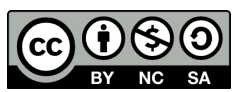
CDM

Automata on Infinite Words

Klaus Sutner

Carnegie Mellon University

Spring 2021



1 Infinite Words

2 Deterministic Languages

3 Muller and Rabin Automata

4 Digression: Algebra and FSM

5 Determinization

A Challenge: Does it make any sense to consider finite state machines on infinite words?

If so, how would this generalization work?

As a matter of principle, infinite words come in two flavors: bi-infinite

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

or one-way infinite

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

Both kinds appear naturally in the analysis of symbolic dynamical systems (reversible and irreversible).

One-way infinite ones can be used to describe the properties of programs that never halt, such as operating systems and user interfaces. Protocols also naturally give rise to infinite descriptions.

Note that neither Σ^∞ nor Σ^ω form a semigroup under concatenation in any conceivable sense of the word: there is no way to combine two infinite words by "placing one after the other" and get another infinite word (at least not of the kind that we are interested).

But not that there is an obvious concatenation operation

$$\Sigma^\star \times \Sigma^\omega \longrightarrow \Sigma^\omega$$

and a slightly less obvious one of type

$$\Sigma^\omega \times \Sigma^\omega \longrightarrow \Sigma^\infty$$

The second one is particularly interesting in conjunction with automata on bi-infinite words, but we won't go there: the technical details are too messy.

Key Question: How do we modify finite state machines to cope with infinite inputs?

- Transition system: same as for ordinary finite state machines.
- Acceptance condition: requires work.

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

Whatever condition we choose, we should not worry about actual acceptance testing, this is a conceptual problem, not an algorithmic one.

But also remember the second killer-app: model checking. We only care whether some machine accepts some input, the actual input is irrelevant.

So we are interested in one-way infinite words:

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

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.

Since we will not change the underlying transition systems, we can lift the definitions of run and trace to the infinite case: a run is an alternating infinite sequence

$$\pi = p_0, a_1, p_1, a_2, \ldots, p_{m-1}, a_m, p_m, \ldots$$

The corresponding infinite sequence of symbols is the trace:

$$\mathsf{lab}(\pi) = a_1, a_2 \ldots, a_{m-1}, a_m, \ldots \in \Sigma^\omega$$

In general, the number of runs on a particular input is going to be uncountable, but that will not affect us (it is path existence that matters).

Again: Key Question: What could it possibly mean for a finite state machine to accept an infinite word?

Obviously we need some notion of acceptance that does not just depend on a finite initial segment of the input: this would ignore "most" of the input and just replay our old theory.

On the other hand, we should keep things simple and not use wildly infinitary conditions to determine acceptance; we don't want to sink in a morass of descriptive set theory.

So here is a fairly natural condition: let's insist that an accepting run must touch the set of final states infinitely often. More precisely, define the set of recurrent states of a run π to be

$$\mathrm{rec}(\pi) = \{\, p \in Q \mid \overset{\infty}{\exists} i \,(p_i = p) \,\}$$

Definition

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

$\mathcal{B}$ accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of $\mathcal{B}$ on x that starts at I and such that $\mathrm{rec}(\pi) \cap F \neq \emptyset$.

The collection of all such words is the acceptance language of $\mathcal{B}$. A language $L \subseteq \Sigma^\omega$ is recognizable or ω-regular if there is some Büchi automaton that accepts it.

So far, this is just a definition. It seems reasonable, but it is absolutely not clear at this point that we will get any mileage out of this.

The standard acceptance testing problem makes little sense in this setting.

Problem:	**Recognition Büchi**
Instance:	A Büchi automaton $\mathcal{A}$ and a word $x \in \Sigma^\omega$.
Question:	Does $\mathcal{A}$ accept input x?

$\mathcal{A}$ will just be some finite data structure. But there is no general way to specify the input x, the space Σ^ω is uncountable. We could consider periodic words or the like, but as stated the decision problem is basically meaningless.

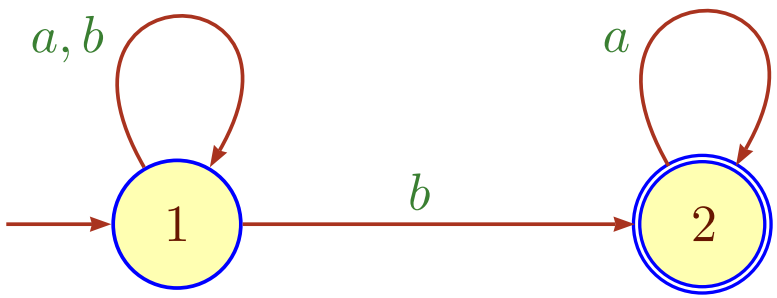
But: We might still be able to generalize our logic approach and use Büchi automata to solve decision problems.

