

CDM

Automata on Infinite Words

Klaus Sutner

Carnegie Mellon University

Fall 2009

Outline

- 1 Infinite Words
- 2 Deterministic Languages
- 3 Muller and Rabin Automata
- 4 Presburger Arithmetic
- 5 Determinizing Büchi Automata
- 6 Safra's Algorithm
- 7 Boolean Operations on Recognizable Languages

Where Are We?

We have seen three different characterizations of regular languages on words:

- acceptance by a finite state machine,
- description by a regular expression,
- description by an $\text{MSO}[<]$ formula.

There are conversion algorithms for all these different representations (but not all of these algorithms are efficient).

Infinite Words

So far, we have dealt only with ordinary, finite words.

As it turns out, it also makes sense to talk about **infinite words**. These come in two flavors: bi-infinite

$$\Sigma^{\infty} = \mathbb{Z} \rightarrow \Sigma$$

or one-way infinite

$$\Sigma^{\omega} = \mathbb{N} \rightarrow \Sigma$$

For example, infinite words appear naturally in the analysis of dynamical systems (symbolic dynamics). They also can be used to describe the properties of programs that never halt, such as operating systems, user interfaces. Protocols also naturally give rise to infinite descriptions.

Automata Recognizing Infinite Words

We will only deal with one-way infinite words here. The two-way infinite case is similar but significantly messier. Significantly.

How do we have to modify finite state machines to cope with infinite inputs?

- **Transition system:** stays the same.
- **Acceptance condition:** requires work.

What kind of acceptance condition might make sense? For finite words there is a natural, reasonable answer, but for infinite words things become a bit more complicated.

Note that no matter what condition we choose, there will be a problem with acceptance testing: we cannot simulate an infinite run of a transition system in finite time (unless the run is exceedingly simple).

ω -Languages

$$\Sigma^\omega = \mathbb{N} \rightarrow \Sigma$$

One-way infinite words are often called ω -words.

Subsets of Σ^ω are called ω -languages.

Given an automaton for infinite words its acceptance language is denoted by $\mathcal{L}^\omega(\mathcal{A})$ and so on.

Note that an ω -language may well be uncountable; there cannot be a good notation system for ω -words.

We will usually drop the ω whenever it is obvious from context.

Traces and Runs

As in the finite case, suppose we have a trace

$$\rho_0, a_1, \rho_1, a_2, \dots, \rho_{m-1}, a_m, \rho_m, \dots$$

with corresponding run

$$\pi = \rho_0, \rho_1, \dots, \rho_{m-1}, \rho_m, \dots$$

Ignoring acceptance testing, the most natural condition seems to be that we would like the run to touch the set of final states infinitely often.

More precisely, define the set of **recurrent** states

$$\text{Inf}(\pi) = \{ p \in Q \mid \exists i \rho_i = p \}$$

Büchi Automata

Definition

A **Büchi automaton** M is a transition system $\langle Q, \Sigma, \tau \rangle$ together with an acceptance condition $I \subseteq Q$ and $F \subseteq Q$.

M accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of M on x that starts at I and such that $\text{Inf}(\pi) \cap F \neq \emptyset$. The collection of all such words is the acceptance language of M .

A language $L \subseteq \Sigma^\omega$ is **recognizable** if there is some Büchi automaton that accepts it.

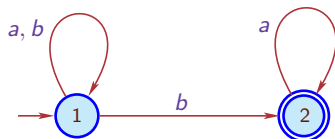
So far, this is just a definition. It seems reasonable, but is absolutely not clear at this point that this is the right way to approach languages on infinite words.

Example I

Consider alphabet $\Sigma = \{a, b\}$. We claim that

$$L = \{x \in \{a, b\}^\omega \mid 1 \leq \#_b x < \infty\}$$

of words containing at least one but only finitely many b 's is recognizable. A Büchi automaton for L looks like so:

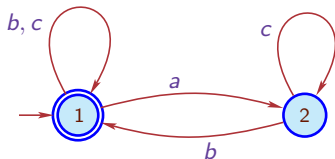


Example II

Consider alphabet $\Sigma = \{a, b, c\}$. Let L be the language

every a is ultimately followed by a b (though there may be arbitrarily many c 's in between, and there may be only finitely many a 's)

Then L is recognizable.



Streamlining Büchi Automata

It is clear that a Büchi automaton may have useless states. In particular, any inaccessible state (in the sense of a classical finite state machine) is clearly useless. But there is more: for example, if a final state belongs to a trivial strongly connected component it is useless: the computation can pass through the state at most once, so we might as well remove it from the set of final states.

Lemma

Useless states can be removed from a Büchi automaton in linear time.

Exercise

Give a careful definition of what it means for a state in a Büchi automaton to be useless. Then produce a linear time algorithm to eliminate useless states.

Acceptance Testing

Since the acceptance condition involves infinity one might feel a bit uneasy about acceptance: given a Büchi automaton \mathcal{A} and an infinite word U , how do we check if \mathcal{A} accepts U ? We cannot simply run the machine on U .

If U is very simple, we can still handle the situation.

Lemma

Let \mathcal{A} be a Büchi automaton and $U = u^\omega$ a periodic infinite word. Then it is decidable whether \mathcal{A} accepts U .

Note that U has a finite description: we only need to write down the finite word u . The most general description is a computable function $U : \mathbb{N} \rightarrow \Sigma$. In this case, we call U recursive.

Exercise

Prove the last lemma.

A Catastrophe

Lemma

It is undecidable if a Büchi automaton accepts a recursive word.

Proof.

For any index e define a recursive infinite word U_e by

$$U_e(i) = \begin{cases} b & \text{if } \{e\}(e) \text{ converges after at most } i \text{ steps,} \\ a & \text{otherwise.} \end{cases}$$

Then $\{e\}(e)$ converges iff $U_e = a^k b^\omega$, which property is easily checked by a 2-state Büchi automaton. If we could test acceptance in a Büchi automaton we could thus solve the Halting Problem. \square

Union and Intersection

Lemma

Recognizable languages of Σ^ω are closed under union and intersection.

Proof. For union simply use the disjoint sum of the Büchi automata.

