Internal Report TCS-96-2
August, 1996

Finite-state Automata on Infinite Inputs[1]

Madhavan Mukund
SPIC Mathematical Institute
92 G N Chetty Rd, Madras 600 017, India
E-mail: madhavan@ssf.ernet.in

Abstract

This paper is a self-contained introduction to the theory of finite-state automata on infinite words. The study of automata on infinite inputs was initiated by Büchi in order to settle certain decision problems arising in logic. Subsequently, there has been a lot of fundamental work in this area, resulting in a rich and elegant mathematical theory. In recent years, there has been renewed interest in these automata because of the fundamental role they play in the automatic verification of finite-state systems.

[1]Tutorial talk, Sixth National Seminar on Theoretical Computer Science, Banasthali Vidyapith, Banasthali, Rajasthan, August 1996

Introduction

Büchi initiated the study of finite-state automata working on infinite inputs in [Bü60]. He was interested in showing that the monadic second order logic of infinite sequences (S1S) was decidable. Büchi discovered a deep and elegant connection between sets of models of formulas in this logic and ω-regular languages, the class of languages over infinite words accepted by finite-state automata.

A few years later, Muller independently proposed an alternative definition of finite-state automata on infinite inputs [Mu63]. His work was motivated by questions in switching theory.

The theory of ω-regular languages and automata on infinite words is substantially more complex than the corresponding theory for finite words. This was evident from Büchi's initial work, where he showed that non-deterministic automata over infinite inputs are strictly more powerful than deterministic automata. This means that basic constructions like complementation are correspondingly more intricate for this class of automata.

During the 1960's, fundamental contributions were made to this area. McNaughton proved that with Muller's definition, deterministic automata suffice for recognizing all ω-regular languages [Mc66]. Later, Rabin extended Büchi's decidability result for S1S to the monadic second order of the infinite binary tree (S2S) [Ra69]. The logical theory S2S is extremely expressive and Rabin's theorem can be used to settle a number of decision problems in logic.

Despite this strong connection between automata on infinite inputs and the decidability of logical theories, there was a lull in the area during the 1970's. One reason for this was Meyer's negative result about the complexity of the automata-theoretic decision procedure for S1S and S2S [Me75]—he showed that, in the worst case, the automata that one constructs would be hopelessly large and impossible to use in practice.

During the past decade or so, however, there has been renewed interest in applying automata on infinite words to solve problems in logic. To a large extent, this is a consequence of the development of temporal logic as a formalism for specifying and verifying properties of programs [Em90, Pn77]. It turns out that automata on infinite words (and trees) can be *directly* applied to settle important questions in temporal logic, without invoking S1S and S2S. In conjunction with these new applications, there has been greater emphasis on evaluating the complexity of different constructions on these automata [Sa88, SV89, SVW87, VW94].

In this tutorial, we present a self-contained introduction to the theory of finite-state automata on infinite words. We begin with some preliminaries on the notation we will use in the paper. In Section 1, we introduce Büchi automata and ω-regular languages and prove some basic results. The next section describes in detail the connection between ω-regular languages and formulas of S1S. In Section 3, we look at stronger definitions of automata, proposed by Muller, Rabin and Streett. The last technical section, Section 4, describes in detail a determinization construction for Büchi automata. We conclude with a quick summary of various aspects of the theory which could not discussed in the paper.

For a more detailed introduction to the area, the reader is encouraged to consult the excellent survey by Thomas [Th90]. This tutorial—especially Sections 1 and 2—draws heavily on the material presented in [Th90].

Notation

Throughout this paper, Σ denotes a finite set of symbols called an *alphabet*. A *word* is a sequence of symbols from Σ. The set Σ^* denotes the set of finite words over Σ while the set Σ^ω is the set of infinite words over Σ. A *language* is a set of words. Every language we consider either consists exclusively of finite words or exclusively of infinite words.

Typically, elements of Σ will be denoted $a, b, c, \ldots$, finite words will be denoted $u, v, w, \ldots$, and infinite words will be denoted $\alpha, \beta, \ldots$. We use $U, V, \ldots$ to denote languages of finite words—that is, subsets of Σ^*. L will be reserved for languages consisting exclusively of infinite words.

An infinite word $\alpha \in \Sigma^\omega$ is an infinite sequence of symbols from Σ. We shall represent α as a function $\alpha : \mathbb{N}_0 \to \Sigma$, where $\mathbb{N}_0$ is the set $\{0, 1, 2, \ldots\}$ of natural numbers. Thus, $\alpha(i)$ denotes the letter occurring at the i^{th} position. For natural numbers m and n, $m \leq n$, $\alpha[m..n]$ denotes the finite word $\alpha(m)\alpha(m+1)\cdots\alpha(n-1)\alpha(n)$ occurring between positions m and n.

In general, if S is a set and σ an infinite sequence of symbols over S—in other words, $\sigma : \mathbb{N}_0 \to S$—then $\mathsf{inf}(\sigma)$ denotes the set of symbols from S which occur infinitely often in σ. Formally, $\mathsf{inf}(\sigma) = \{s \in S \mid \exists^\omega n \in \mathbb{N}_0 : \sigma(n) = s\}$, where $\exists^\omega$ denotes the quantifier "there exist infinitely many".

We shall assume some degree of familiarity with standard results from the theory of regular languages over finite words.

1 Büchi automata

A Büchi automaton is a non-deterministic finite-state automaton which takes infinite words as input. A word is accepted if the automaton goes through some designated "good" states infinitely often while reading it.

Automata *An* automaton *is a triple* $\mathcal{A} = (S, \rightarrow, S_{in})$ *where* S *is a set of* states*,* $S_{in} \subseteq S$ *is a set of* initial states *and* $\rightarrow \subseteq S \times \Sigma \times S$ *is a* transition relation*. Normally, we write* $s \xrightarrow{a} s'$ *to denote that* $(s, a, s') \in \rightarrow$*.*

The automaton is said to be deterministic *if* S_{in} *is a singleton and* $\rightarrow$ *is a function from* $S \times \Sigma$ *to* S*.*

We could have weakened the condition for deterministic automata by permitting $\rightarrow$ to be a *partial* function from $S \times \Sigma$ to S. The present definition demands that at each state there is *precisely* one outgoing transition for each letter of the input alphabet. The weaker definition would relax this to say that there is *at most* one outgoing transition per letter. A "weak" deterministic automaton can always be converted to a "strong" deterministic automaton by adding a "dump" or "reject" state to take care of all missing transitions. Since it is often convenient to assume that deterministic automata are "complete" and never get stuck when reading their input, we shall stick to the stronger definition in this paper.

If u is a finite *non-empty* word, we write $s \xrightarrow{u}^{+} s'$ to denote the fact that there is a sequence of transitions labelled by u leading from s to s'. In other words, if $u = a_1 a_2 \cdots a_m$, then $s \xrightarrow{u}^{+} s'$ if there exist states $s_0, s_1, \ldots, s_m$ such that $s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} s_m = s'$.

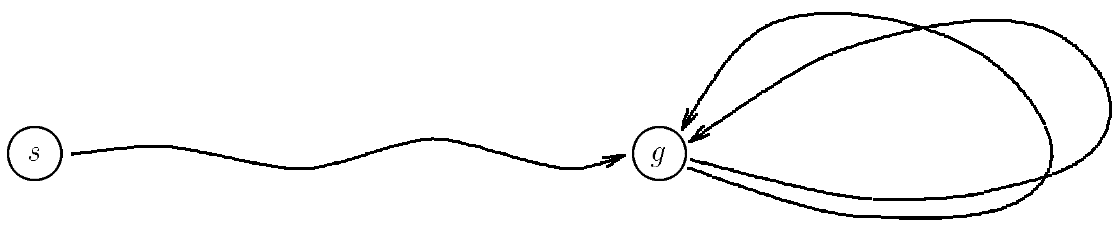

Figure 1: A typical accepting run of a Büchi automaton, with $s \in S_{in}$ and $g \in G$.

Runs *Let $\mathcal{A} = (S, \rightarrow, S_{in})$ be an automaton and $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ an input word. A* run *of $\mathcal{A}$ on α is a infinite sequence $\rho : \mathbb{N}_0 \rightarrow S$ such that $\rho(0) \in S_{in}$ and for all $i \in \mathbb{N}_0$, $\rho(i) \xrightarrow{\alpha(i)} \rho(i+1)$.*

A run of $\mathcal{A}$ on the finite word $w = a_0 a_1 \ldots a_m$ is a sequence of states $s_0 s_1 \ldots s_{m+1}$ such that $s_0 \in S_{in}$ and for all $i \in \{0, 1, \ldots, m\}$, $s_1 \xrightarrow{\alpha(i)} s_{i+1}$.

So, a run is just a "legal" sequence of states that an automaton can pass through while reading the input. In general, an input may admit many runs because of non-determinism. Since a non-deterministic automaton may have states where there are no outgoing transitions corresponding to certain input letters, it is also possible that an input admits *no* runs—in this case, every potential run leads to a state from where there is no outgoing transition enabled for the next input letter. If the automaton is deterministic, each input admits precisely one run.

Automata on finite words *A finite state automaton on finite words is a structure $(\mathcal{A}, F)$ with $\mathcal{A} = (S, \rightarrow, S_{in})$ and $F \subseteq S$. The automaton $\mathcal{A}$ accepts an input $w = a_0 a_1 \ldots a_m$ if w admits a run $s_0 s_1 \ldots s_{m+1}$ such that $s_{m+1} \in F$. The language* recognized by *$(\mathcal{A}, F)$, $L(\mathcal{A}, F)$, is the set of all finite words accepted by $(\mathcal{A}, F)$.*

Throughout this paper, we shall refer to languages recognized by finite state automata on finite words as regular languages. *In other words, a set $U \subseteq \Sigma^*$ is regular iff there is an automaton $(\mathcal{A}, F)$ such that $U = L(\mathcal{A}, F)$.*

Our goal is to study automata which recognize languages of infinite words. The first definition of such automata was proposed by Büchi [Bü60].

Büchi automata *A* Büchi automaton *is a pair $(\mathcal{A}, G)$ where $\mathcal{A} = (S, \rightarrow, S_{in})$ and $G \subseteq S$. G denotes a set of* good states. *The automaton $(\mathcal{A}, G)$* accepts *an input $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ if there is a run ρ of $\mathcal{A}$ on α such that $\mathsf{inf}(\rho) \cap G \neq \emptyset$. The language* recognized by *$(\mathcal{A}, G)$, $L(\mathcal{A}, G)$, is the set of all infinite words accepted by $(\mathcal{A}, G)$. A set $L \subseteq \Sigma^\omega$ is said to be* Büchi-recognizable *if there is a Büchi automaton $(\mathcal{A}, G)$ such that $L = L(\mathcal{A}, G)$.*

According to the definition, a Büchi automaton accepts an input if there is a run along which some subset of G occurs infinitely often. Since G is a finite set, it is easy to see that there must actually be a state $g \in G$ which occurs infinitely often along σ. In other words, if we regard the state space of a Büchi automaton as a graph, an accepting run traces an infinite path which starts at some state s in S_{in}, reaches a good state $g \in G$ and, thereafter, keeps looping back to g infinitely often (see Figure 1).

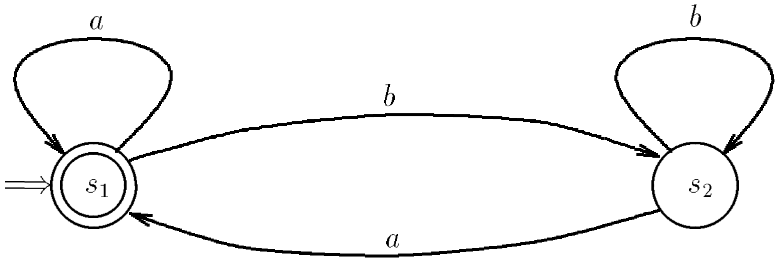

Figure 2: A Büchi automaton for L (Example 1.1)

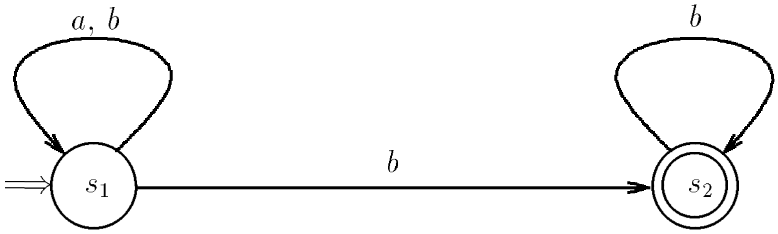

Figure 3: A Büchi automaton for $\overline{L}$ (Example 1.1)

Example 1.1 Consider the alphabet $\Sigma = \{a, b\}$. Let $L \subseteq \Sigma^\omega$ consist of all infinite words α such that there are infinitely many occurrences of a in α. Figure 2 shows a Büchi automaton recognizing L. The initial state is marked by a double arrow. There is only one good state, which is indicated with a double circle. In this automaton, all transitions labelled a lead into the good state and, conversely, all transitions coming into the good state are labelled a. From this, it follows that the automaton accepts an infinite word iff it has infinitely many occurrences of a.

The complement of L, which we denote $\overline{L}$, is the set of all infinite words α such that α has only finitely many occurrences of a. An automaton recognizing $\overline{L}$ is shown in Figure 3. The automaton guesses a point in the input beyond which it will see no more a's—such a point must exist in any input with only a finite number of a's. Once it has made this guess, it can process only b's—there is no transition labelled a from the second state—so if it reads any more a's it gets stuck.

□

In the example, notice that the automaton recognizing L is deterministic while the automaton for $\overline{L}$ is non-deterministic. We now show that the non-determinism in the second case is unavoidable—that is, there is *no* deterministic automaton recognizing $\overline{L}$. This means that Büchi automata are fundamentally different from their counterparts on finite inputs: we know that over finite words, deterministic automata are as powerful as non-deterministic automata.

