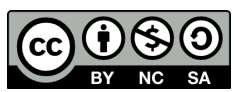
CDM

Safra's Algorithm

Klaus Sutner

Carnegie Mellon University

Fall 2024



1 **Determinizing Büchi Automata**

2 **Safra's Algorithm**

3 **Presburger Arithmetic**

Where Are We?

We have a notion of "regular" (recognizable, rational) ω-language accepted by a nondeterministic Büchi automaton.

We know that deterministic Büchi automata are strictly weaker.

And we have deterministic Muller and Rabin automata that might be expressive enough to deal with complementation.

What is needed is a practical algorithm that converts a nondeterministic Büchi automaton into a, say, Rabin automaton (Muller is just as well).

Note the trade-off: the Rabin automaton must be deterministic, but it has a more flexible acceptance condition.

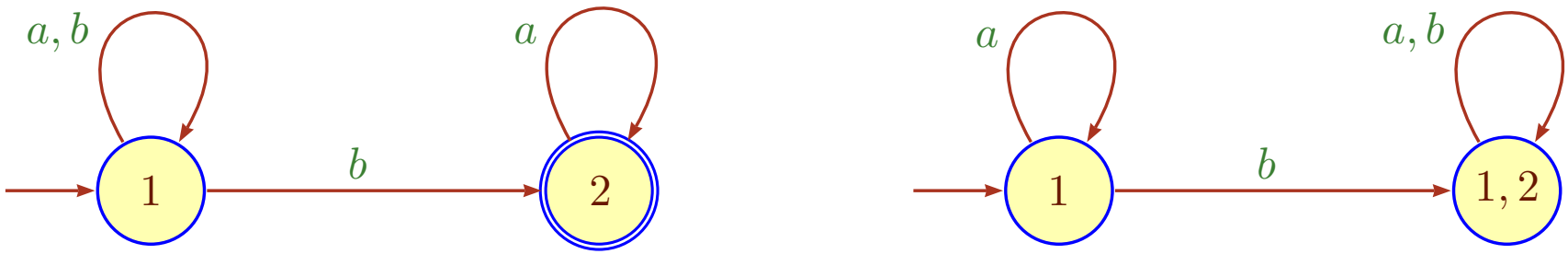
Theorem

For every Büchi automaton there exists an equivalent Rabin automaton. Hence the recognizable ω-languages are effectively closed under complementation.

Wurzelbrunft would think that this theorem is similar to the old Rabin/Scott result (powerset automaton construction). Alas, things don't work out: we are not keeping track of individual computations (a tree of unbounded width), only of reachable sets of states. That introduces spurious computations in the infinite case.

No Luck

For example, for our standard $0 < \#_b < \infty$ example 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 we need a bigger hammer than Rabin-Scott.

In 1960, Büchi already had a construction that showed that ω-regular languages are closed under complementation.

Unfortunately, his argument is based on Ramsey theory and produces a horrible upper bound of

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

The method is not competitive (though related and faster methods seem to be useful in the context of universality testing)

A better solution was given in 1966 by McNaughton, who showed how to convert a Büchi automaton into a (deterministic) Muller automaton by dealing with components of the form KL^ω.

Theorem (McNaughton 1966)

For every Büchi automaton there is an equivalent Muller automaton.

Alas, the construction is still complicated and difficult to implement.

There is also a purely algebraic proof due to Le Saëc, Pin and Weil (using ω-semigroups) that shows that the complement of a recognizable language is again recognizable.

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:

$$I \xrightarrow{u} P_u \xrightarrow{v} P_{uv} \xrightarrow{w} P_{uvw} \longrightarrow \qquad \ldots$$

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 $xyz\ldots$: 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.

$$\begin{array}{ccccccccc} 1 & \xrightarrow{b} & 1 & \xrightarrow{b} & 1 & \xrightarrow{b} & 1 & \xrightarrow{b} \\ & \searrow & & \searrow & & \searrow & & \searrow \\ & & 2 & & 2 & & 2 & \end{array}$$

Our power automaton accepts too much.