For intersection, we use a slightly modified product machine construction. The new state set is

$$Q_1 \times Q_2 \times \{0, 1, 2\}$$

The transitions on Q_1 and Q_2 are inherited from the two given machines.

On the last component act as follows:

- Move from 0 to 1 at the next input.
- From 1 move to 2 whenever a state in F_1 is encountered.
- Reset from 2 to 0 when a state in F_2 is encountered.

Proof, contd.

The initial states are of the form

$$I_1 \times I_2 \times \{0\}$$

and the final states are

$$F = Q_1 \times Q_2 \times \{0\}$$

The infinitely many visits to F imply infinitely many visits to F_1 and F_2 , and conversely. □

There is a message here: though the construction is similar to the finite case it is a bit more complicated. You have to stay alert. Also note that we have not dealt with complements.

Exercise

Fill in the details in the last construction.

- Infinite Words
- ② Deterministic Languages
 - Muller and Rabin Automata
 - Presburger Arithmetic
 - Determinizing Büchi Automata
 - Safra's Algorithm
 - Boolean Operations on Recognizable Languages

Deterministic Büchi Automata

Definition

A Büchi automaton is **deterministic** if it has one initial state and its transition system is deterministic.

Note that we may safely assume that a deterministic Büchi automaton is also complete: otherwise we can simply add one trap state.

In a deterministic and complete Büchi automaton there is exactly one run from the initial state for any input.

Note that the last lemma shows that even for deterministic Büchi automaton acceptance testing is undecidable in general.

Still, deterministic automata should be of interest if one tries to compute complements: to get a machine for the complement, manipulate the acceptance condition.

Another Catastrophe

Proposition

The ω -language L of all words in $\{a, b\}^\omega$ that contain only finitely many b 's is recognizable but cannot be accepted by any deterministic Büchi automaton.

Proof. To see this, suppose there is some deterministic Büchi automaton that accepts L .

Hence, for some n_1 , $\delta(q_0, ba^{n_1}) \in F$. Moreover, for some n_2 , $\delta(q_0, ba^{n_1} ba^{n_2}) \in F$. By induction we produce an infinite word

$$ba^{n_1} ba^{n_2} ba^{n_3} \dots$$

accepted by the automaton. Contradiction. □

This shows that a deterministic transition system together with a Büchi type acceptance condition $\text{Inf}(\pi) \cap F \neq \emptyset$ is not going to work: not only do we have to construct a deterministic transition system, we also have to modify our acceptance conditions. Alas, it is far from clear how one should do this.

Deterministic Recognizable Languages

One might wonder whether the languages recognized by deterministic Büchi automata have some natural characterization. Since you asked . . .

Definition

Let $L \subseteq \Sigma^*$ be a language. Define the **adherence** of L to be

$$\vec{L} = \{x \in \Sigma^\omega \mid x \text{ has infinitely many prefixes in } L\}$$

The best way to visualize this is to think of Σ^* as an infinite tree. Mark the nodes in this tree that belong to L . Then \vec{L} is the set of all branches in the tree that touch infinitely many marked nodes. Note that there may be no such branches even when L is infinite.

L	\vec{L}
a^*b	\emptyset
$(ab)^*$	$(ab)^\omega$
$(a^*b)^*$	$(a^*b)^\omega$

Adherence and Büchi

The reason the adherence operation is interesting is that it connects ordinary languages with ω -languages. Given a Büchi automaton we can think of it as an ordinary NFA and obtain an ordinary language $\mathcal{L}^*(\mathcal{A})$.

What is the relationship, if any, between $\mathcal{L}^*(\mathcal{A})$ and $\mathcal{L}^\omega(\mathcal{A})$?

Consider a trace of the Büchi automaton on some input $x \in \Sigma^\infty$:

$$\pi : p_0 x_0 p_1 x_1 p_2 \dots p_k x_k p_{k+1} \dots$$

If x is accepted by \mathcal{A} then for infinitely many i we have $p_i \in F$. But then $x_0 x_1 \dots x_{i-1}$ is accepted by the NFA \mathcal{A} .

In other words

$$\mathcal{L}^\omega(\mathcal{A}) \subseteq \overrightarrow{\mathcal{L}^*(\mathcal{A})}$$

The Characterization

Lemma

An ω -language L is recognized by a deterministic Büchi automaton if, and only if, L is the adherence of some regular language.

Proof.

We already know that $\mathcal{L}^\omega(\mathcal{A}) \subseteq \overrightarrow{\mathcal{L}^*(\mathcal{A})}$ for any Büchi automaton \mathcal{A} .

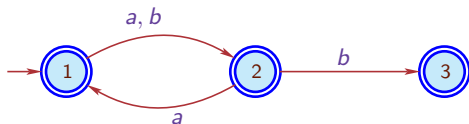
Now suppose \mathcal{A} is in addition deterministic. Then equality holds and we are done.

For the opposite direction assume $L = \overrightarrow{K}$ where K is regular. Then K is accepted by a deterministic finite state machine, for example the minimal automaton \mathcal{A} .

Thinking of \mathcal{A} as a Büchi automaton we have $\mathcal{L}^\omega(\mathcal{A}) = L$. □

Adherence Example

Here is a deterministic automaton \mathcal{A} that has finite computations that do not correspond to prefixes of any ω computation by \mathcal{A} . Here $\mathcal{L}^\omega(\mathcal{A}) = (\{a, b\} a)^\omega$ but $\mathcal{L}^*(\mathcal{A}) = (\{a, b\} a)^* \{a, b\}^{\leq 2}$.



Exercise

Explain precisely why this last argument fails when \mathcal{A} is nondeterministic: show that $\mathcal{L}^\omega(\mathcal{A}) \subsetneq \mathcal{L}^*(\mathcal{A})$ may happen.

Closure Properties

Lemma

Deterministic recognizable languages are closed under union and intersection.

Proof.

Union follows directly from the last lemma and $\overrightarrow{A \cup B} = \overrightarrow{A} \cup \overrightarrow{B}$.

For intersections note that the modified product machine construction from above preserves determinism. □

Exercise

Check all the details in the last proof.