Limit languages *Let $U \subseteq \Sigma^*$ be a language of finite strings. The limit of U, $\mathsf{lim}(U)$ is the set*

$$\{\alpha \in \Sigma^\omega \mid \exists^\omega n \in \mathbb{N}_0 : \alpha[0..n] \in U\}.$$

So, a word belongs to $\mathsf{lim}(U)$ iff it has infinitely many prefixes in U. We then have the following characterization of languages recognized by deterministic Büchi automata.

Theorem 1.2 *A language $L \subseteq \Sigma^\omega$ is recognizable by a deterministic Büchi automaton iff L is of the form $\mathsf{lim}(U)$ for some regular language $U \subseteq \Sigma^*$.*

Proof Let U be a regular language. Then, there exists a deterministic finite state automaton (DFA) of the form $(\mathcal{A}, F)$ where $\mathcal{A} = (S, \rightarrow, S_{in})$ and $F \subseteq S$ such that $(\mathcal{A}, F)$ recognizes U. It is easy to see that if we interpret F as a set of good states, the Büchi automaton $(\mathcal{A}, F)$ accepts $\mathsf{lim}(U)$.

Conversely, let L be recognized by a deterministic Büchi automaton $(\mathcal{A}, G)$. Treat G as a set of final states and let U be the language recognized by the DFA $(\mathcal{A}, G)$. Once again, it is easy to see that $L = \mathsf{lim}(U)$. $\square$

We now show that the language $\overline{L}$ of Example 1.1 is not of the form $\mathsf{lim}(U)$ for *any* language U. Recall that $\overline{L}$ is the set of all infinite words α over the alphabet $\Sigma = \{a, b\}$ such that α contains only finitely many occurrences of a.

Suppose that $\overline{L} = \mathsf{lim}(U)$ for some $U \subseteq \Sigma^*$. Since $b^\omega \in \overline{L}$, there must be some finite prefix $b^{n_1} \in U$. Since, $b^{n_1}ab^\omega \in \overline{L}$, we can then find a prefix $b^{n_1}ab^{n_2} \in U$. From the fact that $b^{n_1}ab^{n_2}ab^\omega \in \overline{L}$, we obtain a prefix $b^{n_1}ab^{n_2}ab^{n_3} \in U$. Proceeding in this way, we get an infinite sequence of words $\{b^{n_1}, b^{n_1}ab^{n_2}, b^{n_1}ab^{n_2}ab^{n_3}, \ldots\} \subseteq U$. From this it follows that the infinite word $\beta = b^{n_1}ab^{n_2}ab^{n_3}a \cdots ab^{n_i}a \cdots$ belongs to $\mathsf{lim}(U)$. But β has infinitely many occurrences of a, so it certainly does not belong to $\overline{L}$, thus contradicting the assumption that $\overline{L} = \mathsf{lim}(U)$.

From this observation and Example 1.1, we deduce the following corollary.

Corollary 1.3 *Non-deterministic Büchi automata are strictly more powerful than deterministic Büchi automata—there are languages recognized by non-deterministic Büchi automata which cannot be recognized by* any *deterministic Büchi automaton.*

1.1 Characterizing Büchi-recognizable languages

For finite words, we can characterize the class of languages recognized by non-deterministic finite state automata in a number of ways—for instance, in terms of regular expressions, or in terms of syntactic congruences. In the same spirit, we now describe a characterization of Büchi-recognizable languages of infinite words. We first need to define the ω-iteration of a set of finite words. Let $U \subseteq \Sigma^*$. Then

$$U^\omega = \{\alpha \in \Sigma^\omega \mid \alpha = u_0 u_1 u_2 \cdots \text{ where } u_i \in U \text{ for all } i \in \mathbb{N}_0\}.$$

Also, we observe that if U is a language of finite words and L is a language of infinite words, we can define the language UL of infinite words obtained by concatenating each finite word in U with an infinite word from L. Formally, $UL = \{\alpha \mid \exists u \in U : \exists \beta \in L : \alpha = u\beta\}$.

ω-regular languages *A language $L \subseteq \Sigma^\omega$ is said to be* ω-regular *if it is of the form $\bigcup_{i\in\{1,2,\ldots,n\}} U_i V_i^\omega$, where each U_i and V_i is a regular language of finite words.*[2]

Theorem 1.4 *A language is Büchi-recognizable iff it is ω-regular.*

Proof
($\Rightarrow$):
Let L be recognized by a Büchi automaton $(\mathcal{A}, G)$, where $\mathcal{A} = (S, \rightarrow, S_{in})$. We have observed earlier that each infinite word $\alpha \in L$ admits an accepting run ρ which begins in an initial state, reaches a good state g, and then loops back through g infinitely often. For $s, s' \in S$, let $V_{ss'} = \{w \in \Sigma^* \mid s \xrightarrow{w}{}^{+} s'\}$ denote the set of finite words which can lead from s to s'. It is easy to see that $V_{ss'}$ is regular—to recognize this set, use the non-deterministic automaton $(S, \rightarrow, \{s\})$ with $\{s'\}$ as the set of final states. From our observation about accepting runs, it follows that we can write $L(\mathcal{A}, G)$ as $\bigcup_{s\in S_{in},\ g\in G} V_{sg} V_{gg}^\omega$.

($\Leftarrow$):
It is not difficult to show that the set of Büchi-recognizable languages satisfies the following closure properties:

(i) If U is regular, then U^ω is Büchi-recognizable.

(ii) If U is regular and L is Büchi-recognizable then UL is Büchi recognizable.

(iii) If $L_1, L_2, \ldots, L_n$ are Büchi-recognizable, so is $\bigcup_{i\in\{1,2,\ldots,n\}} L_i$.

From this, it follows that every language of the form $\bigcup_{i\in\{1,2,\ldots,n\}} U_i V_i^\omega$, where each U_i and V_i is regular, is Büchi-recognizable. □

1.2 Constructions on Büchi automata

It turns out that the class of Büchi-recognizable languages is closed under boolean operations and projection. These operations will be crucially used when applying Büchi automata to settle decision problems in logic.

Union To show closure under finite union (which we have already assumed when proving the previous theorem!), let $(\mathcal{A}_1, G_1)$ and $(\mathcal{A}_2, G_2)$ be two Büchi automata. To construct an automaton $(\mathcal{A}, G)$ such that $L(\mathcal{A}, G) = L(\mathcal{A}_1, G_1) \cup L(\mathcal{A}_2, G_2)$, we take $\mathcal{A}$ to be the *disjoint* union of $\mathcal{A}_1$ and $\mathcal{A}_2$. Since we are permitted to have a *set* of initial states in $\mathcal{A}$, we retain the initial states from both copies. If a run of $\mathcal{A}$ starts in an initial state contributed by $\mathcal{A}_1$, it will never cross over into the state space contributed by $\mathcal{A}_2$ and vice versa. Thus, we can set the good states of $\mathcal{A}$ to be the union of the good states contributed by both components.

[2]Technically speaking, both regularity and ω-regularity are defined algebraically, in terms of homomorphisms from the given language to a finite monoid. However, we shall ignore this complication and stick to a more simple definition of ω-regularity.

Complementation Showing that Büchi-recognizable languages are closed under complementation is highly non-trivial. One problem is that we cannot determinize Büchi automata, as we have observed in Corollary 1.3. Even if we could work with deterministic automata, the formulation of Büchi acceptance is not symmetric with respect to complementation in the following sense. Suppose $(\mathcal{A}, G)$ is a deterministic Büchi automaton and α is an infinite word which does not belong to $L(\mathcal{A}, G)$. Then, the (unique) run ρ_α of $\mathcal{A}$ on α is such that $\inf(\rho_\alpha) \cap G = \emptyset$. Let $\overline{G}$ denote the complement of G. It follows that $\inf(\rho_\alpha) \cap \overline{G} \neq \emptyset$, since *some* state must occur infinitely often in ρ_α. It would be tempting to believe that the automaton $(\mathcal{A}, \overline{G})$ recognizes $\Sigma^\omega - L(\mathcal{A}, G)$. However, there may be words which admit runs which visit both G and $\overline{G}$ infinitely often. These words will be including both in $L(\mathcal{A}, \overline{G})$ as well as in $L(\mathcal{A}, G)$. So, there is no convenient way to express the complement of a Büchi condition again as a Büchi condition.

We shall postpone describing a complementation construction for Büchi automata until Section 4. Till then we shall, however, assume that we can complement these automata.

Intersection Turning to intersection, the natural way to intersect automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is to construct an automaton whose state space is the cross product of the state spaces of $\mathcal{A}_1$ and $\mathcal{A}_2$ and let both copies process the input simultaneously. For finite words, the input is accepted if each copy can generate a run which reaches a final state at the end of the word.

For infinite inputs, we have to do a more sophisticated product construction. An infinite input α should be accepted by the product system provided both copies generate runs which visit good states infinitely often. Unfortunately, there is no guarantee that these runs will ever visit good states simultaneously—for instance, it could be that the first run goes through a good state after $\alpha(0)$, $\alpha(2)$, ... while the second run enters good states after $\alpha(1)$, $\alpha(3)$, ... So, the main question is one of identifying the good states of the product system.

The key observation is that to detect that both components of the product visit good states infinitely often, one need not record *every* point where the copies visit good states; in each copy, it suffices to observe an infinite subsequence of the overall sequence of good states. So, we begin by focusing on the first copy and waiting for its run to enter a good state. When this happens, we switch attention to the other copy and wait for a good state there. Once the second copy reaches a good state, we switch back to the first copy and so on. Clearly, we will switch back and forth infinitely often iff both copies visit their respective good states infinitely often. Thus, we can characterize the good states of the product in terms of the states where one switches back and forth.

Formally, the construction is as follows. Let $(\mathcal{A}_1, G_1)$ and $(\mathcal{A}_2, G_2)$ be two Büchi automata such that $\mathcal{A}_i = (S_i, \rightarrow_i, S^i_{in})$ for $i = 1, 2$. Define $(\mathcal{A}, G)$, where $\mathcal{A} = (S, \rightarrow, S_{in})$, as follows:

- $S = S_1 \times S_2 \times \{1, 2\}$
- The transition relation $\rightarrow$ is defined as follows:

$$(s_1, s_2, 1) \xrightarrow{a} (s_1', s_2', 1) \;\text{ if }\; s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_1 \notin G_1.$$
$$(s_1, s_2, 1) \xrightarrow{a} (s_1', s_2', 2) \;\text{ if }\; s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_1 \in G_1.$$

$$(s_1, s_2, 2) \xrightarrow{a} (s_1', s_2', 2) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_2 \notin G_2.$$
$$(s_1, s_2, 2) \xrightarrow{a} (s_1', s_2', 1) \text{ if } s_1 \xrightarrow{a}_1 s_1',\ s_2 \xrightarrow{a}_2 s_2' \text{ and } s_2 \in G_2.$$

- $S_{in} = \{(s_1, s_2, 1) \mid s_1 \in S_{in}^1 \text{ and } s_2 \in S_{in}^2\}$
- $G = S_1 \times G_2 \times \{2\}$.

In the automaton $\mathcal{A}$, each product state carries an extra tag indicating whether the automaton is checking for a good state on the first or the second component. The automaton accepts if it switches focus from the second component to the first infinitely often. (Notice that we could equivalently have defined G to be the set $G_1 \times S_2 \times \{1\}$.) It is not difficult to verify that $L(\mathcal{A}, G) = L(\mathcal{A}_1, G_1) \cap L(\mathcal{A}_2, G_2)$.

Projection Let Σ_1 and Σ_2 be alphabets such that $|\Sigma_2| \leq |\Sigma_1|$. A *projection function* from Σ_1 to Σ_2 is a surjective map $\pi : \Sigma_1 \to \Sigma_2$. We can extend π from individual letters to words as usual: if $\alpha \in \Sigma_1^\omega$, $\pi(\alpha)$ denotes the word β where $\beta(i) = \pi(\alpha(i))$ for all i in $\mathbb{N}_0$.

Let $L \subseteq \Sigma_1^\omega$. Then $\pi(L)$, the projection of L via π, is the language $\{\beta \in \Sigma_2^\omega \mid \exists \alpha \in L : \beta = \pi(\alpha)\}$. It is easy to verify that if L is Büchi-recognizable, then so is $\pi(L)$. Let $(\mathcal{A}_1, G_1)$ be an automaton recognizing L, where $\mathcal{A}_1 = (S_1, \to_1, S_{in}^1)$. We construct an automaton $\mathcal{A}_2 = (S_2, \to_2, S_{in}^2)$ over Σ_2 as follows: set $S_2 = S_1$, $S_{in}^2 = S_{in}^1$ and $s \xrightarrow{b}_2 s'$ iff $s \xrightarrow{a}_1 s'$ for some $a \in \Sigma_1$ such that $\pi(a) = b$. It is easy to verify that $(\mathcal{A}_2, G_1)$ recognizes $\pi(L)$.

Emptiness In applications, we will need to be able to check whether the language accepted by a Büchi automaton is empty. To do this, we recall our observation that any accepting run of a Büchi automaton must begin in an initial state, reach a final state g and then cycle back to g infinitely often.

If we ignore the labels on the transitions, we can regard the state space of a Büchi automaton $(\mathcal{A}, G)$ as a directed graph $G_\mathcal{A} = (V_\mathcal{A}, E_\mathcal{A})$ where $V_\mathcal{A} = S$ and $(s, s') \in E_\mathcal{A}$ iff for some $a \in \Sigma$, $s \xrightarrow{a} s'$. Recall that a set of vertices X in a directed graph is a *strongly connected component* iff for every pair of vertices $v, v' \in X$, there is a path from v to v'. Clearly, $L(\mathcal{A}, G)$ is non-empty iff there is a strongly connected component X in $G_\mathcal{A}$ such that X contains a vertex g from G and X is reachable from one of the initial states. We thus have the following theorem.

Theorem 1.5 *The emptiness problem for Büchi automata is decidable.*

Notice that it is sufficient to analyze *maximal* strongly connected components in $G_\mathcal{A}$ in order to check that $L(\mathcal{A}, G) \neq \emptyset$. Computing the maximal strongly connected components of a directed graph can be done in time linear in the size of the graph [AHU74], where the size of a graph $G = (V, E)$ is, as usual, given by $|V| + |E|$. Checking reachability can also be done in linear time. So, if $\mathcal{A}$ has n states, checking that $L(\mathcal{A}, G) \neq \emptyset$ can be done in time $O(n^2)$.