Let $\Sigma = \{a, b\}$ and

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

So L is the language of all words containing at least one, but only finitely many b's. This language is recognizable.

In fact, two states suffice. Here is a Büchi automaton for $L = \{a, b\}^\star b\, a^\omega$:

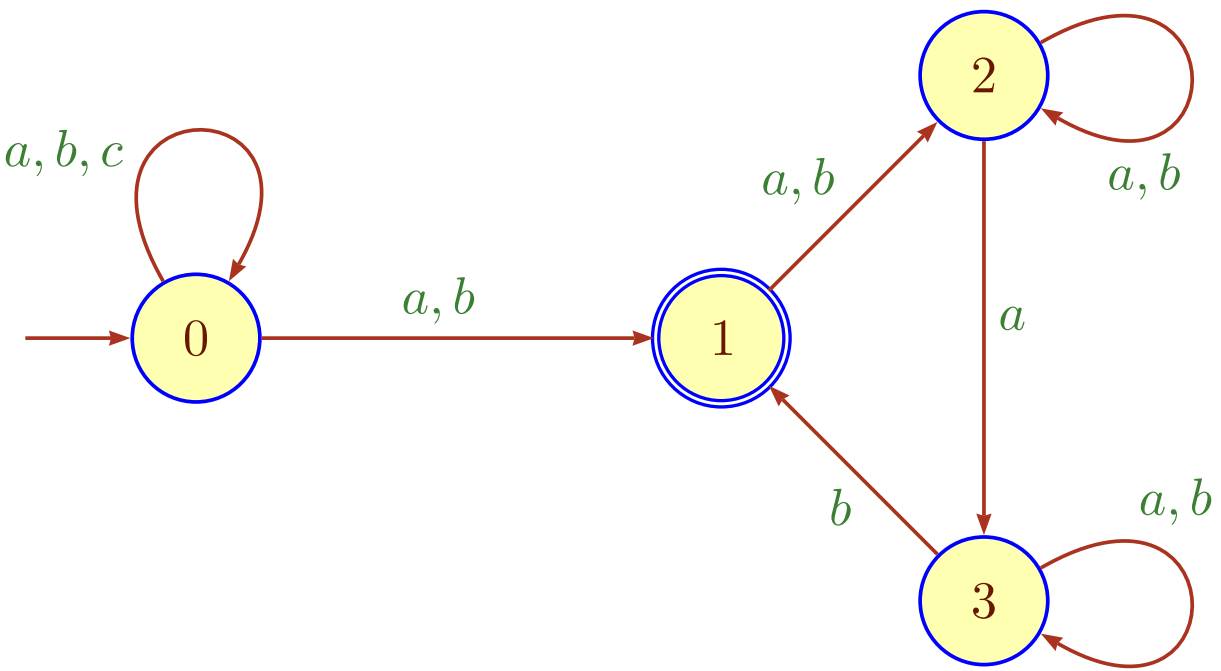


Example II

Let $\Sigma = \{a, b, c\}$ and

$$L = \{\, x \in \{a, b, c\}^\omega \mid \#_a x = \#_b x = \infty \wedge \#_c x < \infty \,\}$$

So L contains finitely many c's, but infinitely many a's and b's. This language is also recognizable.



Note that $0 \xrightarrow{a} 1$ or $0 \xrightarrow{b} 1$ would also work; it's not so clear what the canonical automaton looks like.

Correctness proofs are harder than for ordinary automata, they typically involve some (modest) amount of infinite combinatorics. In this case, one might use the following claims. Write K for the ω-language over $\{a, b\}$ of words containing infinitely many a's and b's.

1. A word x is in L iff $x = uv$ where $u \in \{a, b, c\}^\star$ and $v \in K$.
2. Let $s \in \{a, b\}$. Then $sv \in K$ iff $v \in K$.
3. Any infinite path in the SCC $\{1, 2, 3\}$ touching state 1 infinitely often must use the edges $(2, 3)$ and $(3, 1)$ infinitely often.