- Infinite Words
- Deterministic Languages
- ③ Muller and Rabin Automata
 - Presburger Arithmetic
 - Determinizing Büchi Automata
 - Safra's Algorithm
 - Boolean Operations on Recognizable Languages

Muller Automata

One possibility is to completely pin down $\text{Inf}(\pi)$.

Definition

A **Muller automaton** consists of a deterministic transition system $\langle Q, \Sigma, \tau \rangle$ and an acceptance condition $q_0 \in Q$ and $\mathcal{F} \subseteq \text{pow}(Q)$.

M accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of M on x that starts at q_0 and such that $\text{Inf}(\pi) \in \mathcal{F}$.

\mathcal{F} is often referred to as the **table** of the Muller automaton. Note that the table may have size exponential in the size of the transition system.

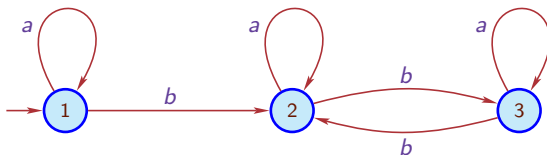
But, complementation is easy (just as it was easy for DFAs): make sure the machine is complete, then replace the old table \mathcal{F} by $\text{pow}(Q) - \mathcal{F}$.

Note that this simple operation might produce an exponential blow-up if done in a ham-fisted way.

Example: Muller

The at-least-one-but-finitely-many- b 's language from above is accepted by the following Muller automaton. The table has the form

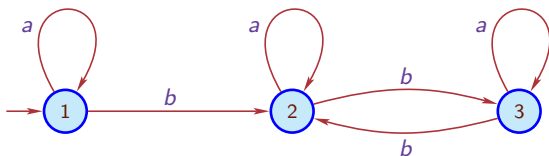
$$\mathcal{F} = ((2), (3))$$



Note that this automaton distinguishes between even and odd number of b 's. This is a bit scary since the distinction is by no means obvious from the original Büchi automaton.

The Complement

We can complement the table to get a machine for the complement of the language.



The complement table contains several useless entries (other than \emptyset):

1	1, 2	1, 3	2, 3	1, 2, 3
a^ω	\emptyset	\emptyset	$a^*(ba^*)^\omega$	\emptyset

However, then non-empty entries duly produce no b 's or infinitely many b 's, exactly the complement of the language.

Muller and Closure

As we have seen, the family of languages accepted by Muller automata is closed under complementation. It is in fact a Boolean algebra.

Lemma

Given two Muller automata M_1 and M_2 one can construct a new Muller automaton M such that $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

Proof.

As usual, we use a product machine $M = M_1 \times M_2$.

The only slightly tricky part is the table of M : it has the form

$$\{ F_1 \times F_2 \mid F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2 \}$$

Since the machines are deterministic it is easy to see that this works. □

Muller versus Deterministic Büchi

Lemma

A language $L \subseteq \Sigma^\infty$ is recognizable by a Muller automaton if, and only if, it is of the form

$$L = \bigcup_{i \leq n} U_i - V_i$$

where $U_i, V_i \subseteq \Sigma^\infty$ are recognizable by a deterministic Büchi automaton.

Proof.

Suppose L is recognized by M with table \mathcal{F} . Since $L = \bigcup_{F \in \mathcal{F}} \mathcal{L}(M, \{F\})$ we only need to deal with tables of size 1. But

$$\mathcal{L}(M(\{F\})) = \bigcap_{p \in F} \mathcal{L}(M(p)) - \bigcup_{q \notin F} \mathcal{L}(M(q))$$

where the automata on the right hand side are deterministic Büchi.

Done by the closure properties of deterministic Büchi languages.

Proof, contd.

For the opposite direction assume U is accepted by a deterministic Büchi automaton \mathcal{A} . Define

$$\mathcal{F} = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$$

Then the corresponding Muller automaton $\mathcal{A}(\mathcal{F})$ accepts U .

But we know that Muller languages form a Boolean algebra, so we can get a Muller automaton for any Boolean combination $\bigcup_{i \leq n} U_i - V_i$.

□

Rabin Automata

Another possibility to modify acceptance conditions is to augment the positive condition of Büchi automata by a negative condition: a successful run must ultimately avoid a certain set of states.

Definition

A **Rabin automaton** consists of a deterministic transition system $\langle Q, \Sigma, \tau \rangle$ and an acceptance condition $q_0 \in Q$ and $\mathcal{R} \subseteq \text{pow}(Q) \times \text{pow}(Q)$.

M accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of M on x that starts at q_0 and such that for some $(L, R) \in \mathcal{R}$: $\text{Inf}(\pi) \cap L = \emptyset$ and $\text{Inf}(\pi) \cap R \neq \emptyset$.

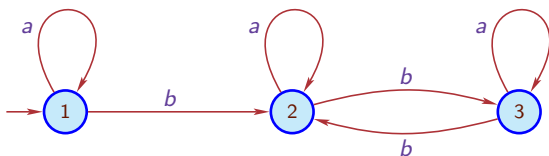
The pairs (L, R) are called **Rabin pairs**: L is the negative condition and R the positive condition.

In the special case where $\mathcal{R} = \{(\emptyset, F)\}$ we are dealing with a deterministic Büchi automaton.

Example: Rabin

The Muller automaton from above can also be turned into a Rabin automaton with Rabin pairs

$$\mathcal{R} = ((1, 2; 3), (1, 3; 2))$$



The excluded sets force a tail end of the run to look like 2^ω or 3^ω .

Equivalence Muller and Rabin

So now we have four classes of automata:

- deterministic Büchi,
- Büchi,
- Muller and
- Rabin.

We will see that Büchi, Muller and Rabin are all equivalent and strictly stronger than deterministic Büchi. This result is similar to equivalences between various types of automata on finite strings, but the arguments are more complicated.

We will not consider nondeterministic versions of Muller and Rabin automata here.

Muller to/from Rabin

Lemma

For every Rabin automaton there exists an equivalent Muller automaton, and conversely.

Proof.