2 The logic of sequences

Büchi's original motivation for studying automata on infinite inputs was to solve a decision problem from logic. He discovered a deep and beautiful connection between ω-regular languages and sets of models of formulas in certain logics.

S1S

The logic that Büchi considered was the *monadic second-order theory of one successor*, abbreviated as S1S. This logic is interpreted over the set $\mathbb{N}_0$ of natural numbers. In general, second-order logic permits quantification over relations and functions, unlike first-order logic, which permits quantification over just individual variables. However, the fact that we are dealing with a "monadic" second-order logic restricts this extra power to quantification over one-place relations. Since a one-place relation is just a subset, this effectively means that we can quantify over individual elements of $\mathbb{N}_0$ and subsets of $\mathbb{N}_0$. The fact that we are dealing with "one successor" means we are talking about $\mathbb{N}_0$ with the usual ordering where each element has a unique successor. Permitting two successors, for instance, would produce the infinite binary tree which has countably many nodes but has two successors for each node.

Formally, the logical language S1S is defined as follows.

Terms A *term* in S1S is built up from the constant $\mathtt{0}$ and *individual variables* $x, y, \ldots$ by application of the *successor function* $succ$. Thus, the following are terms: $\mathtt{0}$, $succ(x)$, $succ(succ(succ(0)))$, $succ(succ(y))$, ...

Atomic formulas Let $t, t', \ldots$ be terms. An *atomic formula* is of the form $t = t'$ or $t \in X$, where X is a *set variable*.

Formulas A *formula* is built up from atomic formulas using the boolean connectives $\neg$ (not) and $\vee$ (or), together with the existential quantifier $\exists$. The quantifier $\exists$ can be applied to *both* individual and set variables—one can write $\exists x$ and $\exists X$. In other words, if φ and ψ are inductively assumed to be formulas, so are $\neg\varphi$, $\varphi \vee \psi$, $(\exists x)\ \varphi$ and $(\exists X)\ \varphi$.

In addition, we can define the remaining boolean connectives like $\wedge$ (and), $\Rightarrow$ (if-then), $\Leftrightarrow$ (iff) as usual, in terms of $\neg$ and $\vee$: for instance, $\varphi \Rightarrow \psi$ is defined as $(\neg\varphi \vee \psi)$. We also have the universal quantifier $\forall$ which is the dual of $\exists$: $(\forall x)\ \varphi \stackrel{def}{=} \neg((\exists x)\ \neg\varphi)$ and $(\forall X)\ \varphi \stackrel{def}{=} \neg((\exists X)\ \neg\varphi)$.

Assigning truth values to formulas Formulas are interpreted over $\mathbb{N}_0$. The constant $\mathtt{0}$ denotes the number 0. Individual variables $x, y, \ldots$ are interpreted as natural numbers—that is, elements of $\mathbb{N}_0$. The function $succ$ corresponds to adding one: $succ(x)$ denotes the number which is one greater than the interpretation of x. Thus, the term $succ(succ(succ(\mathtt{0})))$ represents the number 3. And, if the current interpretation of x is the number 47 then $succ(x)$ denotes 48.

The connective $=$ used in defining atomic formulas denotes equality, as usual. Thus $t = t'$ is true provided t and t' denote the same natural number.

Set variables like X, Y, ... are interpreted as subsets of $\mathbb{N}_0$. The atomic formula $t \in X$ is true iff the number denoted by t belongs to the set denoted by X.

Once the interpretation of atomic formulas has been fixed, the meaning of compound formulas involving $\neg$, $\vee$ and $\exists$ is the "natural" one.

Let φ be a formula. A variable is said to occur free in φ if it is not within the scope of a quantifier. For instance, in the formula $(\exists x)(\forall Y)\ (\mathsf{0} \in Y) \vee (x = y) \vee (x \in X)$, the variables y and X occur free. Variables which do not occur free are said to be bound. In the preceding formula, x and Y are bound. We write $\varphi(x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_\ell)$ to indicate that all the variables which occur free in φ come from the set $\{x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_\ell\}$. Let $\overrightarrow{X} = (x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_\ell)$. To assign a truth value to the formula $\varphi(\overrightarrow{X})$, we have to first fix an interpretation of the variables in $\overrightarrow{X}$. In other words, we must map each individual variable x_i to a natural number $m_i \in \mathbb{N}_0$ and each set variable X_j to a subset $M_j \subseteq \mathbb{N}_0$. Let $\overrightarrow{M} = (m_1, m_2, \ldots, m_k, M_1, M_2, \ldots, M_\ell)$. We write $\overrightarrow{M} \models \varphi(\overrightarrow{X})$ to denote that φ is true under the interpretation $\{x_i \mapsto m_i\}_{i \in \{1,2,\ldots,k\}}$ and $\{X_i \mapsto M_i\}_{i \in \{1,2,\ldots,\ell\}}$. Rather than go into formal details, we look at some illustrative examples.

Example 2.1

(i) Let $\mathsf{Sub}(X, Y) = (\forall x)\ x \in X \Rightarrow x \in Y$.

Then $(M, N) \models \mathsf{Sub}(X, Y)$ iff $M \subseteq N$.

(ii) Let $\mathsf{Zero}(X) = (\exists x)\ [x \in X \wedge \neg(\exists y)(y < x)]$.

This formula asserts that X contains an element which has no predecessors in $\mathbb{N}_0$. Thus, $M \models \mathsf{Zero}(X)$ iff $0 \in M$.

(iii) Let $\mathsf{Lt}(x, y) = \ (\forall Z)[succ(x) \in Z \wedge (\forall z)(z \in Z \Rightarrow succ(z) \in Z)] \Rightarrow (y \in Z)$.

Then $(m, n) \models \mathsf{Lt}(x, y)$ iff $m < n$. What the formula asserts is that any set Z which contains $x+1$ and is closed with respect to the successor function must also contain y.

(iv) Let $\mathsf{Sing}(X) = (\exists Y)\ \ [\mathsf{Sub}(Y, X) \wedge (Y \neq X) \wedge$
$\neg(\exists Z)\ \ (\mathsf{Sub}(Z, Y) \wedge (Z \neq X) \wedge (Z \neq Y))]$.

In this formula, $X \neq Y$ abbreviates $\neg(X = Y)$, where $X = Y$ is itself an abbreviation for $\mathsf{Sub}(X, Y) \wedge \mathsf{Sub}(Y, X)$. The formula asserts that X has only one proper subset, which is Y. This is true only for singletons, where Y is the empty set. So, $M \models \mathsf{Sing}(X)$ iff M is a singleton $\{m\}$.

□

A *sentence* is a formula in which no variables occur free. A sentence φ is either true or false—we do not have to interpret any variables to assign meaning to φ. For instance consider the sentence

$$(\forall X)\ [\mathsf{0} \in X \wedge (\forall x)\ (x \in X \Rightarrow succ(x) \in X)] \Rightarrow (\forall x)\ x \in X.$$

This sentence is true: it expresses the familiar property of mathematical induction for subsets of $\mathbb{N}_0$—if a set of natural numbers contains 0 and is closed with respect to the successor function, then the set in fact includes all of $\mathbb{N}_0$.

Satisfiability An S1S formula $\varphi(x_1,\ldots,x_k,X_1,\ldots,X_\ell)$ is said to be *satisfiable* if we can choose $\overrightarrow{M} = (m_1,\ldots,m_k,M_1,\ldots,M_\ell)$ such that $\overrightarrow{M} \models \varphi(\overrightarrow{X})$.

Büchi showed how to associate an ω-regular language L_φ with each S1S formula φ, such that every word in L_φ represents an interpretation for the free variables in φ under which the formula φ evaluates to true. Moreover, every interpretation which makes φ true is represented by some word in L_φ. Thus, φ is satisfiable iff there is some interpretation which makes it true iff L_φ is non-empty. The language L_φ is defined over the alphabet $\{0,1\}^m$, where m is the number of free variables in φ.

In fact, Büchi showed that the converse is also true. Let us say that a language $L \subseteq (\{0,1\}^m)^\omega$ is S1S-definable if $L = L_\varphi$ for some S1S formula φ. We can always embed an arbitrary alphabet Σ as a subset of $\{0,1\}^m$ for some suitable choice of m. In this way, any language $L \subseteq \Sigma^\omega$ can be converted into an equivalent language $L_{\{0,1\}}$ over $\{0,1\}^m$. Büchi showed that if L is ω-regular, then $L_{\{0,1\}}$ is S1S-definable.

Thus, the notions of S1S-definability and ω-regularity are equivalent. The rest of this section will be devoted to formally stating and proving this result.

We begin by defining L_φ for an S1S formula $\varphi(x_1,x_2,\ldots,x_k,X_1,X_2,\ldots,X_\ell)$. Let $\overrightarrow{M} = (m_1,m_2,\ldots,m_k,M_1,M_2,\ldots,M_\ell)$ such that $\overrightarrow{M} \models \varphi(\overrightarrow{X})$. We can associate with $\overrightarrow{M}$ an infinite word α_M over $\{0,1\}^{k+\ell}$ which represents the characteristic function of $\overrightarrow{M}$. For $i \in \mathbb{N}_0$, and $j \in \{1,2,\ldots,k{+}\ell\}$, let $\alpha_M(i)(j)$ denote the j^{th} component of $\alpha_M(i)$. Then for $i \in \mathbb{N}_0$ and $j \in \{1,2,\ldots,k\}$, $\alpha_M(i)(j) = 1$ iff $i = m_j$ and $\alpha_M(i)(j) = 0$ iff $i \neq m_j$. Similarly, for $i \in \mathbb{N}_0$, and $j \in \{k{+}1,k{+}2,\ldots,k{+}\ell\}$, $\alpha_M(i)(j) = 1$ iff $i \in M_j$ and $\alpha_M(i)(j) = 0$ iff $i \notin M_j$. Then

$$L_\varphi = \{\alpha_M \mid \overrightarrow{M} \models \varphi(\overrightarrow{X})\}.$$

Next we define the $\{0,1\}$-image $L_{\{0,1\}}$ corresponding to a language L over an arbitrary alphabet Σ. Let $\Sigma = \{a_1,a_2,\ldots,a_m\}$. Then, each word $\alpha \in \Sigma^\omega$ can be represented by a word $\alpha_{\{0,1\}}$ over $\{0,1\}^m$, where for all $i \in \mathbb{N}_0$ and $j \in \{1,2,\ldots,m\}$, $\alpha_{\{0,1\}}(i)(j) = 1$ if $\alpha(i) = a_j$ and $\alpha_{\{0,1\}}(i)(j) = 0$ if $\alpha(i) \neq a_j$. Then

$$L_{\{0,1\}} = \{\alpha_{\{0,1\}} \mid \alpha \in L\}.$$

We can now state Büchi's result more precisely.

Theorem 2.2

(i) Let φ be an S1S formula. Then L_φ is an ω-regular language.

(ii) Let L be an ω-regular language. Then $L_{\{0,1\}}$ is S1S-definable.[3]

Proof

(i) To show that L_φ is ω-regular, we proceed by induction on the structure of φ. To do this, it will be convenient to cut down the language S1S to an equivalent language $S1S_0$ which has a simpler syntax.

[3]We have fixed a apecific embedding of Σ into $\{0,1\}^m$ which is relatively easy to describe in S1S. In general, we can choose *any* embedding for defining $L_{\{0,1\}}$ and the result will still go through.

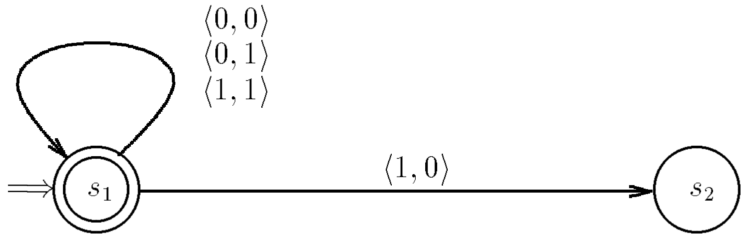

Figure 4: Büchi automaton for the atomic formula $X \subseteq Y$

Formally, in $\mathrm{S1S}_0$ we do not have individual variables x_i—there are *only* set variables X_j. The atomic formulas are of the form $X \subseteq Y$ and $succ(X, Y)$. The first formula is true if X is a subset of Y while the second is true if X and Y are singletons $\{x\}$ and $\{y\}$ respectively and $y = x+1$.

We now argue that every S1S formula φ can be converted to an $\mathrm{S1S}_0$ formula φ_0 such that $L_\varphi = L_{\varphi_0}$.

We begin by eliminating nested applications of the successor function. For instance, if the S1S formula contains the atomic formula $succ(succ(x)) \in X$, we write instead

$$(\exists y)(\exists z)\ y = succ(x) \wedge z = succ(y) \wedge z \in X.$$

We then eliminate formulas of the form $\mathsf{0} \in X$ using the formula $\mathsf{Zero}(X)$ defined in Example 2.1.

Finally, we eliminate singleton variables using the formula Sing from Example 2.1. For instance, we rewrite $(\forall x)(\exists y)\ succ(x) = y \wedge y \in Z$ as

$$(\forall X)\ (\mathsf{Sing}(X) \Rightarrow [(\exists Y)\ \mathsf{Sing}(Y) \wedge succ(X, Y) \wedge Y \subseteq Z]).$$

Notice that we can uniformly replace $\mathsf{Sub}(X, Y)$ by $X \subseteq Y$ in Sing since $X \subseteq Y$ is an atomic formula in SIS_0.

We now construct for each $\mathrm{S1S}_0$ formula φ, a Büchi automaton $(\mathcal{A}_\varphi, G_\varphi)$ recognizing L_φ.

For the atomic formula $X \subseteq Y$, the corresponding automaton over $\{0,1\} \times \{0,1\}$ is shown in Figure 4. This automaton accepts any input word which does not contain $\langle 1, 0\rangle$—if $\alpha(i) = \langle 1, 0\rangle$, in the corresponding interpretation, $i \in X$ but $i \notin Y$, thus violating the requirement that $X \subseteq Y$.