Exercise

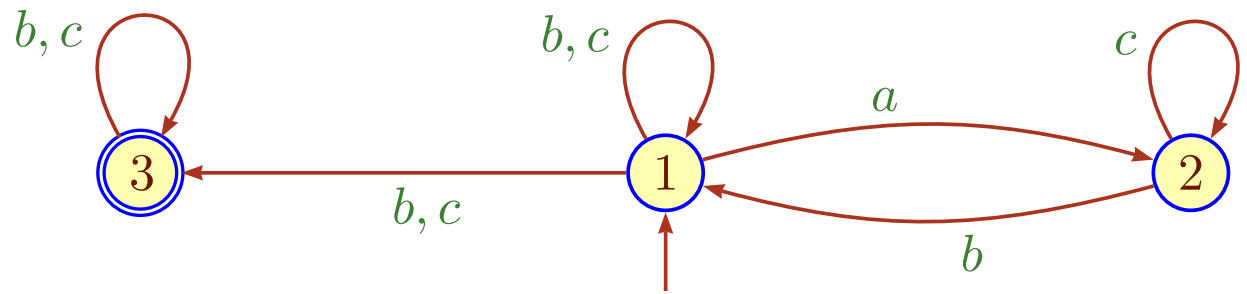
Give a complete proof that the Büchi automaton accepts L.

Example III

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

> *Every a is ultimately followed by a b, though there may be arbitrarily man c's in between, and overall there may be only finitely many a's.*

Then L is recognizable.



Again, think about what a correctness proof would look like.

While acceptance testing in general makes no sense, we can still handle the situation when the word is very simple.

Lemma

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

In this case, U has an obvious finite description: the finite words v and u. A more general case is a computable word: the function $U : \mathbb{N} \to \Sigma$ is computable.

Exercise

Prove the last lemma.

Lemma

It is undecidable if a Büchi automaton accepts a computable word (even a primitive recursive one).

Proof.

For any index e define a computable infinite word U_e by

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

Then $\{e\}(e)$ converges iff $U_e \in a^\star b^\omega$; otherwise $U_e = b^\omega$. This property is easily checked by a 2-state Büchi automaton (even a deterministic one). If we could test acceptance of a computable word in a Büchi automaton we could thus solve the Halting Problem. □

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.

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.

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.

Another piece of evidence for the usefulness our our definition 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 $U_i, V_i \subseteq \Sigma^\star$ 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.

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

$$a^{\star}b(a+b)^{\star}a^{\omega}$$

$$((b+c)^{\star}(\varepsilon+ac^{\star}b))^{\omega}$$

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

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

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

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.

□

1 Infinite Words

2 **Deterministic Languages**

3 Muller and Rabin Automata

4 Digression: Algebra and FSM

5 Determinization

Definition

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

We may safely assume that a deterministic Büchi automaton is also complete: otherwise we can simply add one sink state. In a deterministic and complete Büchi automaton there is exactly one run from the initial state for any input.

Note that the undecidability result for computable words holds already for deterministic Büchi automata.

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

Proposition

Let $L = \{ x \in \{a, b\}^\omega \mid \#_b x < \infty \}$. Then L is ω-regular 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

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

accepted by the automaton. Contradiction. □

This shows that a deterministic transition system together with a Büchi type acceptance condition $\mathrm{rec}(\pi) \cap F \neq \emptyset$ 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.

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

Definition

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

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

The best way to visualize this to think of $\Sigma^\star$ as an infinite tree. Mark the nodes in this tree that belong to L. Then $\overrightarrow{L}$ 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	$\overrightarrow{L}$
$a^\star b$	$\emptyset$
$(ab)^\star$	$(ab)^\omega$
$(a^\star b)^\star$	$(a^\star b)^\omega$

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 on ordinary language $\mathcal{L}^*(\mathcal{A})$: just change the acceptance condition.

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

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

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

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

In other words

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

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 $\mathcal{A}$.

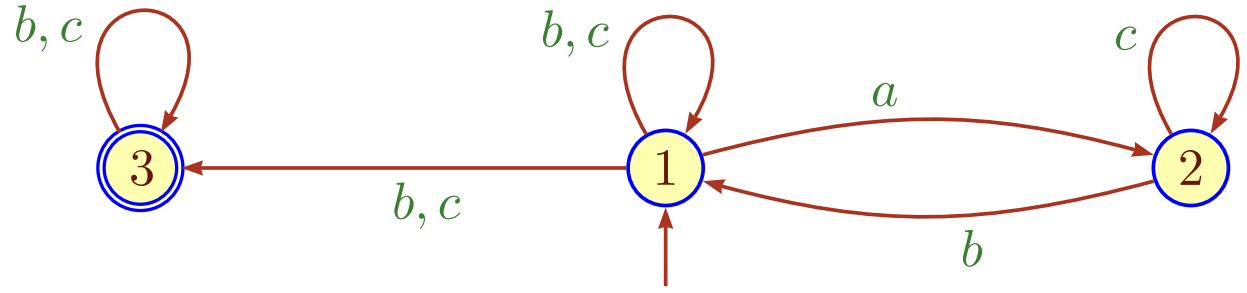
Now suppose $\mathcal{A}$ is in addition deterministic. Then equality holds and we are done.