Consider a Rabin automaton $\langle Q, \Sigma, \tau; \mathcal{R} \rangle$. We will use the same transition system and define a table

$$\mathcal{F} = \{ X \subseteq Q \mid \exists (L, R) \in \mathcal{R} (X \cap L = \emptyset, X \cap R \neq \emptyset) \}$$

It is easy to see that $\langle Q, \Sigma, \tau; \mathcal{F} \rangle$ is an equivalent Muller automaton.

Equivalence, contd.

The opposite direction is harder.

Let $\langle Q, \Sigma, \delta; \mathcal{F} \rangle$ be a Muller automaton, say, $\mathcal{F} = \{F_1, \dots, F_k\}$. Consider a new transition system on state set

$$Q' = \text{pow}(F_1) \times \dots \times \text{pow}(F_k) \times Q$$

and transitions

$$(U_1, \dots, U_k, p) \xrightarrow{a} (U'_1, \dots, U'_k, \delta(p, a))$$

where $U'_i = \emptyset$ if $U_i = F_i$ and $F_i \cap (U_i \cup \{\delta(p, a)\})$ otherwise. The Rabin pairs are defined by

$$L_i = \{(U_1, \dots, U_k, p) \mid p \notin F_i\} \quad R_i = \{(U_1, \dots, U_k, p) \mid U_i = F_i\}$$

One can verify that the new machine is equivalent to the given Muller automaton. □

From Rabin to Büchi

Lemma

For every Rabin automaton there exists an equivalent Büchi automaton.

Proof. Suppose \mathcal{A} is some Rabin automaton with n states and m pairs (L_i, R_i) .

Let

$$Q' = Q \cup \bigcup_i \{ (p, i) \in Q \times [m] \mid p \notin L_i \}$$

The Büchi automaton \mathcal{B} inherits the initial state and the transitions from \mathcal{A} ; furthermore, for each $p \xrightarrow{a} q$ in \mathcal{A} there are additional transitions

$$p \xrightarrow{a} (q, i) \quad (p, i) \xrightarrow{a} (q, i)$$

in \mathcal{B} whenever the corresponding states exist. Lastly, set

$$F = \{ (p, i) \mid p \in R_i \}$$

Done.

□

Efficiency

In the conversion Muller to Rabin the transition system is unchanged. However, we introduce exponentially many entries in the table, for each Rabin pair.

For the direction Rabin to Muller the new transition system already has potentially exponential size in the the size of the old transition system (k can be exponential in the number of states). The number of pairs is also exponential, and each pair has perhaps exponential size.

A Rabin automaton with n states an m pairs can be simulated by a Büchi automaton of size $O(nm)$.

Overall, these conversions will only be feasible for rather small machines.

Rational Languages

Even so, it is a good sign that Büchi, Muller and Rabin automata are all equivalent: it seems that recognizability is not a flimsy concept. Another piece of evidence pointing in this direction is that recognizable language on infinite words can be written down as a type of regular expression.

Definition

A language $L \subseteq \Sigma^\omega$ is **rational** if it is of the form

$$L = \bigcup_{i \leq n} U_i V_i^\omega$$

where $U_i, V_i \subseteq \Sigma^*$ are all regular.

Since we already have a notation system for regular languages of finite words it is easy to obtain a notation system for recognizable languages of ω -words: add one operation $^\omega$ with the understanding that this operation can only be used once and on the right hand side.

Rational Expressions

A basic expression has the form

$$\alpha \beta^\omega$$

where α and β are ordinary regular expressions. As we will see shortly, sums of these expressions then produce exactly all the recognizable ω -languages.

For the examples from above we have fairly simple expressions

$$a^* b (a + b)^* a^\omega$$
$$((b + c)^* (\varepsilon + ac^* b))^\omega$$

Equivalence Recognizable and Rational

Lemma

An ω -language is recognizable if, and only if, it is rational.

Proof. First assume \mathcal{A} is a Büchi automaton accepting some language L . For each final state p define two new automata

$$\mathcal{A}_p^0 = \mathcal{A}(I, p) \quad \mathcal{A}_p^1 = \mathcal{A}(p, p)$$

and let $U_p = \mathcal{L}(\mathcal{A}_p^0)$, $V_p = \mathcal{L}(\mathcal{A}_p^1)$.

Then

$$L = \bigcup_{p \in Q} U_p V_p^\omega$$

since $\text{Inf}(\pi) \cap F \neq \emptyset$ implies that one particular state $p \in F$ must appear infinitely often.

Proof, contd.

For the opposite direction it suffices to show that $L = UV^\omega$ is recognizable for any regular languages U and $V \neq \emptyset, \{\varepsilon\}$ since recognizable languages are closed under union.

To this end consider two machines \mathcal{A}_0 and \mathcal{A}_1 for U and V . Join the final states of \mathcal{A}_0 to the initial states of \mathcal{A}_1 by ε -moves, and the final states of \mathcal{A}_1 to the initial states of \mathcal{A}_1 .

Perform ε -elimination to obtain a plain nondeterministic automaton \mathcal{A} .

Set the initial states of \mathcal{A} to the initial states of \mathcal{A}_0 and the final states to the final states of \mathcal{A}_1 .

The resulting Büchi automaton \mathcal{A} accepts L .



- Infinite Words
- Deterministic Languages
- Muller and Rabin Automata
- 4 Presburger Arithmetic
 - Determinizing Büchi Automata
 - Safra's Algorithm
 - Boolean Operations on Recognizable Languages

MSO on Infinite Words

We can define $W \models \varphi$ for an infinite word $W \in \Sigma^\omega$ and a MSO[<] formula φ just like in the finite case.

Since we skipped over one direction in the proof for the finite case, here is a detailed argument.

We are going to construct a machine by induction of the build-up of the formula. One problem one has to confront is to make sense of free variables.

For example, what should the automaton for

$$X(x) \wedge Q_b(x)$$

do?

It needs to operate not just on the actual string W but it needs additional information for x and X .

Adding Tracks

The solution is similar to what we did to build automata for relations: one adds tracks to the actual word W , one for each first order variable and one for each second order variable.

W	a	a	b	a	a	a	b	a	a	a	\dots
x	0	0	1	0	0	0	0	0	0	0	\dots
X	0	0	1	0	1	0	1	0	0	0	\dots

These additional tracks are binary.