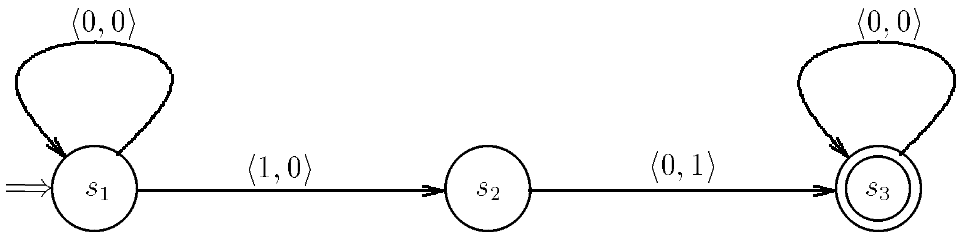

Figure 5: Büchi automaton for the atomic formula $succ(X, Y)$

The other atomic formula is $succ(X, Y)$. The corresponding automaton is shown in Figure 5. This automaton accepts inputs of the form $\langle 0,0\rangle^i\langle 1,0\rangle\langle 0,1\rangle\langle 0,0\rangle^\omega$, $i \in \mathbb{N}_0$, corresponding to the interpretation where $X = \{i\}$ and $Y = \{i+1\}$.

For the induction step, we need to consider the connectives $\neg$, $\vee$ and $\exists X$.

Let $\varphi = \neg\psi$. Then, L_φ is the complement of L_ψ. By the induction hypothesis, there exists a Büchi automaton $(\mathcal{A}_\psi, G_\psi)$ which recognizes L_ψ. As we mentioned earlier, there is an effective way to construct an automaton $(\mathcal{A}_\varphi, G_\varphi)$ recognizing the complement of L_ψ. The details will be described in Section 4.

If $\varphi = \varphi_1 \vee \varphi_2$, then $L_\varphi = L_{\varphi_1} \cup L_{\varphi_2}$. By the induction hypothesis, there exist automata $(\mathcal{A}_{\varphi_1}, G_{\varphi_1})$ and $(\mathcal{A}_{\varphi_2}, G_{\varphi_2})$ such that $L(\mathcal{A}_{\varphi_i}, G_{\varphi_i}) = L_{\varphi_i}$ for $i = 1, 2$. We have seen in Section 1.2 how to construct an automaton $(\mathcal{A}_\varphi, G_\varphi)$ such that $L(\mathcal{A}_\varphi, G_\varphi) = L_{\varphi_1} \cup L_{\varphi_2}$.

Finally, if $\varphi = (\exists X_1)\ \psi(X_1, X_2, \ldots X_m)$, the language L_φ corresponds to the projection of L_ψ via the function $\pi : \{0,1\}^m \to \{0,1\}^{m-1}$ which erases the first component of each m-tuple in $\{0,1\}^m$. A word of $(m-1)$-tuples belongs to L_φ if it can be padded out with an extra component so that the resulting word over m-tuples is in L_ψ. This padding operation corresponds to guessing a witness for the set X_1. The automaton $(\mathcal{A}_\varphi, G_\varphi)$ recognizing L_φ can be obtained from $(\mathcal{A}_\psi, G_\psi)$, the automaton recognizing L_ψ, as described in Section 1.2.

In this way, we inductively associate with each $\mathrm{S1S}_0$ formula φ, a Büchi automaton $(\mathcal{A}_\varphi, G_\varphi)$ such that $L_\varphi = L(\mathcal{A}_\varphi, G_\varphi)$.

There is a slight technicality involved when we deal with sentences. Notice that every time we encounter an existential quantifier, we eliminate one component from the input alphabet of $\mathcal{A}_\varphi$. If φ is a sentence—that is, all variables in φ are bound—we would have erased *all* components of the input by the time we construct $\mathcal{A}_\varphi$. In other words, $\mathcal{A}_\varphi$ will be an input-free automaton whose states and transitions define an unlabelled directed graph. If we were dealing with languages of finite words, we could say that a sentence φ is true iff L_φ contains the empty word. Since the empty word is not a member of Σ^ω, we would have to slightly modify our definition of L_φ to accommodate this case cleanly in our framework. However, we shall not worry too much about this since it is clear that a sentence φ is true iff there is an unlabelled path in the graph corresponding to $\mathcal{A}_\varphi$ which begins at some initial state and visits a good state infinitely often.

(ii) Let $(\mathcal{A}, G)$ be a Büchi automaton recognizing $L \subseteq \Sigma^\omega$, where $\Sigma = \{a_1, a_2, \ldots, a_m\}$ and $\mathcal{A} = (S, \to, S_{in})$, with $S = \{s_1, s_2, \ldots, s_k\}$. We use free variables $A_1, A_2, \ldots, A_m$ to describe each infinite word over Σ—the variable A_i describes the positions in the input where letter a_i occurs. We then use existentially quantified variables $S_1, S_2, \ldots, S_k$ to describe runs of the automaton over the input—the variable S_j describes the positions in the run where the automaton is in state s_j.

The formula φ_L can then be written as follows:

$$
\begin{array}{lllll}
& (\exists S_1)(\exists S_2)\cdots(\exists S_k) & & & \\
& (\forall x) & \displaystyle\bigvee_{i\in\{1,2,\ldots,m\}} (x \in A_i) \wedge & \displaystyle\bigwedge_{i\in\{1,2,\ldots,m\}} & \left(x \in A_i \Rightarrow \displaystyle\bigvee_{j\neq i} x \notin A_j\right) \\
\wedge & (\forall x) & \displaystyle\bigvee_{i\in\{1,2,\ldots,k\}} (x \in S_i) \wedge & \displaystyle\bigwedge_{i\in\{1,2,\ldots,k\}} & \left(x \in S_i \Rightarrow \displaystyle\bigvee_{j\neq i} x \notin S_j\right) \\
\wedge & & \displaystyle\bigvee_{s_i\in S_{in}} (\mathsf{0} \in S_i) & & \\
\wedge & (\forall x) & \displaystyle\bigvee_{(s_i,a_j,s_k)\in\rightarrow} (x \in S_i) \wedge (x \in A_j) \wedge (succ(x) \in S_k) & & \\
\wedge & & \displaystyle\bigvee_{s_i\in G} (\forall x)(\exists y)\,(x < y) \wedge (y \in S_i) & &
\end{array}
$$

The first two lines capture the fact that with each position $x \in \mathbb{N}_0$ we associate precisely one input letter and one state from the run. The third line asserts that the position 0 corresponds to an initial state. The fourth line guarantees that the sequence of states represented by $S_1, S_2, \ldots, S_k$ is a run. The last line then says that the run is good—that is, some final state appears infinitely often. It is not difficult to verify that $L_{\varphi_L} = L_{\{0,1\}}$.

□

3 Stronger acceptance conditions

As we saw earlier, deterministic Büchi automata cannot recognize all ω-regular languages (Corollary 1.3). It turns out that we can define classes of deterministic automata which recognize ω-regular languages by strengthening the acceptance criterion. We begin with the definition proposed by Muller [Mu63].

Muller automata *A* Muller automaton *is a pair* $(\mathcal{A}, \mathcal{T})$ *where* $\mathcal{A} = (S, \rightarrow, S_{in})$ *is an automaton, as before, and* $\mathcal{T} = \langle F_1, F_2, \ldots, F_k\rangle$ *is an* acceptance table *with* $F_i \subseteq S$ *for* $i \in \{1, 2, \ldots, k\}$.

The automaton $(\mathcal{A}, \mathcal{T})$ *accepts an input* $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ *if there is a run* ρ *of* $\mathcal{A}$ *on* α *such that* $\mathsf{inf}(\rho) \in \mathcal{T}$*—that is,* $\mathsf{inf}(\rho) = F_i$ *for some* $i \in \{1, 2, \ldots, k\}$.

The acceptance table of a Muller automaton places a much more stringent requirement on runs than a Büchi condition does. The table entry F_i makes a positive demand on the states in F_i, as well as a negative demand on the states in $S - F_i$—states in F_i must all be visited infinitely often while states outside F_i must be visited only finitely often. In other words, for a run ρ to satisfy the table entry F_i, after some point it must "settle down" in the set F_i and visit all the states in this set infinitely often without making transitions to any state outside F_i.

Example 3.1 Recall the language L over $\{a, b\}$ defined in Example 1.1—L contains all words which contain infinitely many occurrences of a. We saw that the deterministic automaton with two states shown in Figure 6 recognizes L with a Büchi condition $\{s_1\}$.

To accept L using a Muller condition, we retain the same automaton and set the acceptance table to $\langle\{s_1\}, \{s_1, s_2\}\rangle$.

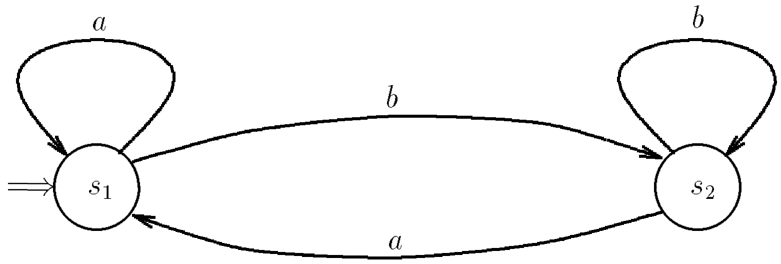

Figure 6: Automaton for L and $\overline{L}$ (Example 3.1)

We also saw that $\overline{L}$, the complement of L could not be recognized by *any* deterministic Büchi automaton. However, it is easy to verify that $\overline{L}$ can be recognized with a Muller condition by using the *same* automaton as for L, but with the acceptance table given by $\langle\{s_2\}\rangle$. $\Box$

Simulations The example shows that deterministic Muller automata are strictly more powerful than deterministic Büchi automata. It is quite straightforward to simulate a Büchi automaton by a Muller automaton—we construct an entry in the Muller table for each subset of states which contains a good state. Formally, let $(\mathcal{A}, G)$ be a Büchi automaton, where $\mathcal{A} = (S, \rightarrow, S_{in})$. The corresponding Muller automaton is given by $(\mathcal{A}, \mathcal{T}_G)$ where $\mathcal{T}_G = \{F \subseteq S \mid F \cap G \neq \emptyset\}$. It is easy to see that $L(\mathcal{A}, G) = L(\mathcal{A}, \mathcal{T}_G)$—any successful run of the Büchi automaton will satisfy one of the entries in the Muller table. Conversely, any run which satisfies an entry in $\mathcal{T}_G$ must visit a good state infinitely often. Notice that the Muller automaton $(\mathcal{A}, \mathcal{T}_G)$ is deterministic iff the original automaton $(\mathcal{A}, G)$ was deterministic: this simulation neither introduces nor removes any non-determinism.

Conversely, any Muller automaton can be simulated by a *non-deterministic* Büchi automaton. Let $(\mathcal{A}, \mathcal{T})$ be a Muller automaton, where $\mathcal{T} = \langle F_1, F_2, \ldots, F_k\rangle$. For each $i \in \{1, 2, \ldots, k\}$, we construct a Büchi automaton $(\mathcal{A}_i, G_i)$ such that $(\mathcal{A}_i, G_i)$ accepts an input α iff there is a run ρ of $(\mathcal{A}, \mathcal{T})$ on α with $\mathsf{inf}(\rho) = F_i$. It is easy to see that $L(\mathcal{A}, \mathcal{T}) = \bigcup_{i \in \{1,2,\ldots,k\}} L(\mathcal{A}_i, G_i)$. As described in Section 1.2, we can then construct a Büchi automaton $(\mathcal{A}_{\mathcal{T}}, G_{\mathcal{T}})$ which recognizes $L(\mathcal{A}, \mathcal{T})$.

To construct $(\mathcal{A}_i, G_i)$ we proceed as follows. When reading an input α, $\mathcal{A}_i$ simulates a run of $\mathcal{A}$. At some point, $\mathcal{A}$ non-deterministically decides that no more states from $S - F_i$ will occur along the run being simulated. After this guess is made, $\mathcal{A}_i$ will only simulate moves which stay within F_i. At the same time, $\mathcal{A}_i$ repeatedly cycles through F_i, checking that all states from F_i are seen infinitely often.

Let $\mathcal{A} = (S, \rightarrow, S_{in})$ and $F_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$. Then $(\mathcal{A}_i, G_i)$, with $\mathcal{A}_i = (S_i, \rightarrow_i, S^i_{in})$, is defined as follows:

- $S_i = \{(s, \mathsf{finite}) \mid s \in S\} \cup \{(s, \mathsf{infinite}, j) \mid s \in F_i, j \in \{0, 1, \ldots, m{-}1\}\}$.
- The transition relation $\rightarrow_i$ is given as follows:

$$(s, \mathsf{finite}) \xrightarrow{a}_i (s', \mathsf{finite}) \text{ if } s \xrightarrow{a} s'.$$

$(s, \mathsf{finite}) \xrightarrow{a}_i (s', \mathsf{infinite}, 0)$ if $s \xrightarrow{a} s'$ and $s' \in F_i$.

$(s, \mathsf{infinite}, k) \xrightarrow{a}_i (s', \mathsf{infinite}, k)$ if $s \xrightarrow{a} s'$, $s' \in F_i$ and $s \neq s_{i_{k+1}}$.

$(s, \mathsf{infinite}, k) \xrightarrow{a}_i (s', \mathsf{infinite}, (k{+}1) \bmod m)$ if $s \xrightarrow{a} s'$, $s' \in F_i$ and $s = s_{i_{k+1}}$.

- $S^i_{in} = \{(s, \mathsf{finite}) \mid s \in S_{in}\}$.
- $G_i = \{(s_{i_m}, \mathsf{infinite}, m{-}1)\}$.

These two simulation constructions show that the class of Muller-recognizable languages coincides with the class of Büchi-recognizable languages. In other words, Muller automata also recognize ω-regular languages.

However, Example 3.1 suggests that *deterministic* Muller automata may suffice for recognizing all ω-regular languages. In fact, this is the case—this non-trivial result was proved by McNaughton [Mc66].