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

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

But beware . . . Adherence Example

Consider again the automaton



Note that $(abb)^\star \subseteq \mathcal{L}^*(\mathcal{A})$.

But then $(abb)^\omega \in \overrightarrow{\mathcal{L}^*(\mathcal{A})} - \mathcal{L}^\omega(\mathcal{A})$

The problem is that our finite computations do not have infinite extensions.

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.

1 Infinite Words

2 Deterministic Languages

3 **Muller and Rabin Automata**

4 Digression: Algebra and FSM

5 Determinization

As we have seen, there are regular ω-languages that cannot be accepted by any deterministic Büchi automaton. Alas, without determinization it is unclear how we could deal with complements, which we need to handle negation.

Wild Hope: Maybe there are alternative machine models that allow for deterministic descriptions of regular languages.

As before, we will not change the transition system, just the acceptance condition.

One fairly natural possibility is to completely pin down $\mathrm{rec}(\pi)$.

Definition

A Muller automaton consists of a deterministic transition system $\langle Q, \Sigma, \tau \rangle$ and an acceptance condition $q_0 \in Q$ and $\mathcal{F} \subseteq \mathfrak{P}(Q)$.
$\mathcal{A}$ accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of $\mathcal{A}$ on x that starts at q_0 and such that $\mathrm{rec}(\pi) \in \mathcal{F}$.

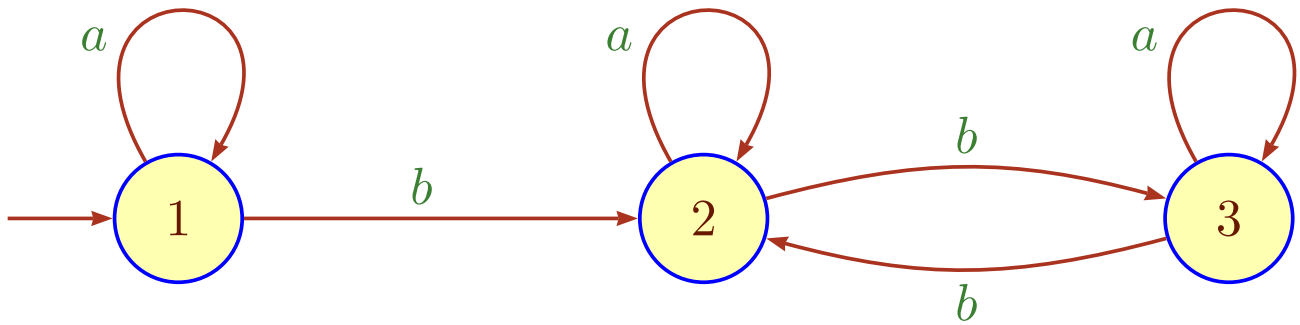
$\mathcal{F}$ is often referred to as the table of the Muller automaton. Note that the table may have size exponential in the size of the transition system.

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 $\mathfrak{P}(Q) - \mathcal{F}$.

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

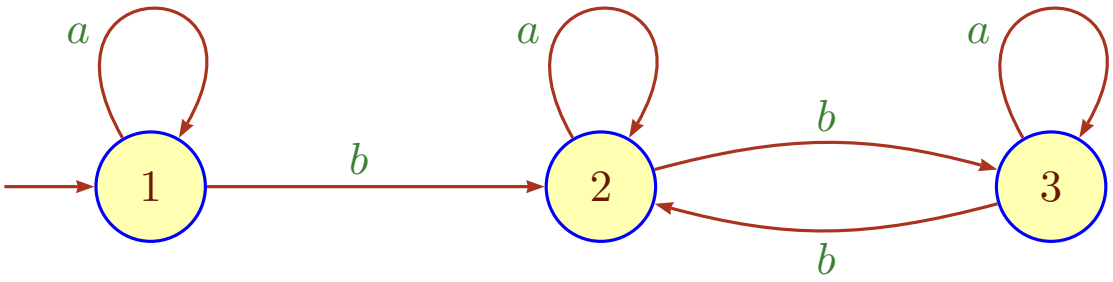
The at-least-one-but-finitely-many-b's language from above is accepted by the following Muller automaton.

