** (1969), 295–311.

[7] Olivier Carton and Max Michel, *Unambiguous Büchi automata*, Theor. Comput. Sci. **297** (2003), no. 1-3, 37–81.

[8] Thomas Colcombet, *Forms of determinism for automata (invited talk)*, 29th International Symposium on Theoretical Aspects of Computer Science (Paris) (Christoph Dürr and Thomas Wilke, eds.), LIPIcs, vol. 14, Schloss Dagstuhl, Leibniz-Zentrum für Informatik, 2012, pp. 1–23.

[9] Thomas Colcombet and Konrad Zdanowski, *A tight lower bound for determinization of transition labeled Büchi automata*, Automata, Languages and Programming: Part II (Rhodes, Greece) (Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, eds.), Lecture Notes in Computer Science, vol. 5556, Springer, 2009, pp. 151–162.

[10] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis, *Memory-efficient algorithms for the verification of temporal properties*, Formal Methods in System Design **1** (1992), no. 2/3, 275–288.

[11] David L. Dill, Alan J. Hu, and Howard Wong-Toi, *Checking for language inclusion using simulation preorders*, Computer Aided Verification (Aalborg, Denmark) (Kim Guldstrand Larsen and Arne Skou, eds.), Lecture Notes in Computer Science, vol. 575, Springer, 1991, pp. 255–265.

[12] Doron Drusinsky and David Harel, *On the power of bounded concurrency I: Finite automata*, J. ACM **41** (1994), no. 3, 517–539.

[13] E. Allen Emerson and Charanjit S. Jutla, *Tree automata, Mu-Calculus and determinacy (extended abstract)*, 32nd Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico), IEEE Computer Society, 1991, pp. 368–377.

[14] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller, *Fair simulation relations, parity games, and state space reduction for Büchi automata*, SIAM J. Comput. **34** (2005), no. 5, 1159–1175.

[15] Ehud Friedgut, Orna Kupferman, and Moshe Y. Vardi, *Büchi complementation made tighter*, Int. J. Found. Comput. Sci. **17** (2006), no. 4, 851–868.

[16] Gregor Gramlich and Georg Schnitger, *Minimizing NFA's and regular expressions*, J. Comput. Syst. Sci. **73** (2007), no. 6, 908–923.

[17] Yuri Gurevich and Leo Harrington, *Trees, automata, and games*, Proceedings of the 14th Annual ACM Symposium on Theory of Computing (San Francisco, California) (Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, eds.), ACM, 1982, pp. 60–65.

[18] Bernard R. Hodgson, *Décidabilité par automate finite*, Annales des sciences mathématiques du Québec **7** (1983), no. 1, 39–57.

[19] Nils Klarlund, *Progress measures for complementation of ω-automata with applications to temporal logic*, 32nd Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico), IEEE Computer Society, 1991, pp. 358–367.

[20] Orna Kupferman and Moshe Y. Vardi, *Weak alternating automata are not that weak*, ACM Trans. Comput. Log. **2** (2001), no. 3, 408–429.

[21] Dénes Kőnig, *Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*, Akademischer Verlag, Leipzig, 1936.

[22] Christof Löding, *Optimal bounds for transformations of ω-automata*, Foundations of Software Technology and Theoretical Computer Science (Chennai, India) (C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, eds.), Lecture Notes in Computer Science, vol. 1738, Springer, 1999, pp. 97–109.

[23] _____, *Efficient minimization of deterministic weak ω-automata*, Inf. Process. Lett. **79** (2001), no. 3, 105–109.

[24] Donald A. Martin, *Borel determinacy*, Annals of Mathematics **102** (1975), no. 2, pp. 363–371 (English).

[25] Robert McNaughton, *Testing and generating infinite sequences by a finite automaton*, Information and Control **9** (1966), no. 5, 521–530.

[26] Max Michel, *Complementation is more difficult with automata on infinite words*, CNET, Paris, 1988.

[27] Satoru Miyano and Takeshi Hayashi, *Alternating finite automata on ω-words*, Theor. Comput. Sci. **32** (1984), 321–330.

[28] Andrzej Włodzimierz Mostowski, *Regular expressions for infinite trees and a standard form of automata*, Symposium on Computation Theory (Zaborów, Poland) (Andrzej Skowron, ed.), Lecture Notes in Computer Science, vol. 208, Springer, 1984, pp. 157–168.

[29] David E. Muller, *Infinite sequences and finite machines*, Switching Theory and Logical Design (Chicago, Illinois), IEEE, 1963, pp. 3–16.