Theorem 3.2 (McNaughton) *Every ω-regular language is recognized by a* deterministic *Muller automaton.*

We shall prove McNaughton's result indirectly in Section 4. Notice that McNaughton's theorem, combined with the simulation constructions described above, yields a complementation construction for Büchi automata. This is because complementing deterministic Muller automata is easy. Let $(\mathcal{A}, \mathcal{T})$ be a deterministic Muller automaton, where $\mathcal{A} = (S, \rightarrow, S_{in})$. Let $\overline{\mathcal{T}} = \{F \subseteq S \mid F \notin \mathcal{T}\}$. It is straightforward to verify that $L(\mathcal{A}, \overline{\mathcal{T}}) = \Sigma^\omega - L(\mathcal{A}, \mathcal{T})$. So, to complement a Büchi automaton $(\mathcal{A}, G)$, we first convert it into an equivalent deterministic Muller automaton $(\mathcal{A}, \mathcal{T})$ using McNaughton's theorem. We then simulate $(\mathcal{A}, \overline{\mathcal{T}})$ using the construction described earlier to get a Büchi automaton $(\mathcal{A}_{\overline{\mathcal{T}}}, G_{\overline{\mathcal{T}}})$ which accepts the complement of $L(\mathcal{A}, G)$.

Rather than follow this route, we shall describe an alternative determinization construction due to Safra [Sa88]. Safra's construction converts a Büchi automaton to a deterministic automaton with a pairs table. Acceptance in terms of a pairs table first described by Rabin [Ra69].

Rabin automata *A* Rabin automaton *is a structure $(\mathcal{A}, \mathcal{PT})$ where $\mathcal{A} = (S, \rightarrow, S_{in})$ is an automaton, as before, and $\mathcal{PT} = \langle (G_1, R_1), (G_2, R_2), \ldots, (G_k, R_k)\rangle$ is a* pairs table *with $G_i, R_i \subseteq S$ for $i \in \{1, 2, \ldots, k\}$.*

The automaton $(\mathcal{A}, \mathcal{PT})$ accepts an input $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ if there is a run ρ of $\mathcal{A}$ on α such that for some $i \in \{1, 2, \ldots, k\}$, $\mathsf{inf}(\rho) \cap G_i \neq \emptyset$ and $\mathsf{inf}(\rho) \cap R_i = \emptyset$.

Thus each pair (G_i, R_i) in the pairs table of a Rabin automaton specifies a positive and a negative requirement on the run, as in the acceptance table of a Muller automaton. The positive entry G_i is just a Büchi condition while the negative entry R_i is like the one specified for $S - F_i$ by an entry F_i in a Muller acceptance table. If we think of G_i and R_i as "green lights" and "red lights", a run ρ satisfies (G_i, R_i) if some green light from G_i flashes infinitely often and no red light from R_i flashes infinitely often.

Returning to Example 3.1, the language L is accepted by the automaton of Figure 6 with the pairs table $\langle(\{s_1\}, \emptyset)\rangle$, while $\overline{L}$ is accepted by the same automaton with the pairs table $\langle(\{s_2\}, \{s_1\})\rangle$.

Büchi automata can be trivially simulated by Rabin automata—if $(\mathcal{A}, G)$ is a Büchi automaton, the corresponding Rabin automaton is $(\mathcal{A}, \mathcal{PT}_G)$, where $\mathcal{PT}_G = \langle\{G, \emptyset\}\rangle$.

Conversely, we can simulate Rabin automata by Büchi automata using a construction similar to the one for simulating Muller automata by Büchi automata. As before, it suffices to construct a separate Büchi automaton $(\mathcal{A}_i, G'_i)$ for each entry (G_i, R_i) in the pairs table of a Rabin automaton $(\mathcal{A}, \mathcal{PT})$. The automaton $(\mathcal{A}_i, G'_i)$ simulates a run of $\mathcal{A}$ and guesses when no more states from R_i will be seen. It then checks that states from G_i occur infinitely often.

Let $\mathcal{A} = (S, \rightarrow, S_{in})$ and $\mathcal{PT} = \langle(G_1, R_1), (G_2, R_2), \ldots, (G_k, R_k)\rangle$. Then $(\mathcal{A}_i, G'_i)$, with $\mathcal{A}_i = (S_i, \rightarrow_i, S^i_{in})$, is defined as follows:

- $S_i = \{(s, \mathsf{finite}) \mid s \in S\} \cup \{(s, \mathsf{infinite}, j) \mid s \in (S - R_i), j \in \{0, 1\}\}$.
- The transition relation $\rightarrow_i$ is given as follows:

$$(s, \mathsf{finite}) \xrightarrow{a}_i (s', \mathsf{finite}) \text{ if } s \xrightarrow{a} s'.$$
$$(s, \mathsf{finite}) \xrightarrow{a}_i (s', \mathsf{infinite}, 0) \text{ if } s \xrightarrow{a} s' \text{ and } s' \notin R_i.$$
$$(s, \mathsf{infinite}, 0) \xrightarrow{a}_i (s', \mathsf{infinite}, 0) \text{ if } s \xrightarrow{a} s',\ s' \notin R_i \text{ and } s \notin G_i.$$
$$(s, \mathsf{infinite}, 0) \xrightarrow{a}_i (s', \mathsf{infinite}, 1) \text{ if } s \xrightarrow{a} s',\ s' \notin R_i \text{ and } s \in G_i.$$
$$(s, \mathsf{infinite}, 1) \xrightarrow{a}_i (s', \mathsf{infinite}, 0) \text{ if } s \xrightarrow{a} s' \text{ and } s' \notin R_i.$$

- $S^i_{in} = \{(s, \mathsf{finite}) \mid s \in S_{in}\}$.
- $G'_i = \{(s, \mathsf{infinite}, 1) \mid s \in (S - R_i)\}$.

Notice that a Rabin automaton can also simulated by a Muller automaton in quite a straightforward manner. Let $(\mathcal{A}, \mathcal{PT})$ be a Rabin automaton, where $\mathcal{A} = (S, \rightarrow, S_{in})$ and $\mathcal{PT} = \langle(G_1, R_1), (G_2, R_2), \ldots, (G_k, R_k)\rangle$. Each pair (G_i, R_i) generates a Muller table $\mathcal{T}_i = \{F \subseteq (S - R_i) \mid F \cap G_i \neq \emptyset\}$. Let $\mathcal{T} = \bigcup_{i \in \{1,2,\ldots,k\}} \mathcal{T}_i$. It is easy to see that $(\mathcal{A}, \mathcal{T})$ recognizes $L(\mathcal{A}, \mathcal{PT})$. Once again, since we have not modified $\mathcal{A}$, the simulating automaton is deterministic iff the original automaton was.

To simulate Muller automata using Rabin automata one has to use a construction which is pretty much the same as the one for simulating Muller automata by Büchi automata. Such a simulation introduces non-determinism: there is no straightforward way to directly simulate a deterministic Muller automaton by a deterministic Rabin automaton even though deterministic Rabin automata *do* recognize all ω-regular languages, as we shall see in the next section.

The last acceptance condition we look at is obtained by interpreting the pairs table of a Rabin automaton in a complementary fashion.

Streett automata *A* Streett automaton *is a structure* $(\mathcal{A}, \mathcal{PT})$ *where* $\mathcal{A} = (S, \rightarrow, S_{in})$ *and* $\mathcal{PT} = \langle(G_1, R_1), (G_2, R_2), \ldots, (G_k, R_k)\rangle$ *are defined in the same way as for Rabin automata.*

The Streett automaton $(\mathcal{A}, \mathcal{PT})$ *accepts an input* $\alpha : \mathbb{N}_0 \rightarrow \Sigma$ *if there is a run* ρ *of* $\mathcal{A}$ *on* α *such that for every* $i \in \{1, 2, \ldots, k\}$, *if* $\mathsf{inf}(\rho) \cap G_i \neq \emptyset$ *it is also the case that* $\mathsf{inf}(\rho) \cap R_i \neq \emptyset$.

These automata were defined by Streett in [St88]. They are useful for describing *fairness conditions* in infinite computations—for instance, conditions of the form "if a request for a resource is made infinitely often, then the system grants access to the resource infinitely often". The following observation is immediate from the close connection between Rabin and Streett automata.

Proposition 3.3 *Let $(\mathcal{A}, \mathcal{PT})$ be a deterministic automaton with a pairs table. Let L_R be the language accepted by $(\mathcal{A}, \mathcal{PT})$ when $\mathcal{PT}$ is interpreted as a Rabin condition and L_S be the language accepted by $(\mathcal{A}, \mathcal{PT})$ when $\mathcal{PT}$ is interpreted as a Streett condition. Then L_S is the complement of L_R.*

As usual, simulating a Büchi automaton $(\mathcal{A}, G)$ by a Streett automaton is easy. Let $\mathcal{A} = (S, \rightarrow, S_{in})$. Construct an automaton $(\mathcal{A}, \mathcal{PT}_G)$ where $\mathcal{PT}_G = \langle (S, G) \rangle$. Since $\mathsf{inf}(\rho) \cap S$ must be non-empty for any run ρ of $\mathcal{A}$, it follows that a run ρ satisfies the pair (S, G) iff $\mathsf{inf}(\rho) \cap G \neq \emptyset$, which is precisely what the Büchi condition demands.

In the converse direction, Safra describes a construction due to Vardi which shows that Streett automata can be efficiently simulated by Büchi automata [Sa88].

Lemma 3.4 *Let $(\mathcal{A}, \mathcal{PT})$ be a Street automaton where $\mathcal{A} = (S, \rightarrow, S_{in})$. Let $n = |S|$ and let k be the number of pairs in $\mathcal{PT}$: that is, $\mathcal{PT} = \langle (G_1, R_1), (G_2, R_2), \ldots, (G_k, R_k) \rangle$. Then, we can construct a Büchi automaton $(\mathcal{A}', G')$ with $\mathcal{A}' = (S', \rightarrow', S'_{in})$ such that $L(\mathcal{A}', G') = L(\mathcal{A}, \mathcal{PT})$ and $|S'| = n \cdot 2^{O(k)}$.*

Proof The automaton $\mathcal{A}'$ simulates $\mathcal{A}$. As usual, $\mathcal{A}'$ guesses an initial prefix of the run after which every state which is visited by the run will in fact be visited infinitely often. After making this guess, $\mathcal{A}$ checks that the acceptance criterion is met for each pair $(G_i, R_i) \in \mathcal{PT}$. In other words, for every i such that some state from G_i appears in the infinite portion of the run, $\mathcal{A}'$ ensures that some state from R_i also appears infinitely often. To do this, $\mathcal{A}'$ maintains two sets as part of its state. The first set accumulates the list of indices corresponding to pairs (G_i, R_i) where some element of G_i occurs infinitely often. The second set repeatedly accumulates indices of pairs (G_i, R_i) for which some element of R_i has been visited. Each time the second set becomes as large as the first, it is reset to empty. It is not difficult to see that the acceptance criterion specified by $\mathcal{PT}$ is met iff the second set is reset to empty infinitely often during the simulation.

Formally, we construct $(\mathcal{A}', G')$ as follows:

- $S' = \{(s, \mathsf{finite}) \mid s \in S\} \cup \{(s, X_1, X_2) \mid s \in S \text{ and } X_1, X_2 \subseteq \{1, 2, \ldots, k\}\}$.
- The transition relation $\rightarrow'$ is defined as follows:

$$(s, \mathsf{finite}) \xrightarrow{a}{}' (s', \mathsf{finite}) \text{ if } s \xrightarrow{a} s'.$$

$$(s, \mathsf{finite}) \xrightarrow{a}{}' (s', \emptyset, \emptyset) \text{ if } s \xrightarrow{a} s'.$$

$$\begin{aligned}(s, X, Y) \xrightarrow{a}{}' (s', X \cup G_{s'}, Y \cup R_{s'}) &\text{ if } s \xrightarrow{a} s' \text{ and } X \cup G_{s'} \not\subseteq Y \cup R_{s'},\\ \text{where } G_{s'} &= \{i \in \{1, 2, \ldots, k\} \mid s' \in G_i\} \text{ and}\\ R_{s'} &= \{i \in \{1, 2, \ldots, k\} \mid s' \in R_i\}.\end{aligned}$$

$$(s, X, Y) \xrightarrow{a}{}' (s', X \cup G_{s'}, \emptyset) \text{ if } s \xrightarrow{a} s' \text{ and } X \cup G_{s'} \subseteq Y \cup R_{s'}.$$

- $S^i_{in} = \{(s, \mathsf{finite}) \mid s \in S_{in}\}$.
- $G_i = \{(s, X, \emptyset) \mid s \in S, X \subseteq \{1, 2, \ldots, k\}\}$.

$\square$

4 Determinizing Büchi automata

We now describe an elegant construction due to Safra for determinizing Büchi automata [Sa88]. Safra's construction converts a non-deterministic Büchi automaton $(\mathcal{A}, G)$ into a deterministic Rabin automaton $(\mathcal{A}_G, \mathcal{PT}_G)$ such that $L(\mathcal{A}_G, \mathcal{PT}_G) = L(\mathcal{A}, G)$. If we regard $(\mathcal{A}_G, \mathcal{PT}_G)$ as a Streett automaton, we get a deterministic automaton recognizing the complement of $L(\mathcal{A}, G)$. By Lemma 3.4, we can simulate the Streett automaton $(\mathcal{A}_G, \mathcal{PT}_G)$ by a Büchi automaton. Thus Safra's construction also solves the complementation problem for Büchi automata.

Also, recall that it is easy to convert a deterministic Rabin automaton into a deterministic Muller automaton. As a consequence, Safra's construction gives an indirect proof of McNaughton's Theorem (Theorem 3.2).

Safra's determinization construction for Büchi automata is a clever extension to the infinite word case of the classical subset construction for determinizing automata on finite words. In order to motivate the construction, we begin with the subset construction for finite words and enhance it in a graded manner to achieve the final result.