The table has the form $\mathcal{F} = ((2), (3))$.



Note that this automaton distinguishes between an 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.

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



The complement table contains several useless entries:

F	$\emptyset$	1	$1,2$	$1,3$	$2,3$	$1,2,3$
$\mathcal{L}$	$\emptyset$	a^ω	$\emptyset$	$\emptyset$	$a^\star(ba^\star)^\omega$	$\emptyset$

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

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 $\mathcal{A}_1$ and $\mathcal{A}_2$ one can construct a Muller automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Similarly, there is a Muller automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Proof.

As usual, we build a product machine $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$. So the new state set is $Q = Q_1 \times Q_2$.

Write $\text{prj}_i : Q \to Q_i$ for the natural projections.

For union, the table of $\mathcal{A}$ has the form

$$\{\, F \subseteq Q_1 \times Q_2 \mid \mathsf{prj}_1(F) \in \mathcal{F}_1 \vee \mathsf{prj}_2(F) \in \mathcal{F}_2 \,\}$$

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

We could use de Morgan's law in conjunction with the previous construction to build a Muller automaton for the intersection. It is more interesting to construct the machine $\mathcal{A}$ directly.

This time, the table of $\mathcal{A}$ looks like

$$\{\, F \subseteq Q_1 \times Q_2 \mid \mathsf{prj}_1(F) \in \mathcal{F}_1 \wedge \mathsf{prj}_2(F) \in \mathcal{F}_2 \,\}$$

□

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 $U_i, V_i \subseteq \Sigma^\omega$ are recognizable by a deterministic Büchi automaton. In other words, L must lie in the Boolean algebra generated by deterministic Büchi languages.

Proof.

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

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

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 $\bigcup_{i \leq n} U_i - V_i$.

□

Note that the table will have exponential size.

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

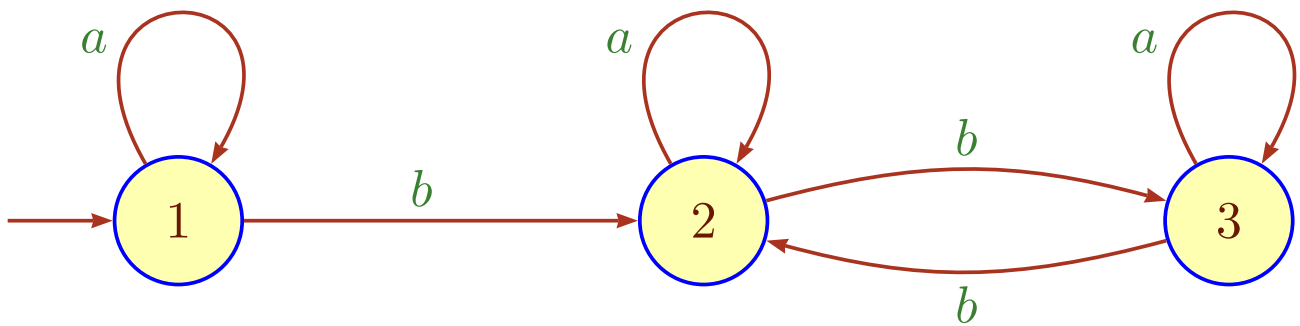
A Rabin automaton consists of a deterministic transition system $\langle Q, \Sigma, \tau \rangle$ and an acceptance condition $q_0 \in Q$ and $\mathcal{R} \subseteq \mathfrak{P}(Q) \times \mathfrak{P}(Q)$.
$\mathcal{A}$ accepts an infinite word $x \in \Sigma^\omega$ if there is a run π of $\mathcal{A}$ on x that starts at q_0 and such that for some $(L, R) \in \mathcal{R}$: $\mathrm{rec}(\pi) \cap L = \emptyset$ and $\mathrm{rec}(\pi) \cap R \neq \emptyset$.

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.

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.

The acceptance condition for all three has the form

- initial states I (a singleton for Muller and Rabin)
- a family $\mathcal{F} \subseteq \mathfrak{P}(Q)$ of permissible values for the recurrent state set of a run.

Note that we may safely assume that $\mathcal{F}$ contains only strongly connected sets.

For Büchi automata the family is trivial: $\mathcal{F} = \{F\}$ and thus a data structure of size $O(n)$.

For Muller automata it is explicitly specified and potentially large.