Moreover, for first order variables there must be exactly one 1 in the track; the word in this track has to be in 0^*10^ω .

The word in a second order track has to be in $\{0, 1\}^\omega$.

Building Automata

A Büchi automaton scanning such an augmented word from left to right can easily test the basic predicates

$$x = y \quad x < y \quad Q_a(x)$$

and verify that the variable tracks have the right format.

Boolean connectives \wedge and \vee can be handled by building a product automaton or a disjoint union, respectively.

Negation is harder; it requires a determinization algorithm that we will discuss later.

Existential quantifiers are dealt with by projection (erasing a track) and universal quantifiers are reduced to existential one plus two negations.

Büchi's Theorem

Theorem (Büchi 1960)

$\text{MSO}[\<]$ is decidable over Σ^ω : for every sentence φ one can effectively construct an automaton \mathcal{A}_φ whose acceptance language is non-empty iff φ is valid.

Here is a trivial example: $\exists x Q_a(x)$.



Weak Monadic Second Order

Note that the automata construction would also work if we constrained the the set variables to range over finite sets only: we just have to check that the corresponding tracks are in

$$\{0, 1\}^* 0^\omega$$

We already know how to do this with a Büchi automaton.

It follows that **weak monadic second order** with $<$ (or successor, it does not matter) is also decidable.

Who Cares?

One might wonder why Büchi's theorem is important outside of theoretical computer science.

One-way infinite words arise naturally in the study of non-terminating programs (such as operating systems) or certain protocols, so it is important to have some tools available to deal with infinite words.

Another application is perhaps more surprising: logic on infinite words can be used to express assertions in arithmetic – which, in turn, are important for program verification.

Full Arithmetic

Ordinary arithmetic can be considered as the study of the structure

$$\mathfrak{N} = \langle \mathbb{N}, +, *, <, 0, 1 \rangle$$

Alas, even statements of the form

$$\exists x_1, \dots, x_n \varphi(x_1, \dots, x_n)$$

are undecidable over \mathfrak{N} in general (where φ has only bounded quantifiers).

This follows from the undecidability of Diophantine equations: we can have φ express that a polynomial with integer coefficients has a root at x_1, \dots, x_n .

More generally, every semi-decidable set $A \subseteq \mathbb{N}$ has a representation

$$a \in A \iff \mathfrak{N} \models \exists x \varphi(x, a)$$

where φ has only bounded quantifiers.

Presburger Arithmetic

How about weaker structures that have fewer operations?

Realistically, the only useful choice is to drop multiplication. This yields **Presburger arithmetic**:

$$\mathfrak{N}_0 = \langle \mathbb{N}, +, <, 0 \rangle$$

Since multiplication is missing one cannot describe polynomials in this setting, only linear combinations.

So the problem of Diophantine equations disappears.

Admissible Operations

Full multiplication is absent, but multiplication by a constant is available; for example

$$y = 3 * x \iff y = x + x + x$$

We can also do modular arithmetic with fixed modulus:

$$y = x \bmod 2 \iff \exists z (x = 2 * z + y \wedge y < 2)$$

$$y = x \operatorname{div} 2 \iff \exists z (x = 2 * y + z \wedge z < 2)$$

A non-trivial example of a valid Presburger formula:

$$\exists x \forall y \exists u, v (x < y \Rightarrow x = 5 * u + 7 * v)$$

Presburger Arithmetic is Decidable

Without multiplication arithmetic is much less complicated.

Theorem (M. Presburger 1929)

First order logic over \mathfrak{N}_0 is decidable.

Presburger's original algorithm is based on **quantifier elimination**: a formula is translated into an equivalent formula that has one fewer quantifier.

Unfortunately, it turns out that the complexity of Presburger arithmetic is pretty bad:

$$\Omega(2^{2^{cn}}) \text{ and } O(2^{2^{2^{cn}}})$$

Using WMSO[<]

WMSO[<] can be used to give a decision procedure for Presburger arithmetic that seems to work reasonably well in practice (though, in principle, the use of determinization could cause blow-up).

One might think that natural numbers would be represented by first order variables ranging over positions in a word: after all, in an infinite word these positions are just \mathbb{N} .

Alas, that won't work: we need to be able to check addition. We would need a machine that accepts three track binary words of the form

$$0^i 10^\omega : 0^j 10^\omega : 0^{i+j} 10^\omega,$$

which is clearly impossible.

Binary Representation

The trick is to represent natural numbers by second order variables, finite sets $X \subseteq \mathbb{N}$:

$$\text{val}(X) = \sum_{i \in X} 2^i$$

Thus, X is essentially just the standard binary expansion, LSD first.

Now an automaton can check $\text{val}(X) + \text{val}(Y) = \text{val}(Z)$:

X	1	0	1	1	1	0	1	0	0	0	...
Y	0	0	1	0	1	0	0	0	0	0	...
Z	1	0	0	0	1	1	1	0	0	0	...

It is even easier to check $\text{val}(X) < \text{val}(Y)$.

OK, but Why?

In programs that are not too terribly complex, index arithmetic can often be described in terms of Presburger arithmetic.

Being able to check the validity of Presburger formulae is thus directly relevant in program verification.

This is used for example in Microsoft's Spec# system, an extension of C# that includes specifications and tools to verify these specifications.

- Infinite Words
- Deterministic Languages
- Muller and Rabin Automata
- Presburger Arithmetic
- ⑤ Determinizing Büchi Automata
 - Safra's Algorithm
 - Boolean Operations on Recognizable Languages

Determinization

We now close the ring of equivalences by showing that every Büchi automaton has an equivalent Rabin automaton. Note the trade-off: the Rabin automaton must be deterministic, but it has a more flexible (more complicated) acceptance condition.

Theorem

For every Büchi automaton there exists an equivalent Rabin automaton.

It is worth to think a bit about why this is an interesting result.

The natural first attempt at determinization is to use the classical Rabin-Scott method as in the case of finite words – replace states by sets of states to construct a deterministic machine from the given nondeterministic Büchi automaton.