Subset construction For automata on finite words, the subset construction is the standard way to eliminate non-determinism. If the original automaton is $(\mathcal{A}, F)$, with $\mathcal{A} = (S, \rightarrow, S_{in})$, each state of the *subset automaton* $(S_{sub}, \rightarrow_{sub}, S^{sub}_{in})$ is a subset of S. The (single) initial state of the subset automaton is the set S_{in} of initial states of $\mathcal{A}$. The subset automaton's transition relation $\rightarrow_{sub}$ is defined as follows:

$$X \xrightarrow{a}_{sub} Y \text{ iff } Y = \{y \in S \mid \exists x \in X : x \xrightarrow{a} y\}.$$

Henceforth, we use $\delta_{sub}(X, a)$ to denote the set Y such that $X \xrightarrow{a}_{sub} Y$.

The subset automaton satisfies the following property: If $X \xrightarrow{w}^{+}_{sub} Y$ then for each state y in Y, there is an state $x \in X$ such that $x \xrightarrow{w}^{+} y$ in the original automaton.

From this, it follows that if we set the set of final states of the subset automaton to be $F_{sub} = \{X \subseteq S \mid S \cap F \neq \emptyset\}$, then $(\mathcal{A}_{sub}, F_{sub})$ recognizes the same set of words as the original automaton.

Let $(\mathcal{A}, G)$ be a non-deterministic Büchi automaton, with $\mathcal{A} = (S, \rightarrow, S_{in})$. The natural extension of the subset construction to Büchi automata would set the good states of the subset automaton to $G_{sub} = \{X \subseteq S \mid X \cap G \neq \emptyset\}$.

It is easy to see that if $(\mathcal{A}, G)$ accepts an input α, so will $(\mathcal{A}_{sub}, G_{sub})$. Unfortunately, the converse is not true—the subset automaton will accept words which are not part of the original language.

Example 4.1 Consider the automaton of Example 1.1 (Figure 3) which recognizes $\overline{L}$ over $\{a, b\}$ given by $\overline{L} = \{\alpha \mid \alpha \text{ has only a finite number of occurrences of } a\}$.

In this example, G_{sub}, the set of good states of the extended subset automaton, is given by $\{\{s_1, s_2\}, \{s_2\}\}$. On the input $(ab)^\omega = ababab\cdots$, the (unique) run of the subset

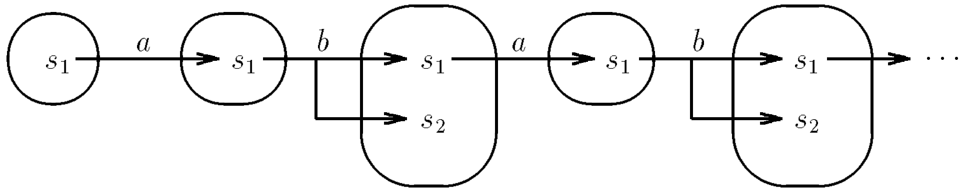

Figure 7: A run of the extended subset automaton for $\overline{L}$ on input $(ab)^\omega$ (Example 4.1)

automaton is $\{s_1\}(\{s_1\}\{s_1,s_2\})^\omega = \{s_1\}\{s_1\}\{s_1,s_2\}\{s_1\}\{s_1,s_2\}\{s_1\}\{s_1,s_2\}\cdots$. Since this run visits G_{sub} infinitely often, the automaton $(\mathcal{A}_{sub}, G_{sub})$ accepts the word $(ab)^\omega$, even though a occurs infinitely often in this word.

The problem with the subset construction is best brought out by drawing all the "threads" between the individual subsets in this run of the subset automaton—see Figure 7.

As we can see, every finite run of the original automaton which reaches a good state actually dies out at that point. In general, all that this subset construction guarantees is that the original automaton has arbitrarily long *finite* runs which visit good states. □

Marked subset construction We next try to strengthen the subset construction so that it explicitly keeps track of the threads between subsets. In the marked subset construction, in addition to keeping a subset of states, the subset automaton also has the ability to "mark" each state in the subset. A state in the current subset is marked if it satisfies one of two conditions: either it is a good state, or it has a marked predecessor in the previous subset. However, if all the states in the previous subset are marked, then only good states are marked in the current subset—no marks are inherited from a fully marked state. The good states in the marked subset automaton are those where the entire subset is marked.

Concretely, let $(\mathcal{A}, G)$ be the original non-deterministic Büchi automaton with $\mathcal{A} = (S, \rightarrow, S_{in})$. Then, the marked subset automaton $(\mathcal{A}_M, G_M)$, with $\mathcal{A}_M = (S_M, \rightarrow_M, S_{in}^M)$, is given as follows.

- $S_M = \{(X, f) \mid X \subseteq S, f : X \rightarrow \{\mathsf{marked}, \mathsf{unmarked}\}\}$.
- The transition function $\rightarrow_M$ is as follows:
  $(X, f) \xrightarrow{a}_M (Y, g)$ iff
  - $Y = \delta_{sub}(X, a)$.
    (Recall that $Y = \delta_{sub}(X, a)$ iff in the normal subset automaton, $X \xrightarrow{a}_{sub} Y$.)
  - If $f(x) = \mathsf{marked}$ for all $x \in X$
    then
    $$\forall y \in Y : g(y) = \begin{cases} \mathsf{marked} & \text{if } y \in G \\ \mathsf{unmarked} & \text{otherwise} \end{cases}$$
    else
    $$\forall y \in Y : g(y) = \begin{cases} \mathsf{marked} & \text{if } y \in G \text{ or} \\ & (\exists x \in X : f(x) = \mathsf{marked} \text{ and } x \xrightarrow{a} y) \\ \mathsf{unmarked} & \text{otherwise} \end{cases}$$

- $S_{in}^M = \{(S_{in}, f) \mid \forall s \in S_{in} : f(s) = \mathsf{marked}\}$.
- $G_M = \{(X, f) \mid \forall x \in X : f(x) = \mathsf{marked}\}$.

The main property satisfied by this automaton is the following.

> *Let ρ be a run of $(\mathcal{A}_M, G_M)$ on an input α such that $\rho(i) = (X_i, f_i)$ for all $i \in Nat$. Suppose that $(X_k, f_k) \in G$ for some $k > 0$. Let j be the largest natural number less than k such that $(X_j, f_j) \in G$—such a number j must exist because the initial state of $\mathcal{A}_M$ belongs to G.*
>
> *Then, for each state $y \in Y_k$, there is a state $x \in X_j$ such that $x \xrightarrow{\alpha[j..k-1]}{}^{*}\ y$ in the original automaton and, moreover, when going from x to y on reading $\alpha[j..k{-}1]$, the original automaton goes through some good state.*

From this observation, we can deduce that the marked subset construction is sound.

Proposition 4.2 *Let $(\mathcal{A}_M, G_M)$ be the marked subset automaton which corresponds to the Büchi automaton $(\mathcal{A}, G)$. Then, if $(\mathcal{A}_M, G_M)$ accepts an input α, so does $(\mathcal{A}, G)$.*

Proof Let ρ_α be the (unique) run of $\mathcal{A}_M$ on an input α with $\rho_\alpha(i) = (X_i, f_i)$ for $i \in \mathbb{N}_0$. If $(\mathcal{A}_M, G_M)$ accepts α, there must be an infinite sequence of positions $\{i_0, i_1, \ldots\} \subseteq \mathbb{N}_0$ such that $0 = i_0 < i_1 < \cdots$ and $(X_j, f_j) \in G$ for all $j \in \{i_0, i_1, \ldots\}$.

From our previous observation about the marked subset construction, we know that for each index i_{k+1} in the set $\{i_0, i_1, \ldots\}$ and for each state $x \in X_{i_{k+1}}$, there is a state $y \in X_{i_k}$ such that in the original automaton, there is a sequence of transitions leading from y to x on the input $\alpha[i_k..i_{k+1}{-}1]$ which passes through some good state. Let us call such a state y a *good predecessor* of x.

We construct an infinite tree T_α as follows. The root of the tree is the set S_{in} of initial states. At each level k of the tree, $k \geq 1$, we have a node $n_{(x,i_k)}$ corresponding to each state $x \in X_{i_k}$. The parent of a node $n_{(x,j)}$ at level j, $j > 1$, is a node $n_{(y,j-1)}$ at level $j{-}1$ such y is a good predecessor of x. (Of course, x may have more than one good predecessor. If this is the case, we arbitrarily select one of them and make the corresponding node the parent of $n_{(x,j)}$ in the tree.)

The tree T_α is finitely branching and has an infinite number of nodes. By König's lemma, it must have an infinite path. Each infinite path in T_α corresponds to a run of the original automaton $\mathcal{A}$ on α. By construction, such a run must pass through a good state between each level in the tree. Thus, $\mathcal{A}$ has at least one run on α which meets G infinitely often. $\square$

Unfortunately, though the marked subset construction is sound, it is not complete—there may be inputs accepted by $(\mathcal{A}, G)$ which are not accepted by $(\mathcal{A}_M, G_M)$. Consider the following example.

Example 4.3

In the Büchi automaton shown in Figure 8, the input a^ω generates the sequence of subsets $\{s_0\}(\{s_1, s_2\})^\omega$. Since s_2 is not a final state, the subset $\{s_1, s_2\}$ never becomes fully marked. Thus, though the original automaton has an accepting run $s_0 s_1^\omega$ on this input, the marked subset construction fails to detect this. $\square$

The problem is that the marked subset construction demands too much from the underlying runs. As the example shows, it should be sufficient to identify a portion of the subset which is marked and which can infinitely often regenerate its marks.

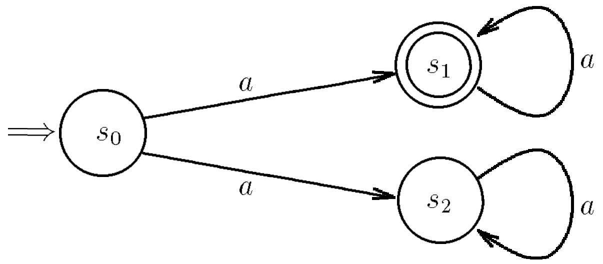

Figure 8: The marked subset construction is not complete (Example 4.3)

Hierarchical Marked Subset Construction A first attempt to weaken the marked subset construction would be to have a hierarchy of marks. At the base level, the subset automaton runs the marked subset construction and marks states using a level 1 mark. The states which have level 1 marks then start off a nested copy of the marked subset construction with level 2 marks. Similarly, the states which have level 2 marks start off a marked subset construction with level 3 marks. What we would like to detect is whether some level i can get completely marked. This corresponds to checking if the set of nodes marked at level i is equal to the set of nodes marked at level $i{+}1$. If so, we reset all marks at levels greater than i and continue.

Since the number of nodes marked at level i is always strictly greater than the set of nodes marked at level $i{+}1$, there can be at most as many levels as there are states in the original automaton.

To specify the acceptance condition, we need to verify that some level $i{+}1$ gets set to empty infinitely often *and* that level i does not get set to empty in between. In other words, level i denotes a permanent thread through the subset construction which gets marked infinitely often. To do this, we have to pass from a Büchi condition to a Rabin condition—for each i, a positive condition for level $i{+}1$ has to be qualified by a negative condition for level i.

Here is a formal description of a hierarchical marked subset construction which attempts to achieve this goal. Let $(\mathcal{A}, G)$ be a Büchi automaton, with $\mathcal{A} = (S, \rightarrow, S_{in})$. Define $(\mathcal{A}_H, \mathcal{PT}_H)$, with $\mathcal{A}_H = (S_H, \rightarrow_H, S^H_{in})$, as follows:

- Let $|S| = n$. S_H consists of pairs of the form (σ, χ) where:
  - $\sigma : \{1, 2, \ldots, n\} \rightarrow 2^S$ is a *subset list* satisfying the condition that $\sigma(i{+}1)$ is a *proper* subset of $\sigma(i)$ whenever $\sigma(i)$ is non-empty.
  - $\chi : \{1, 2, \ldots, n\} \rightarrow \{\mathsf{white}, \mathsf{green}\}$ is a *colour list.*
- $S^H_{in} = (\sigma_0, \chi_0)$, where $\sigma_0(1) = S_{in}$, $\sigma_0(i) = \emptyset$ for all $i \in \{2, 3, \ldots, n\}$ and $\chi(i) = \mathsf{white}$ for all $i \in \{1, 2, \ldots, n\}$.
- The transition function $\rightarrow_H$ performs the following sequence of actions. Initially, each level runs the subset construction locally. Next, any final states appearing in the new subset at level i are added to the subset at level $i{+}1$—this corresponds to generating fresh marks at level i. We now look for the smallest level i whose subset

is the same as that at level $i{+}1$. If such an i exists, we "clear out" the subset list from level $i{+}1$ onwards and set the colour of level i to green.

More formally, on reading an input a, the state (σ, χ) generates a new state (σ', χ') as follows:

(i) Let $\sigma_1 : \{1, 2, \ldots, n+1\} \to 2^S$ be defined as follows:

- $\sigma_1(1) = \delta_{sub}(\sigma(1), a)$.
- For $i \in \{2, 3, \ldots, n\}$, $\sigma_1(i) = \delta_{sub}(\sigma(i), a) \cup (\delta_{sub}(\sigma(i{-}1), a) \cap G)$.
- $\sigma_1(n{+}1) = \delta_{sub}(\sigma(n), a) \cap G$.

(ii) If there is no index $i \in \{1, 2, \ldots, n\}$ such that $\sigma_1(i) = \sigma_1(i{+}1)$, then

- $\sigma'(i) = \sigma_1(i)$ for all $i \in \{1, 2, \ldots, n\}$.
- $\chi'(i) = \mathsf{white}$ for all $i \in \{1, 2, \ldots, n\}$.

else, let m be the smallest index such that $\sigma_1(m) = \sigma_1(m{+}1)$. Then,

- $\sigma'(i) = \sigma_1(i)$ for all $i \in \{1, 2, \ldots, m\}$ and $\sigma'(i) = \emptyset$ for all $i > m$.
- $\chi'(m) = \mathsf{green}$ and $\chi'(i) = \mathsf{white}$ for all $i \neq m$.