For Rabin automata the specification is implicit: all $X \subseteq Q$ such that $\exists\,(L, R) \in \mathcal{R}\,(X \cap L = \emptyset, X \cap R \neq \emptyset)$. Each Rabin pair is $O(n)$, but there may be exponentially many.

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.

1 Infinite Words

2 Deterministic Languages

3 Muller and Rabin Automata

4 **Digression: Algebra and FSM**

5 Determinization

Suppose we have some set X and a collection F of endofunctions on X.

Definition

(X, F) is a transformation semigroup or composition semigroup if F is closed under composition, and a transformation monoid if, in addition, F contains a unit element.

If you prefer, you can think of the semigroup F as acting on X on the left in the natural way:

$$f \cdot x = f(x)$$

This is for standard composition; if we use diagrammatic composition (which is more natural in connection with finite state machines), we get a right action.

To see the connection to finite state machines, note that we can think of the transition function of a DFA as a Σ-indexed list of functions from states to states:

$$\delta_a : Q \to Q$$
$$\delta_a(p) = \delta(p, a)$$

This turns the DFA into a Σ-algebra

$$\mathcal{A} = \langle Q; \delta_{a_1}, \ldots, \delta_{a_k} \rangle$$

This may seem like a pointless exercise, but it naturally leads to another interesting perspective: algebra.

But Why?

First off, nothing is lost: we can "iterate" these functions according to some input word $u = u_1 u_2 \dots u_n$ (diagrammatic composition):

$$\delta_u = \delta_{u_1} \circ \delta_{u_2} \circ \dots \delta_{u_{n-1}} \circ \delta_{u_n}$$

Acceptance then translates into: $\mathcal{A}$ accepts a word u iff $\delta_u(q_0) \in F$.

Plus, we get some additional concepts more or less for free: a subautomaton of $\mathcal{A}$ is another Σ-algebra $\mathcal{B} = \langle P; \gamma_{a_1}, \dots, \gamma_{a_k} \rangle$ such that $P \subseteq Q$ and $\gamma_a(p) = \delta_a(p)$.

Similarly, a morphism $\varphi : \mathcal{A} \to \mathcal{B}$ of Σ-algebras must be a map $\varphi : Q \to P$ such that

$$\varphi(\delta_a(p)) = \gamma_a(\varphi(p))$$

One may want to augment this by conditions about initial and final states.

It is also straightforward to define products of the form

$$\mathcal{A} \times \mathcal{B}$$

And we get congruences: an equivalence relation E on Q is a congruence if

$$p \; E \; q \quad \text{implies} \quad \delta_a(p) \; E \; \delta_a(q)$$

To be sure, all these concepts can be developed without any appeal to algebra, given enough thought.

But the whole point here is that they pop up for free, courtesy of some general, universal ideas.

Exercise

Figure out exactly what morphism, product and congruence mean in this context.

The functions δ_a, $a \in \Sigma$, generate a transformation semigroup (monoid) T over Q, a subsemigroup of the full monoid of endofunctions $Q \to Q$.

Definition

T is called the transformation semigroup (monoid) of the DFA.

One way of writing down T is

$$T = \{ \delta_x : Q \to Q \mid x \in \Sigma^\star \} = \langle \delta_a \mid a \in \Sigma \rangle$$

where $\delta_x(p) = \delta(p, x)$.

Analyzing this semigroup can help quite a bit in getting a better understanding of a DFA. And, there are powerful algebraic tools available that help in dealing with the monoid.

Example 1: Even-Even

There is a natural 4-state DFA that accepts all strings over $\{a, b\}^\star$ that contain an even number of a's and an even number of b's.

p	1	2	3	4
$\delta_a(p)$	2	1	4	3
$\delta_b(p)$	3	4	1	2

The initial state is 1 and $F = \{1\}$.

But note that

$$\delta_a \circ \delta_a = I$$
$$\delta_b \circ \delta_b = I$$
$$\delta_a \circ \delta_b = \delta_b \circ \delta_a$$

so that the transformation semigroup consists of $\{I, \delta_a, \delta_b, \delta_a \circ \delta_b\}$. Note that this is actually a monoid and even a group (Kleinsche Vierergruppe).