No Luck

Unfortunately, for the “infinitely many b 's” example from above we get



There is no way the power automaton on the right can be made to accept the right language, no matter whether we deal with Rabin or Muller automata: The second state 1, 2 must be recurrent on any accepting run, but then there is a run accepting b^ω .

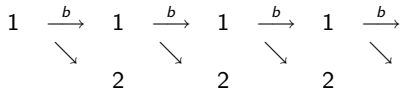
It seems that we need a construction more powerful than Rabin-Scott.

The Problem

Suppose $\mathcal{A} = \langle Q, \Sigma, \tau; I, F \rangle$ is some Büchi automaton and \mathcal{B} the corresponding power automaton. The problem is that \mathcal{B} only keeps track of the set of all reachable states:

$$I \longrightarrow P_x \longrightarrow P_{xy} \longrightarrow P_{xyz} \longrightarrow \dots$$

Suppose all the displayed states contain some $q \in F$. Then there is no reason whatsoever why \mathcal{A} should have an accepting run on $xyz\dots$: The final states may not lie on the same infinite branch. There is some infinite run of \mathcal{A} , but it may well fail to be accepting.



Our power automaton accepts too much.

The Solution, Sort Of

To fix this problem we need to consider subsets of $P'_u \subseteq P_u = \delta(I, u)$ that do not have this problem. For example, we want that

$$P'_x \longrightarrow P'_{xy}$$

implies that every state $p \in P'_{xy}$ is the target of a run of \mathcal{A} starting at a state $q \in P'_x$ that contains a final state.

Of course, we have no idea what P' should be or how to keep track of having hit an intermediate final state.

The trick is to consider $P \cap F$ whenever this set is not empty.

Unfortunately, we have to iterate the trick.

Safra Trees

The best way to organize the computation of the states of \mathcal{B} is to use ordered labeled trees, so-called **Safra trees**.

Each node in a Safra tree carries three pieces of information. Assume that the Büchi automaton has n states.

- Name: $v \in \{1, 2, \dots, 2n\}$.
- Label: $\emptyset \neq \lambda(v) \subseteq Q$.
- Mark: a bit.

The names of all nodes in a tree are always distinct. $2n$ is a magic number, but we will see in a while why it works. The root is always node 1. The mark bit is also restricted: only leaves can be marked.

In a sense, we will run the power automaton construction on all the nodes of the tree.

The Safra Conditions

In order to constrain the possible number of Safra trees we impose several conditions:

$$(S1) \bigcup_{u \text{ par } v} \lambda(v) \not\subseteq \lambda(u)$$

$$(S2) \text{ } u \text{ and } v \text{ incomparable implies } \lambda(v) \cap \lambda(u) = \emptyset$$

Thus if v_1, v_2, \dots, v_k are the children of u then their labels form a partition of a proper subset of $\lambda(u)$.

Proposition

A Safra tree has at most n nodes.

Proof. For every node v there exists a state $p \in \lambda(v)$ that appears nowhere else in the tree. □

Counting Trees

Of course, the number of these trees is still wildly exponential: the only obvious bound is

$$2^{O(n \log n)}$$

This is uncomfortably large, but at least it's finite: we can use Safra trees as states in the deterministic machine.

It remains to explain how to compute the transition function

$$\delta(T, a) = T'$$

where T and T' are Safra trees and $a \in \Sigma$.

Batten down the hatches.

- Infinite Words
- Deterministic Languages
- Muller and Rabin Automata
- Presburger Arithmetic
- Determinizing Büchi Automata
- ⑥ Safra's Algorithm
 - Boolean Operations on Recognizable Languages

Set-Up

Suppose

$$\mathcal{B} = \langle Q, \Sigma, \tau; I, F \rangle$$

is an arbitrary Büchi automaton on n states.

We want to construct a (deterministic) Rabin automaton \mathcal{A} whose states will be Safra trees over \mathcal{B} .

The initial tree is

- $(1 : I)$ if $I \cap F = \emptyset$,
- $(1 : I!)$ if $I \subseteq F$ (root is marked),
- $(1 : I; 2 : I \cap F!)$ otherwise (leaf is marked).

The Steps

- 1 **Unmark**
Unmark all the nodes in the tree.
- 2 **Update**
Replace $\lambda(v)$ by $\tau(\lambda(v), a)$.
- 3 **Create**
If $\lambda(v) \cap F \neq \emptyset$ attach a new rightmost child u to v .
Set $\lambda(u) = \lambda(v) \cap F$ and mark u .
- 4 **Horizontal Merge**
Remove all states in $\lambda(u)$ that appear in nodes v to the left of u .
- 5 **Kill empty**
Remove all nodes with empty label set.
- 6 **Vertical Merge**
Mark all states u such that $\lambda(u) = \bigcup_{u \text{ par } v} \lambda(v)$ and remove all descendants.

Invariance

Note that the Update and Create steps usually destroy the Safra properties: the transition function moves the states around and the disjointness conditions will fail to hold in general.

However, after Horizontal Merge the same state can only appear along a branch in the tree, so (S2) holds.

After killing empty nodes all label sets are non-empty.

After Vertical Merge condition (S1) also holds.

Hence, after step 6, the new tree is indeed a Safra tree.

Exercise

Check in detail that the new tree is Safra.

Details

Note that the new names in the Create step must be chosen from V – current nodes.

In practice, the choice is always

$$\text{new} = \min(V - \text{current nodes}).$$

This works fine since there can be at most n nodes before step 3.

In Horizontal Merge we move from left to right; of course, this is completely arbitrary – we could just as well work right to left.

The nodes are critical, we are not just dealing with trees of a certain shape. For example, the tree $(1 : P, 2 : R)$ is **not** the same as $(1 : P, 3 : R)$. The construction breaks without this distinction.

The Whole Rabin Machine

The Rabin machine \mathcal{A} is defined as follows: The state set is the collection of Safra trees generated by applying the 6-step-procedure in all possible ways starting at the initial tree.

This process also produces the transition function of \mathcal{A} .