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

$$P'_u \xrightarrow{v} P'_{uv}$$

implies that every state $p \in P'_{uv}$ is the target of a run of $\mathcal{A}$ starting at a state $q \in P'_u$ 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 this trick to make things work properly. .

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: $v \in V = \{1, 2, \ldots, 2n\}$.
- 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 that will be explained later. The root is always named 1. Only leaves can be marked. Since names are unique, we will occasionally confuse them with nodes.

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

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

(S1) $\bigcup_{u \operatorname{par} v} \lambda(v) \subsetneq \lambda(u)$

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

Here $u \operatorname{par} v$ means that u is the parent of v. Thus if $v_1, v_2, \ldots, v_k$ 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. □

Proposition

A Safra tree has at most n nodes.

Proof. For every node v define

$$f(v) :\in \lambda(v) - \bigcup \lambda(u_i) \neq \emptyset$$

Thus $f(v)$ is a state that is present at v, but missing from the subtrees of v.

It is easy to see that f is injective.

□

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.

1 Determinizing Büchi Automata

2 **Safra's Algorithm**

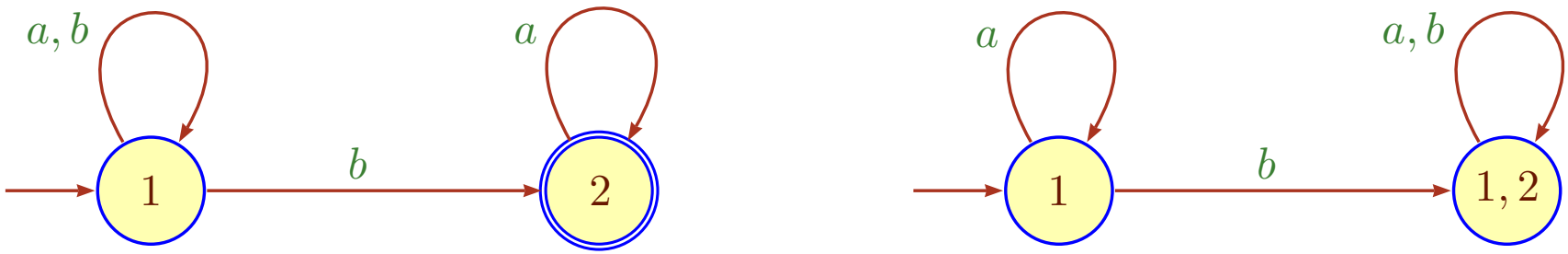
3 Presburger Arithmetic

We would like a (somewhat) practical algorithm that converts a Büchi automaton into a Rabin (or Muller) automaton. Note the trade-off: the Rabin automaton must be deterministic, but it has a more flexible acceptance condition.

Wurzelbrunft would think that determinization of Büchi automata must be similar to the old Rabin/Scott result, the powerset automaton construction.

Alas, things don't work out, at least not without hugely more effort.

For example, for our standard $0 < \#_b < \infty$ example 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 we need a bigger hammer than Rabin-Scott.

In 1960, Büchi already had a construction that showed that ω-regular languages are closed under complementation.

Unfortunately, his argument is based on Ramsey theory and produces a horrible upper bound of

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

The method is not competitive (though related and faster methods seem to be useful in the context of universality testing)

A better solution was given in 1966 by McNaughton, who showed how to convert a Büchi automaton into a (deterministic) Muller automaton by dealing with components of the form KL^ω.

Theorem (McNaughton 1966)

For every Büchi automaton there is an equivalent Muller automaton.

Alas, the construction is still complicated and difficult to implement.

There is also a purely algebraic proof due to Le Saëc, Pin and Weil (using ω-semigroups) that shows that the complement of a recognizable language is again recognizable.