- The acceptance table $\mathcal{PT}_H$ consists of n pairs $\langle (G_1, R_1), (G_2, R_2), \ldots, (G_n, R_n) \rangle$ where:
  - $R_i = \{(\sigma, \chi) \mid \sigma(i) = \emptyset\}$
  - $G_i = \{(\sigma, \chi) \mid \chi(i) = \mathsf{green}\}$

In this construction, the list σ implicitly records the levels of marks associated with the states in the current subset—a state s belongs to $\sigma(i{+}1)$ iff s has a level i mark in the current subset. It is not difficult to show that this construction is complete.

Proposition 4.4 *Let $(\mathcal{A}_H, \mathcal{PT}_H)$ be the hierarchical marked subset automaton which corresponds to the Büchi automaton $(\mathcal{A}, G)$. If $(\mathcal{A}, G)$ accepts an input α, so does $(\mathcal{A}_H, \mathcal{PT}_H)$.*

Proof Suppose $(\mathcal{A}, G)$ accepts α. Then, α admits a run ρ which visits G infinitely often. We must show that the unique run ρ_α of $\mathcal{A}_H$ on α satisfies some entry in $\mathcal{PT}_H$.

For $j \in \mathbb{N}_0$, let $\rho_\alpha(j) = (\sigma_j, \chi_j)$. We know that for all $j \in \mathbb{N}_0$, $\sigma_j(1)$ is the set of states maintained by the subset construction. Since ρ is a valid run of $\mathcal{A}$ on α, $\sigma_j(1)$ is always non-empty. So, ρ_α satisfies condition R_1. If it also satisfies G_1 then $(\mathcal{A}_H, \mathcal{PT}_H)$ accepts α and we are done.

If ρ_α does not satisfy G_1, let k_0 be the last position where the colour of the first level is green. We wait for the first position $i_0 > k_0$ where ρ, the accepting run of $\mathcal{A}$ on α, next visits a good state. We know that $\sigma_j(2)$ is non-empty for all $j \geq i_0$—once the good state seen at position i_0 gets pushed to level 2, the accepting run ρ will be part of the subset construction maintained at level 2, thus guaranteeing that at least one valid state is generated at each point. So, ρ_α satisfies R_2. If it also satisfies G_2 we are done.

Otherwise, we repeat the argument above and deduce that ρ_α satisfies R_3, with the accepting run ρ a part of $\sigma_j(3)$ for all j greater than a finite index i_1. We can repeat this

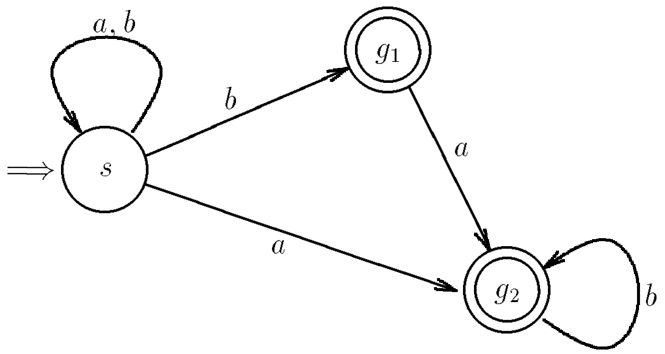

Figure 9: The hierarchical marked subset construction is not sound (Example 4.5)

argument only a finite number of times, till we reach level n. The subset maintained at level n can never be more than a singleton. If the accepting run ρ is part of the subset construction at level n, it must generate the signal green infinitely often in which case ρ_α satisfies the pair (G_n, R_n). □

Unfortunately, the hierarchical subset construction is *not* sound. Consider this example.

Example 4.5

The automaton shown in Figure 9 does not accept the input $(ba)^\omega$. However, the run of the hierarchical marked subset automaton on this input is the following:

$$\begin{array}{|c|c|}\hline 1 & (\{s\}, \mathsf{white}) \\ \hline 2 & (\emptyset, \mathsf{white}) \\ \hline 3 & (\emptyset, \mathsf{white}) \\ \hline \end{array} \xrightarrow{b}_H \begin{array}{|c|c|}\hline 1 & (\{s, g_1\}, \mathsf{white}) \\ \hline 2 & (\{g_1\}, \mathsf{white}) \\ \hline 3 & (\emptyset, \mathsf{white}) \\ \hline \end{array} \xrightarrow{a}_H \begin{array}{|c|c|}\hline 1 & (\{s, g_2\}, \mathsf{white}) \\ \hline 2 & (\{g_2\}, \mathsf{green}) \\ \hline 3 & (\emptyset, \mathsf{white}) \\ \hline \end{array} \xrightarrow{b}_H$$

$$\begin{array}{|c|c|}\hline 1 & (\{s, g_1, g_2\}, \mathsf{white}) \\ \hline 2 & (\{g_1, g_2\}, \mathsf{white}) \\ \hline 3 & (\{g_2\}, \mathsf{white}) \\ \hline \end{array} \xrightarrow{a}_H \begin{array}{|c|c|}\hline 1 & (\{s, g_2\}, \mathsf{white}) \\ \hline 2 & (\{g_2\}, \mathsf{green}) \\ \hline 3 & (\emptyset, \mathsf{white}) \\ \hline \end{array} \xrightarrow{b}_H \cdots$$

Since level 2 remains populated forever and turns green infinitely often, the hierarchical construction incorrectly accepts this input. □

In the preceding example, the problem is that the good state g_2 which appears to be permanently part of level 2 is actually a transient state. Each time an a is read, the g_2 state at level 2 disappears, only to be replaced by a fresh copy of g_2 which is pushed from level 1.

To rectify this defect, Safra's construction maintains each level of marks as a disjoint set of nodes. The hierarchy of subsets then becomes a tree, with the set at level 1 as the root.

Safra's Construction Before presenting Safra's construction, we review some terminology regarding trees. A tree is a structure $T = (V, v_r, \pi)$ where V is a set of nodes, $v_r \in T$ is a special node known as the *root* and for all $v \in V - \{v_r\}$, $\pi(v) \in T$ fixes the *parent* of the node. If $v = \pi^i(v')$ for some $i > 0$, we say that v is an *ancestor* of v'. The root v_r is an ancestor of every other node. If $v' = \pi(v)$ then v is said to be a *child* of v'. We assume that for any node v, all the children of v are ordered so that we can talk of one child being to the *left* of another. This generates a total order on nodes—if v and v' are nodes, we say that $v < v'$ if v is an ancestor of v' or if there is a common ancestor u of v and v' such that v is in the subtree rooted at a child u_1 of v, v' is in the subtree rooted at a child u_2 of v and u_1 is to the left of u_2.

Given a Büchi automaton $(\mathcal{A}, G)$, with $\mathcal{A} = (S, \rightarrow, S_{in})$, Safra's construction produces a Rabin automaton $(\mathcal{A}_G, \mathcal{PT}_G)$, with $\mathcal{A}_G = (S_G, \rightarrow_G, S_{in}^G)$. The automaton $(\mathcal{A}_G, \mathcal{PT}_G)$ is as follows:

- Each state in S_G is a structure $(T, \sigma, \chi, \lambda)$ where
  - $T = (V, v_r, \pi)$ is a tree.
  - $\sigma : V \rightarrow 2^S$ associates a set of states of $\mathcal{A}$ with each node in V in such a way that:
    - The union of the sets associated with the children of a node v is a *proper* subset of $\sigma(v)$.
    - If v and v' are two nodes such that v is not an ancestor of v' and v' is not an ancestor of v then $\sigma(v)$ is disjoint from $\sigma(v')$.
    - If $\sigma(v) = \emptyset$, then v is the root v_r.

    It is not difficult to verify that the conditions imposed on the function σ ensure that $|V|$ can be no larger than n, where n is the number of states in S.
  - $\chi : V \rightarrow \{\mathsf{white}, \mathsf{green}\}$ fixes a colour for each node.
  - $\lambda : V \rightarrow \mathcal{L}$ is an injective function which attaches a *label* from the set $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_{2n}\}$ to each node. Notice that $\mathcal{L}$ has $2n$ elements.

- On reading an input a, the state $(T, \sigma, \chi, \lambda)$ is transformed to the state $(T', \sigma', \chi', \lambda')$ as follows:

  (i) Let $T = (V, v_r, \pi)$. Expand the T to a tree $T_1 = (V_1, v_r, \pi_1)$ as follows: For each $v \in V$, if $\sigma(v) \cap G \neq \emptyset$, add a node v' such that $\pi_1(v') = v$ and v' is the right-most child of v.

  (ii) Extend σ and λ to functions σ_1 and λ_1 over T_1 as follows:
  For all nodes v in $V_1 \cap V$, let $\sigma_1(v) = \sigma(v)$. For a new node $v \in V_1 - V$, $\sigma_1(v) = \sigma(\pi_1(v)) \cap G$.
  All nodes v in $V_1 \cap V$ inherit the label $\lambda(v)$. For each node in $V_1 - V$, choose a new label from $\mathcal{L}$ which is not assigned to any other node. Since there $2n$ labels to choose from, this is always possible—each node in V generates at most one new child in V_1 and there were not more than n nodes in V.

  (iii) For every node v, apply the subset construction locally. In other words, define a new function $\sigma_1' : V_1 \rightarrow 2^S$ such that $\sigma_1'(v) = \delta_{sub}(\sigma_1(v), a)$ for all $v \in V_1$.

  At this stage, we have to "clean up" T_1 and σ_1' so that the structure once again satisfies the conditions specified for states of $\mathcal{A}_G$.

(iv) For every node $v \in V_1$, if $s \in \sigma'_1(v)$ and s also belongs to $\sigma'_1(v')$ for some smaller node v', $v' < v$, (recall the total order on all nodes in a tree) remove s from $\sigma'_1(v)$.

(v) Remove all nodes v such that $\sigma'_1(v) = \emptyset$ and v is not the root v_r.

(vi) For each node v such that $\sigma'_1(v)$ is equal to $\bigcup\{\sigma'_1(v') \mid v = \pi_1(v')\}$, remove all the children of v and set $\chi_1(v) = \mathsf{green}$. For all other nodes, set $\chi_1(v) = \mathsf{white}$.

(vii) Let the set of nodes remaining be V'. For $v \in V'$, $\sigma'(v)$ is that part of $\sigma'_1(v)$ which remains after discarding states which already appear to the left, as specified in step (iv) above. The label $\lambda'(v)$ of a node v is retained from T_1. Finally, set $\chi' = \chi_1$.

- The initial state of $\mathcal{A}_G$ is the tree $(\{v_r\}, v_r, \emptyset)$ where $\sigma(v_r) = S_{in}$, $\chi(v_r) = \mathsf{white}$ and $\lambda(v_r) = \ell_1$.
- The pairs table $\mathcal{PT}_G = \langle (G_1, R_1), (G_2, R_2), \ldots, (G_{2n}, R_{2n}) \rangle$ is defined as follows:
  - $R_i = \{(T = (V, v_r, \pi), \sigma, \chi, \lambda) \mid \forall v \in V : \lambda(v) \neq \ell_i\}$.
  - $G_i = \{(T = (V, v_r, \pi), \sigma, \chi, \lambda) \mid \exists v \in V : \lambda(v) = \ell_i \text{ and } \chi(v) = \mathsf{green}\}$.

The labelling procedure guarantees that the labels of new nodes added at each stage are disjoint from the labels of the existing nodes. In other words, if a node labelled ℓ_i is deleted from the tree during a transition, the label ℓ_i temporarily disappears from the tree.

Thus, an entry (G_i, R_i) in the pairs table specifies the following condition. The condition R_i is satisfied if at some stage a node labelled ℓ_i is added to the tree and it is never deleted henceforth. The condition G_i then says that this node turns green infinitely often.

Figure 10 describes the run generated by Safra's construction on the input $(ba)^\omega$ for the automaton shown in Figure 9. In the figure, each node of a tree is denoted by a circle, with the label indicated inside the circle and the associated subset written by its side. Nodes coloured green are drawn as double circles, while white nodes are drawn as single circles. When selecting labels for new nodes added to the tree at each stage, we have followed the policy of using the first "free" label in $\mathcal{L}$.

At the second level of the tree, nodes labelled ℓ_2 and ℓ_3 turn green infinitely often, so the run satisfies G_2 and G_3. However, since both these labels also disappear from the tree infinitely often, the run does not satisfy R_2 or R_3, thus ensuring that the automaton rejects this input.

It is not difficult to see that Safra's construction satisfies a property similar to the only described for the marked subset construction:

Let ρ be a run of $(\mathcal{A}_G, \mathcal{PT}_G)$ on an input α such that $\rho(i) = (T_i, \sigma_i, \chi_i, \lambda_i)$ for all $i \in \mathbb{N}_0$. Let $j, k \in \mathbb{N}_0$ with $j < k$ and ℓ be a label from $\mathcal{L}$ such that:

- *For all positions $i \in \{j, j+1, \ldots, k\}$, there is a node v in the tree T_i such that $\lambda_i(v) = \ell$.*
- *$\chi_j(v) = \chi_k(v) = \mathsf{green}$ and for all i such that $j < i < k$, $\chi_i(v) = \mathsf{white}$.*

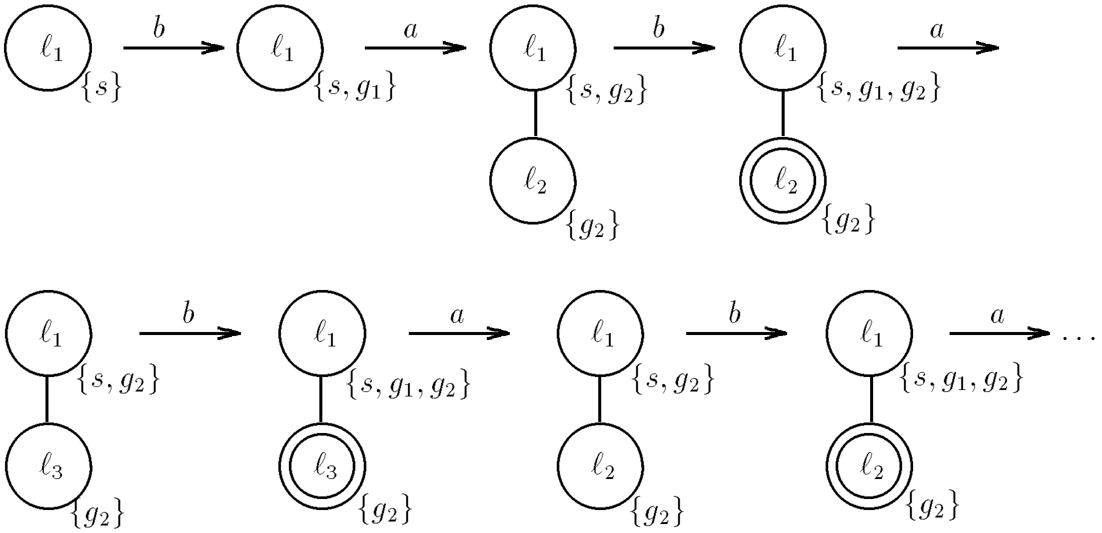

Figure 10: A run generated by Safra's construction

Then, for each state $y \in \sigma_k(v)$, there is a state $x \in \sigma_j(v)$ such that $x \xrightarrow{\alpha[j..k-1]}{}^* \; y$ in the original automaton and, moreover, when going from x to y on reading $\alpha[j..k-1]$, the original automaton goes through some good state.