Moreover, from the equations it is easy to see that for any word x

$$\delta_x = I \iff \#_a x \text{ even}, \#_b x \text{ even}$$

Similarly we have

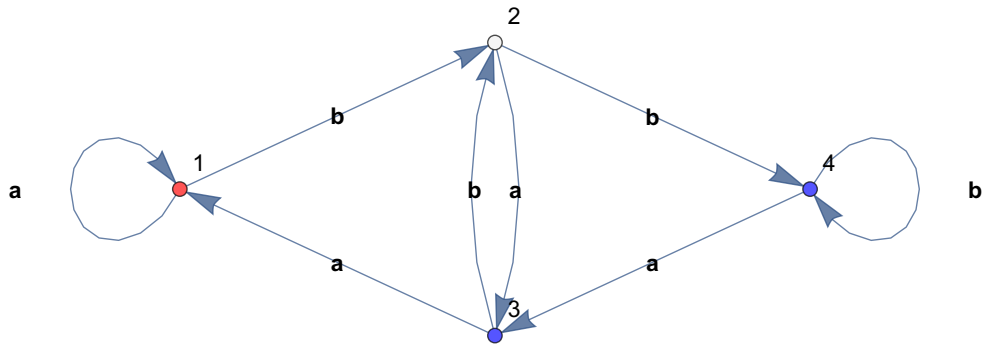
$$\delta_x = \delta_a \iff \#_a x \text{ odd}, \#_b x \text{ even}$$

and so on.

At the very least this very elegant and concise.

Example 2: $L_{a,-2}$

The usual de Bruijn automaton



yields the transformations

p	1	2	3	4
$\delta_a(p)$	1	3	1	3
$\delta_b(p)$	2	4	2	4

These generate the semigroup (no monoid here)

$$(1,3,1,3),(2,4,2,4),(1,1,1,1),(2,2,2,2),(3,3,3,3),(4,4,4,4)$$

Question: What do the constant functions mean?

The distinction between semigroups and monoids here is a bit of a technical nuisance, but there is no easy way to get rid of it.

At any rate, note that we can turn any semigroup $\mathcal{S}$ into a monoid $\mathcal{S}^1$ by simply adding a new element 1 and defining

$$x \cdot 1 = 1 \cdot x = x$$

for all x in $\mathcal{S}$.

Clearly, $\mathcal{S}$ and $\mathcal{S}^1$ are essentially the same.

Also note that a transformation semigroup may be a monoid without containing the identity function.

The reason monoids are important here is because they provide a characterization of regular languages that is free of any combinatorial aspect. Always remember: algebra is the great simplifier.

Theorem

A language $L \subseteq \Sigma^\star$ is regular iff there is a finite monoid M, $M_0 \subseteq M$ and a monoid homomorphism $f : \Sigma^\star \to M$ such that $L = f^{-1}(M_0)$.

Proof.

If L is regular, let M be the transformation monoid of a DFA that recognizes L, and define $f(x) = \delta_x$ and $M_0 = \{\, g \in M \mid g(q_0) \in F \,\}$.

The opposite direction is more interesting: we construct a DFA

$$\mathcal{A} = \langle M, \Sigma, \delta; 1_M, M_0 \rangle$$

where $\delta(p, a) = p \cdot f(a)$. Then $\delta(p, x) = p \cdot f(x)$ and $\delta(q_0, x) = f(x)$. □

Message: anything goes as a state set, as long as the set is finite. For the implementer, the state set is always $[n]$, but that's not a good way to think about it.

Algebraic automata theory is a fascinating subject with lots of elegant results, but it requires work and there is no essential algorithmic payoff. So, we won't go there.

1 Infinite Words

2 Deterministic Languages

3 Muller and Rabin Automata

4 Digression: Algebra and FSM

5 **Determinization**

Recall the algebraic approach to showing that the complement of a regular language $L \subseteq \Sigma^\star$ is again regular:

- Find a congruence of finite index $\equiv$ on $\Sigma^\star$,
- that saturates L: $L = \bigcup_{x \in L} [x]$.
- Then both L and $\Sigma^\star - L$ are finite unions of equivalence classes of $\equiv$.
- But these classes are all regular, done.

For example, we could use the syntactic congruence $x \equiv_L y \Leftrightarrow \forall\, u, v\, (uxv \in L \Leftrightarrow uyv \in L)$ (which can be expressed in terms of a DFA for L).

Or we could exploit the monoid that recognizes L.

And ω?

It is tempting to try the same approach for ω-languages. Incidentally, this is what Büchi did.

First off, our congruence must derive from nondeterministic machines.

This is slightly tricky, we need to involve the acceptance condition For Büchi automata. Here goes:

$$x \equiv y \Leftrightarrow \forall\, p, q\, (p \xrightarrow{x} q \Leftrightarrow p \xrightarrow{y} q) \wedge$$
$$\forall\, p, q\, (p \xrightarrow{x}_F q \Leftrightarrow p \xrightarrow{y}_F q)$$

where $p \xrightarrow{x}_F q$ means: there is a run from p to q with label x that touches a final state.