[30] David E. Muller, Ahmed Saoudi, and Paul E. Schupp, *Alternating automata, the weak monadic theory of trees and its complexity*, Theor. Comput. Sci. **97** (1992), no. 2, 233–244.

[31] David E. Muller and Paul E. Schupp, *Alternating automata on infinite trees: New results and proofs of the theorems of Rabin, McNaughton and Safra*, Theor. Comput. Sci. **54** (1987), 267–276.

[32] Dominique Perrin and Jean-Éric Pin, *Infinite words: Automata, semigroups, logic, and games*, Pure and Applied Mathematics, vol. 141, Elsevier, Amsterdam, 2004.

[33] Nir Piterman, *From nondeterministic Büchi and Streett automata to deterministic parity automata*, Logical Methods in Computer Science **3** (2007), no. 3, 1–21.

[34] Sebastian Preugschat and Thomas Wilke, *Effective characterizations of simple fragments of temporal logic using Carton–Michel automata*, Logical Methods in Computer Science **9** (2013), no. 2, 1–22.

[35] Michael O. Rabin, *Decidability of second-order theories and automata on infinite trees*, Trans. Amer. Math. Soc. **141** (1969), 1–35.

[36] Frank Plumpton Ramsey, *On a problem of formal logic*, Proc. of the London Mathematical Society **30** (1929), 338–384.

[37] Shmuel Safra, *On the complexity of ω-automata*, 29th Annual Symposium on Foundations of Computer Science (White Plains, New York), IEEE Computer Society, 1988, pp. 319–327.

[38] ———, *Exponential determinization for ω-automata with a strong fairness acceptance condition*, SIAM J. Comput. **36** (2006), no. 3, 803–814.

[39] Sven Schewe, *Büchi complementation made tight*, 26th International Symposium on Theoretical Aspects of Computer Science (Freiburg, Germany) (Susanne Albers and Jean-Yves Marion, eds.), LIPIcs, vol. 3, Schloss Dagstuhl, Leibniz-Zentrum für Informatik, 2009, pp. 661–672.

[40] ———, *Tighter bounds for the determinisation of büchi automata*, Foundations of Software Science and Computational Structures (York, UK) (Luca de Alfaro, ed.), Lecture Notes in Computer Science, vol. 5504, Springer, 2009, pp. 167–181.

[41] A. Prasad Sistla and Edmund M. Clarke, *The complexity of propositional linear temporal logics*, J. ACM **32** (1985), no. 3, 733–749.

[42] Ludwig Staiger, *Finite-state ω-languages*, J. Comput. Syst. Sci. **27** (1983), no. 3, 434–448.

[43] Ludwig Staiger and Klaus Wagner, *Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen*, Elektronische Informationsverarbeitung und Kybernetik **10** (1974), no. 7, 379–392.

[44] Robert S. Streett, *Propositional dynamic logic of looping and converse is elementarily decidable*, Inform. and Control **54** (1982), no. 1-2, 121–141.

[45] Wolfgang Thomas, *Automata on infinite objects*, Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (Jan van Leeuwen, ed.), Elsevier, 1990, pp. 133–192.

[46] Wolfgang Thomas, *Languages, automata, and logic*, Handbook of Formal Languages (Gregorz Rozenberg and Arto Salomaa, eds.), vol. III, Springer, New York, 1997, pp. 389–455.

[47] Moshe Y. Vardi, *The Büchi complementation saga*, STACS 2007 (Aachen, Germany) (Wolfgang Thomas and Pascal Weil, eds.), Lecture Notes in Computer Science, vol. 4393, Springer, 2007, pp. 12–22.

[48] Moshe Y. Vardi and Pierre Wolper, *Reasoning about infinite computations*, Inf. Comput. **115** (1994), no. 1, 1–37.

[49] Klaus W. Wagner, *Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen*, Elektronische Informationsverarbeitung und Kybernetik **13** (1977), no. 9, 473–487.

[50] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla, *Reasoning about infinite computation paths (extended abstract)*, 24th Annual Symposium on Foundations of Computer Science (Tucson, Arizona), IEEE Computer Society, 1983, pp. 185–194.

[51] Qiqi Yan, *Lower bounds for complementation of ω-automata via the full automata technique*, Automata, Languages and Programming: Part II (Venice, Italy) (Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, eds.), Lecture Notes in Computer Science, vol. 4052, Springer, 2006, pp. 589–600.