Once we have this property, the soundness of Safra's construction follows from an argument very similar to the one described for the marked subset construction in Proposition 4.2. In other words, we can show the following.

Proposition 4.6 *Let $(\mathcal{A}_G, \mathcal{PT}_G)$ be the Rabin automaton generated by Safra's construction, corresponding to the Büchi automaton $(\mathcal{A}, G)$. Then, if $(\mathcal{A}_G, \mathcal{PT}_G)$ accepts an input α, so does $(\mathcal{A}, G)$.*

Proof As in the proof of Proposition 4.2, we construct a finitely branching tree T_α with an infinite set of nodes for each input α and argue that each infinite path in T_α corresponds to an accepting run of $\mathcal{A}$ on α. We omit the details. $\square$

The completeness of Safra's construction is shown by an argument similar to the one described for the hierarchical marked subset construction.

Proposition 4.7 *Let $(\mathcal{A}_G, \mathcal{PT}_G)$ be the Rabin automaton generated by Safra's construction corresponding to the Büchi automaton $(\mathcal{A}, G)$. If $(\mathcal{A}, G)$ accepts an input α, so does $(\mathcal{A}_G, \mathcal{PT}_G)$.*

Proof Suppose $(\mathcal{A}, G)$ accepts α. Then, α admits a run ρ which visits G infinitely often. We must show that the unique run ρ_α of $\mathcal{A}_G$ on α satisfies some entry in $\mathcal{PT}_G$.

For $j \in \mathbb{N}_0$, let $\rho_\alpha(j) = (T_j, \sigma_j, \chi_j, \lambda_j)$. Initially, the root v_r is assigned the label ℓ_1. Since the root is never removed, the run ρ_α satisfies R_1. If it also satisfies G_1 then $(\mathcal{A}_G, \mathcal{PT}_G)$ accepts α and we are done.

If ρ_α does not satisfy G_1, let k_1 be the last position at which the root is coloured green. Let i_1 be the first position after k where ρ, the accepting run of $\mathcal{A}$ on α, visits G. At this point, a child v_1 of the root comes into existence.

We know that the accepting run ρ is part of the overall subset construction maintained by the root node. Once v_1 is created, we know that ρ is also being maintained at the first level. It would appear that the run is maintained by v_1 itself, but there is a subtle complication to be taken into account. Since we only retain the left-most copy of each state, the run ρ may be passed on by v_1 to some sibling on the left. In any case, it can only move left a finite number of times. Let us suppose it eventually settles down at some node v_1'.

It is not difficult to verify that the node v_1' must already have been in the tree when v_1 was added. Let $\ell_{i_1} = \lambda_{i_1}(v_1')$ be the label of v_1'. Since α never dies out, v_1' will never be deleted from the tree. In other words, ρ_α satisfies the condition R_{i_1}. If it satisfies the corresponding condition G_{i_1} we are done.

Otherwise, let k_2 be the last time where v_1' turns green. As before, we wait for i_2, the next time ρ visits a good state, and look at the child v_2 of v_1' which is created at this point. The run ρ is copied into the subset maintained by v_2 and passed on left a finite number of times till it settles down at a node v_2'. If ℓ_{i_2} is the label of v_2', ρ_α must satisfy R_{i_2}. If ρ_α does not also satisfy G_{i_2} we push ρ down one more level.

Since there are only n levels in the tree, we cannot do this indefinitely. Thus, we must eventually find a node v_m' labelled ℓ_{i_m} such that ρ_α satisfies the pair (G_{i_m}, R_{i_m}). $\square$

The complexity of Safra's construction The automaton $\mathcal{A}_G$ has $2^{O(n \log n)}$ states, where n is the number of states in $\mathcal{A}$. To see this, we estimate the number of bits required to write down a typical state of $\mathcal{A}_G$. We have to specify the structure $(T, \sigma, \lambda, \chi)$.

Since T has at most n nodes, we can "name" the nodes $\{1, 2, \ldots, n\}$, with $v_r = 1$. The structure (V, v_r, π) can then be written down as a list of the form $\{\pi(i)\}_{i \in \{1,2,\ldots,n\}}$. Since $\pi(i) \in \{1, 2, \ldots, n\}$ requires $\log n$ bits to write down, T can be described using $n \log n$ bits. Similarly, λ and χ can be written as lists of length n with each entry made up of $\log n$ bits and 1 bit, respectively.

The only catch is with σ—if we naïvely represent the function $\sigma : V \to 2^S$ as a list of subsets, we will need n bits to represent each entry, resulting in n^2 bits overall. However, notice that if a state s belongs to $\sigma(v)$ and $\sigma(v')$ for two different nodes v and v' it must be the case that v is an ancestor of v' or that v' is an ancestor of v. Also, if $s \in \sigma(v)$, s must belong to $\sigma(v')$ for *every* ancestor v' of v, all the way upto the root. Thus, we can characterize the set of nodes where s appears in terms of the lowest node v_s such that $s \in \sigma(v_s)$: if $s \in \sigma(v_s)$ then $s \in \sigma(v')$ for any other node v' iff v' is an ancestor of v_s. In this way, σ can also be represented as a list of length n by matching each state s in S to its corresponding node v_s in V. Each entry in this list can be written down using $\log n$ bits.

Since we can characterize a state of $\mathcal{A}_G$ using $O(n \log n)$ bits, it follows that the number of distinct states is bounded by $2^{O(n \log n)}$.

The number of pairs in $\mathcal{PT}_G$ is $O(n)$—by construction, there is a pair (G_i, R_i) for each label $\ell_i \in \mathcal{L}$ and $\mathcal{L}$ contains exactly $2n$ elements.

By Lemma 3.4, we can simulate the Streett automaton $(\mathcal{A}_G, \mathcal{PT}_G)$ by a Büchi automaton with $2^{O(n \log n)}$ states. Thus, complementing Büchi automata using Safra's construction results in the state space blowing up from n to $2^{O(n \log n)}$. Recall that for automata on finite words, the number of states in the complement (via the subset construction) is $2^{O(n)}$. It has been shown that the bound achieved by Safra's construction is optimal [Mi88].

Why complement Büchi automata? We have seen that if we work with Muller, Rabin or Streett conditions, we can in fact accept all ω-regular languages using deterministic automata. So, why do we bother about complementing non-deterministic Büchi automata?

The reason is that the natural translation of logical questions into automata necessarily introduces non-determinism. For instance, when we constructed the automaton $(\mathcal{A}_\varphi, G_\varphi)$ corresponding to an S1S formula φ in Section 2, non-determinism was unavoidable in the inductive step for handling existential quantification. This non-determinism arises regardless of what type of acceptance condition we choose to work with. Surprisingly, determinizing Muller or Rabin automata directly is no easier than first converting them to Büchi automata and then applying Safra's construction [Sa88].

Arguably, for our purposes it should suffice to complement Büchi automata— determinization is a stronger construction which yields complementation as a corollary. In fact, Klarlund [Kl91] has shown that it *is* possible to directly complement non-deterministic Büchi automata without determinizing them and without sacrificing the optimal $2^{O(n \log n)}$ bound achieved by Safra's construction. However, there are applications where determinization is crucial—for instance, in the game-theoretic analysis of automata on infinite trees [GH82].

5 Discussion

Our main focus in this survey has been in describing how Büchi automata can be used to settle decision problems in logic. On the way, we have proved some simple results about ω-regular languages. There has also been a lot of work on the algebraic and topological aspects of ω-regular languages which we have not even touched on. A detailed introduction can be found in the survey [Th90].

As mentioned in the Introduction, Meyer showed in [Me75] that the decision procedure for S1S has a *non-elementary* complexity. A formula of length n may generate an automaton with $2^{2^{\cdot^{\cdot^{2}}}}$ states, where the tower of exponentials is of height n. In other words, the size of the automaton cannot be bounded by a function of constant exponential height. This result appears to make it impossible to use this elegant theory in a practical setting for verifying properties of programs.

However, for temporal logics, it turns out that there are direct ways to construct a Büchi automaton $(\mathcal{A}_\varphi, G_\varphi)$ recognizing L_φ for a temporal logic formula φ, such that the size of $\mathcal{A}_\varphi$ is exponential in the length of the formula [VW86]. As shown in [VW86], Büchi automata also provide a clean solution to the *model-checking problem* for finite state systems. The model checking problem is the following—given a finite-state program P and a temporal logic formula φ, do all the computations of P satisfy φ?

Rabin showed that Büchi's decidability result for S1S could be extended to the logic S2S, the monadic second order theory of the infinite binary tree [Ra69]. The logic S2S is very powerful—for instance, it is powerful enough to embed the logic SωS, the monadic second order theory of the infinite *countably branching* tree.

Rabin's results are proved by extending the techniques developed for automata on infinite words to automata operating on infinite trees. These extensions are highly non-trivial—especially the result that automata on infinite trees are closed under complementation. A number of attempts have been made to simplify Rabin's difficult proof. A novel and fruitful approach has been to analyze these automata in terms of infinite games [GH82].

Recently, there has been a lot of work on lifting the theory of Büchi automata and ω-regular languages to the setting of concurrent programs [GP92, EM93]. When dealing with concurrent programs, it is often advantageous to regard the runs of the system as partial orders rather than as sequences. Two actions in such a run are unordered if they occur independently. A sequential description of a concurrent program will generate a number of equivalent interleavings for each partially ordered run. To verify properties of a concurrent program in terms of such a sequential description, we have to check all these equivalent interleavings when, in principle, it should suffice to check one representative interleaving for each partially ordered computation. By directly working with infinite labelled partial orders rather than infinite sequences, we can avoid some of this duplication of effort. A challenging open problem is to extend the work of [GP92, EM93] to branching structures with concurrency, corresponding to the case of infinite trees for sequential systems.

Acknowledgments I thank Deepak D'Souza, P. Madhusudan, Samik Sengupta and Barbara Sprick for patiently listening to me when I lectured on this material. Their questions and observations in class have greatly influenced the manner in which I have organized the material in this tutorial. In particular, Example 4.5 is due to Madhu and Barbara.

References

[AHU74] A.V. Aho, J.E. Hopcroft and J.D. Ullman: *The Design and Analysis of Algorithms*, Addison-Wesley, Reading (1974).

[Bü60] J.R. Büchi: On a decision method in restricted second order arithmetic, *Z. Math. Logik Grundlag. Math*, **6** (1960) 66–92

[EM93] W. Ebinger and A. Muscholl: Logical Definability on Infinite Traces, *Proc. ICALP '93, LNCS* **700** (1993) 335–346.

[Em90] E.A. Emerson: Temporal and modal logic, in: J. van Leeuwen ed., *Handbook of Theoretical Computer Science: Volume B*, North-Holland, Amsterdam (1990) 995–1072.

[GP92] P. Gastin and A. Petit: Asynchronous Cellular Automata for Infinite Traces, *Proc. ICALP '92, LNCS* **623** (1992) 583–594.

[GH82] Y. Gurevich and L. Harrington: Trees, automata and games, *Proc 14th ACM STOC*, (1982) 60–65.

[Kl91] N. Klarlund: Progress measures for complementation of ω-automata with applications to temporal logic, *Proc. 32nd IEEE FOCS* (1991) 358–367.

[Mc66] R. McNaughton: Testing and generating infinite sequences by a finite automaton, *Inform. Control*, **9** (1966) 521–530.

[Me75] A. Meyer: Weak monadic second order theory of successor is not elementary recursive, *Proc. Logic Colloquium, Lecture Notes in Mathematics* **453** (1975) 132–154.

[Mi88] M. Michel: Complementation is more difficult with automata on infinite words, Manuscript (1988).

[Mu63] D.E. Muller: Infinite sequences an finite machines, *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical Design*, (1963) 3–16.

[Pn77] A. Pnueli: The temporal logic of programs, *Proc. 18th IEEE FOCS* (1977) 46–57.

[Ra69] M.O. Rabin: Decidability of second order theories and automata on infinite trees, *Trans. AMS*, **141**(1969) 1–37.

[Sa88] S. Safra: On the complexity of ω-automata, *Proc. 29th IEEE FOCS*, (1988) 319–327.

[SV89] S. Safra and M. Vardi: On ω-automata and temporal logic, *Proc. 21st ACM STOC*, (1989) 127–137.

[SVW87] A.P. Sistla, M. Vardi and P. Wolper: The complementation problem for ω-automata with applications to temporal logic, *Theoret. Comput. Sci.*, **49** (1987) 217–237.

[St88] R. Streett: Propositional dynamic logic of looping and converse is elementarily decidable, *Inform. Control*, **48** (1981) 261–283.

[Th90] W. Thomas: Automata on infinite objects, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume B*, North-Holland, Amsterdam (1990) 133–191.

[VW86] M. Vardi and P. Wolper: An automata theoretic approach to automatic program verification, *Proc. 1st IEEE LICS*, (1986) 332–345.

[VW94] M. Vardi and P. Wolper: Reasoning about infinite computations, *Inform. Comput.*, **115**(1) (1994) 1–37.