The Rabin pairs of \mathcal{A} are (L, R) where v is some node and

$$L = \{ T \mid v \notin T \} \quad R = \{ T \mid v \in T, \text{ marked} \}$$

Of course, we only need consider nodes $v \in V$ that appear marked in at least one tree.

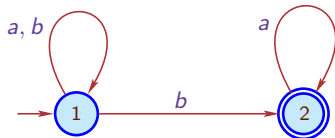
That's all.

Example I

Use the old example

$$L = \{x \in \{a, b\}^\omega \mid 1 \leq \#_b x < \infty\}$$

of words containing at least one but only finitely many b 's is recognizable. A Büchi automaton \mathcal{A} for L looks like so:



The Rabin automaton \mathcal{B} has initial state $(1 : 1)$.

Computing All States

$$\begin{array}{lcl}
 (1 : 1) & \xRightarrow{a} & (1 : 1) \\
 (1 : 1) & \xRightarrow{b} & (1 : 1, 2; 2 : 2!) \\
 (1 : 1, 2; 2 : 2!) & \xRightarrow{a} & (1 : 1, 2; 2 : 2!) \\
 (1 : 1, 2; 2 : 2!) & \xRightarrow{b} & (1 : 1, 2; 3 : 2!) \\
 (1 : 1, 2; 3 : 2!) & \xRightarrow{a} & (1 : 1, 2; 3 : 2!) \\
 (1 : 1, 2; 3 : 2!) & \xRightarrow{b} & (1 : 1, 2; 2 : 2!)
 \end{array}$$

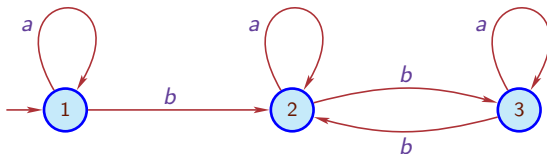
The Rabin pairs are

$$((1, 2; 3), (1, 3; 2))$$

since 2 and 3 are the only marked nodes.

The Diagram

The diagram should look familiar:

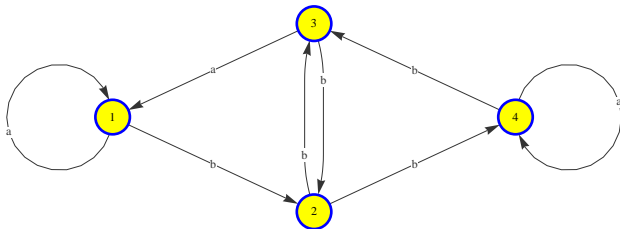


This is the machine we already saw last time.

In this particular case we have already verified that the machine behaves properly.

Example II

Recall the de Bruijn automata describing CAs. If we construe one of these as a Büchi automaton (why not?), what is the corresponding Rabin automaton going to look like?



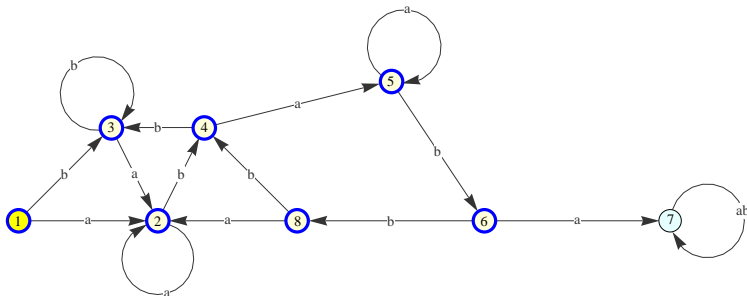
So there are 4 states, all initial and final.

What happens if Safra's algorithm is applied to this automaton?

Think, Don't Compute

Since all states are final, the tree construction never produces anything but a tree with one node, the root.

In other words, we are just running the classical Rabin-Scott algorithm. For ECA 110 the algorithm produces the following machine:



States

The 8 states (labels of the root) are:

$$\{1, 2, 3, 4\}, \{1, 4\}, \{2, 3, 4\}, \{2, 3\}, \{1\}, \{2\}, \emptyset, \{3, 4\}$$

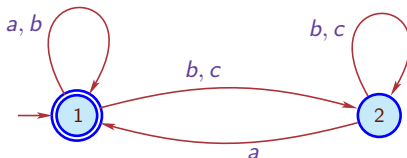
State number 1 (corresponding to $\{1, 2, 3, 4\}$ is initial), everybody except for the sink 7 (corresponding to \emptyset) is final.

This is mildly reassuring: when $Q = F$ the only question is whether there is an infinite run on some given word $x \in \Sigma^\omega$.

The standard Rabin-Scott construction can take care of that, and Safra's algorithm degenerates to Rabin-Scott in this case.

Example III

Here is another Büchi automaton \mathcal{B} on alphabet $\{a, b, c\}$.
This one is slightly more complicated.

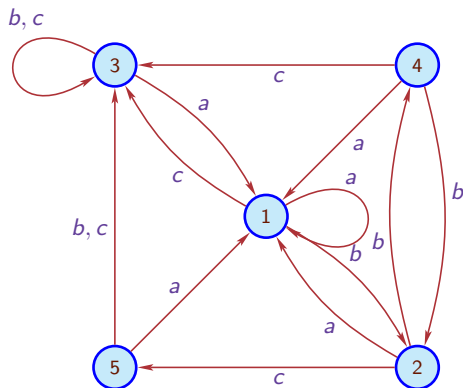


The language is a

$$((b + c)^* a + b)^\omega$$

Example III, contd.

The Rabin automaton with pairs $((\emptyset; 1, 4, 5), (1, 3, 4, 5; 2))$



Comments

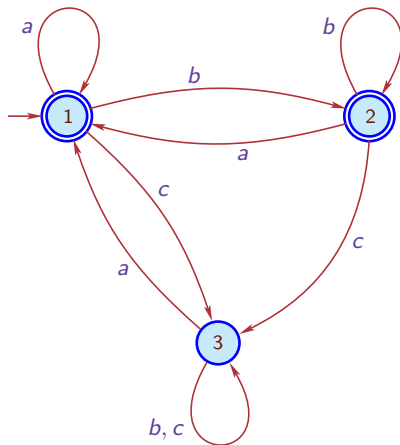
The Safra trees corresponding to the 5 states are

- 1 (1 : 1!)
- 2 (1 : 1, 2; 2 : 1!)
- 3 (1 : 2)
- 4 (1 : 1, 2!)
- 5 (1 : 2!)