It is easy to see that $\equiv$

- is a congruence,
- has finite index,
- the languages $[u][v]^\omega$ saturate L,
- the languages $[u][v]^\omega$ are ω-regular.

Looks like we nailed it. Alas ...

We need to show that $\boldsymbol{x} \in \Sigma^\omega$ implies $\boldsymbol{x} \in [u][v]^\omega$ for some $u, v \in \Sigma^\star$.

Suppose we have a morphism $f : \Sigma^\star \to M$ into a finite monoid M.

Lemma

For $\boldsymbol{x} \in \Sigma^\omega$ there exists $u, e \in M$, e idempotent, such that

$$\boldsymbol{x} \in f^{-1}(u)(f^{-1}(e))^\omega$$

Proof. Define $m(i,j) = f(x_i \dots x_j) \in M$ for $i < j$, so we are coloring pairs of elements in $\mathbb{N}$ by M.

By Ramsey's celebrated theorem, we get an infinite homogeneous subset $I \subseteq \mathbb{N}$.

Set $u = m(0, i_1)$ and $e = m(i_1, i_2)$.

□

The pairs (u, e) in M are sometimes called linked pairs.

So we now have a characterization of ω-regular languages in terms of recognizability, entirely analogous to the finite word case.

There is a finite monoid M and a morphism $f : \Sigma^\star \to M$ and a set of linked pairs P such that

$$L = \bigcup_{(u,e)\in P} f^{-1}(u)\Big(f^{-1}(e) \Big)^\omega$$

We still need a (at least on occasion) practical algorithm to construct automata for complements. This was an open problem for some 25 years after Büchi's groundbreaking work.

Theorem (Safra 1988)

There is an algorithm to convert a Büchi automaton into an equivalent Rabin (or Muller) automaton.

The algorithm has running time

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

Unfortunately, this is optimal: there are examples where the deterministic automata are that large.

More next time.

Safra's algorithm is quite complicated, in all aspects:

- the code is messy if one tries to make it reasonably efficient,
- correctness is highly non-trivial, and
- the running time analysis (lower bounds) is a mess.

So, for the time being, let's ignore Safra's algorithm and just verify that Büchi, Muller and Rabin automata are reasonably well-behaved.

Lemma

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

Proof.

Consider a Rabin automaton $\langle Q, \Sigma, \tau; q_0, \mathcal{R} \rangle$. 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.

The opposite direction is harder.

Let $\langle Q, \Sigma, \delta; q_0, \mathcal{F}\rangle$ be a Muller automaton. For simplicity assume $\mathcal{F} = \{F\}$. Consider a new transition system on state set

$$Q' = \mathfrak{P}(F) \times Q$$

Let $q = \delta(p, a)$ and define transitions

$$(U, p) \xrightarrow{a} (U', q)$$

where $U' = \emptyset$ if $U = F$ and $F \cap (U \cup \{q\})$ otherwise. There is one Rabin pair:

$$L = \{\, (U, p) \mid p \notin F \,\} \qquad R = \{\, (U, p) \mid U = F \,\}$$

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

For a full table $\mathcal{F} = \{F_1, \ldots, F_k\}$ rinse and repeat. □

Consider a Muller automaton, again assume a singleton table, $\mathcal{F} = \{F\}$.

Here is a construction of an equivalent Büchi automaton that avoids powersets.

Let $F = \{q_1, \ldots, q_{n-1}\}$.

We construct a nondeterministic Büchi automaton on states $Q' = Q \cup (F \times \underline{n})$ where $\underline{n} = \{0, 1, \ldots, n-1\}$.

Let $q = \delta(p, a)$ and define τ in two parts:

$$\tau(p, a) = \begin{cases} \{q\} & \text{if } q \notin F, \\ \{q, (q, 0)\}, & \text{otherwise.} \end{cases}$$

$$\tau((p, k), a) = \begin{cases} \{(q, k), (q, k + 1 \bmod n)\} & \text{if } p = q_k,\ k > 0, \\ \{(q, 1)\} & \text{if } k = 0. \end{cases}$$

$F' = Q \times \{0\}$.

Automata constructions over finite words are typically fairly natural, there is usually one good answer.

This is no longer true for ω-automata, there may well be different reasonable ways to construct machines for a particular purpose.

Also, correctness proofs become much, much harder.

Exercise

Prove that the last construction really works as advertised.

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

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

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

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

Done. □

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.