The second Rabin pair is useless: there is no run that conforms to (1, 3, 4, 5; 2). Hence we really have built a deterministic Büchi automaton.

Alas, the last machine is too big: by “visual inspection” one finds that we could merge states 3 and 5, as well as 2 and 4.

After Merging



As a Büchi automaton $F = \{1, 2\}$.

Why Should This Work?

The key property of the construction is the following lemma (which says, in essence, our original plan has been duly implemented).

Lemma

Suppose T is a Safra tree in \mathcal{A} that contains a marked node v . Let $x = x_1x_2 \dots x_k$ be a finite word such that v is an unmarked node in $\delta(T, x_1 \dots x_i)$ for $i < k$ and a marked node in $\delta(T, x_1 \dots x_k)$. Let P_i be the label sets associated with node v in these trees.

Then $P_i \subseteq \delta(P_0, x_1 \dots x_i)$ and for all $p \in P_k$ there is a run in the Büchi automaton starting at some $q \in P_0$ that touches a final state.

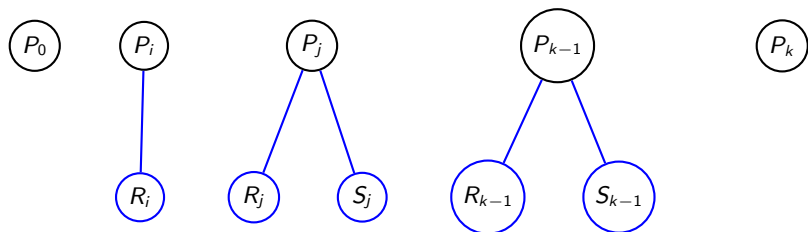
Proof.

We forgo the opportunity to inflict significant cognitive pain on the student body and do not prove the general case: we will only deal with the case where v is the root.

Proof, contd.

Since the root has no siblings to the left we have $P_i = \delta(P_0, x_1 \dots x_i)$.

Since P_k is marked, at time $k - 1$ we must have had a tree where the root had children; for simplicity let's assume there are only 2 children. Hence there are times $0 < i < j < k$ where the children were introduced:



But then any run from P_0 to P_k passes through a final state: $R_i = P_i \cap F$ and $S_j \subseteq P_j \cap F$.

□

Correctness

Theorem

Let \mathcal{A} be the Rabin automaton obtained by applying Safra's algorithm to a Büchi automaton \mathcal{B} . Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

Proof.

First assume \mathcal{A} accepts $x \in \Sigma^\omega$. Then there is a node named v that appears infinitely often marked in the run of \mathcal{A} on x . Moreover, after some initial segment, all trees in the run contain v . Since v is marked infinitely often there is a chain of state sets P_{t_i} , $i \in \mathbb{N}$ and $t_i < t_{i+1}$, that appear as labels of v when the node is marked.

By the lemma every state in $P_{t_{i+1}}$ can be traced back to a state in P_{t_i} . By induction, there is a partial (meaning finite) run starting at l to every state in P_{t_i} for all i .

Think of these runs as defining nodes in a tree, the tree of all finite initial segments of computations of the Büchi automaton. Clearly, the tree is finitely branching and is infinite. By König's lemma it must contain an infinite branch – which branch corresponds to an accepting computation of \mathcal{B} on x .

Proof, contd.

For the opposite direction let \mathcal{B} accept x and let π be a corresponding run. Then there is a final state p that appears infinitely often in a run of \mathcal{B} on x . Then the states p_i appear in the root of the Safra trees in the unique run of \mathcal{A} on x . If the root is marked infinitely often \mathcal{A} accepts and we are done.

Otherwise, since a final state appears infinitely often in π , the final state must appear in child of the root. After a while, it will settle down in the leftmost position. If the corresponding node is marked infinitely often we are done.

Otherwise, by the same argument, we consider a node at level 2.

Since the trees have bounded depth we must ultimately reach a level where the node is marked infinitely often, and \mathcal{A} accepts x .

□

Complexity

It is clear that Safra's algorithm is quite difficult. Indeed, some software systems avoid the construction of a deterministic Rabin automaton and require the user to provide a corresponding machine.

The only general bound on the size of the Rabin automaton is

$$2^{O(n \log n)}$$

Unfortunately, this is asymptotically optimal.

Exercise

Implement Safra's algorithm in the language of your choice.

- Infinite Words
 - Deterministic Languages
 - Muller and Rabin Automata
 - Presburger Arithmetic
 - Determinizing Büchi Automata
 - Safra's Algorithm
- 7 Boolean Operations on Recognizable Languages

Complementation

Lemma

Recognizable languages of Σ^ω are closed under complementation.

Proof. To see this, first construct a Rabin automaton for the language. Then convert the Rabin automaton into an equivalent Muller automaton (which is still deterministic). We know how to complement the Muller automaton. \square

Hence recognizable ω -languages again form a Boolean algebra, and the operations are effective: we can compute the corresponding machines.

Note that concatenation and Kleene star make no sense for infinite words (though concatenation of finite and infinite words does make sense).

Complexity

If we measure complexity in terms of the minimal number of states in any Büchi automaton recognizing the language we have

union	$O(n_1 + n_2)$
intersection	$O(n_1 n_2)$
complement	$2^{O(n \log n)}$

The horrendous upper bound for complementation is not just theoretical: there are languages L_n accepted by a Büchi automaton on $n + 2$ states such that any Büchi automaton for the complement of L_n has at least $n!$ states.

Thus any algorithm using repeated complementation will be of very limited practical use.

Summary

- There are 3 equivalent machine models for recognizable sets of infinite words: Büchi automata, Muller automata and Rabin automata.
- Deterministic Büchi automata are strictly weaker than the others.
- Safra's algorithm allows one to construct a deterministic Rabin automaton for every recognizable language of infinite words.
- Complexity is a huge problem in Safra's algorithm, a lot of work has gone into streamlining the algorithm.