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:

$$I \xrightarrow{u} P_u \xrightarrow{v} P_{uv} \xrightarrow{w} P_{uvw} \longrightarrow \qquad \ldots$$

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 $xyz\ldots$: 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.

$$\begin{array}{ccccccccc} 1 & \xrightarrow{b} & 1 & \xrightarrow{b} & 1 & \xrightarrow{b} & 1 & \xrightarrow{b} \\ & \searrow & & \searrow & & \searrow & & \searrow \\ & & 2 & & 2 & & 2 & \end{array}$$

Our power automaton accepts too much.

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

$$P'_u \xrightarrow{v} P'_{uv}$$

implies that every state $p \in P'_{uv}$ is the target of a run of $\mathcal{A}$ starting at a state $q \in P'_u$ 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 this trick to make things work properly. .

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: $v \in V = \{1, 2, \ldots, 2n\}$.
- 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 that will be explained later. The root is always named 1. Only leaves can be marked. Since names are unique, we will occasionally confuse them with nodes.

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

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

(S1) $\bigcup_{u \,\mathrm{par}\, v} \lambda(v) \subsetneq \lambda(u)$

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

Here $u \,\mathrm{par}\, v$ means that u is the parent of v. Thus if $v_1, v_2, \ldots, v_k$ 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. □

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.

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 $\mathcal{B}$.

For each Safra tree T and letter $a \in \Sigma$, we will explain in a moment how to construct a new Safra tree $\delta_a(T)$.

We will use a feeble list notation to indicate the nodes in a Safra tree, without actually writing down the parent relation. Since our example trees are microscopic, this actually works fine.

The initial tree T_0 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 2 is marked).

1. Unmark
   Unmark all the nodes in the tree.

2. Update
   Replace $\lambda(v)$ by $\tau(\lambda(v), a)$ everywhere.

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.

Let T be an ordered tree (children are ordered left-to-right).

A node v in T is to the left of node u if there is a subtree T' of T such that T' has root r and children $r_1, r_2, \ldots, r_k$ and there is $1 \leq i < j \leq k$ such that v is in the subtree with root r_i and u is in the subtree with root r_j.

Exercise

Figure out a fast way of performing the Horizontal Merge in a Safra tree.

- Pre-processing: removing marks is just a simple warm-up.
- Main steps: Update and Create is where the real action is. In general, they will destroy the Safra properties of the tree.
- Post-processing: the remaining steps reestablish them. After Horizontal Merge (S2) holds. After Kill Empty, all label sets are non-empty. After Vertical Merge condition (S1) also holds.

Exercise

Check in detail that the new tree is Safra.

In the Create step, new names 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 and names are chosen in $V = [2n]$.

Also, we will traverse the tree in top-down, left-to-right order.

Warning: The node names 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.

Similarly, in Horizontal Merge, we arbitrarily have adopted the convention to move from left to right (top-down is not an issue here).

Exercise

Figure out what would happen in the following examples if we changed any of these conventions.

The 6-step procedure defines (somewhat complicated) functions

$$\delta_a : \text{Safra trees} \longrightarrow \text{Safra trees}$$

for each $a \in \Sigma$.

The Rabin machine $\mathcal{A}$ is now simply defined as follows:

> Run the vanilla closure algorithm starting at tree T_0 and with operations δ_a, $a \in \Sigma$.

This produces a finite collection of Safra trees as state set Q' of $\mathcal{A}$, plus the transition function of $\mathcal{A}$ (the usual Cayley graph argument).

Of course, $T_0 \in Q'$ is the initial state.

It remains to determine the Rabin pairs of $\mathcal{A}$.

The pairs are (L, R) where v is (the name of) some node and

$$L = \{\, T \in Q' \mid v \notin T \,\} \qquad R = \{\, T \in Q' \mid v \in T, \text{marked} \,\}$$

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

That's all.

Lame Example

Suppose the Büchi automaton has $Q = F$.

Then Safra's algorithm degenerates into the ordinary Rabin-Scott powerset construction: all the trees have exactly one node, the root.

This is reassuring, since any infinite run is accepting in this case and the existence of an infinite run (without any additional conditions) can be tested by the power automaton.

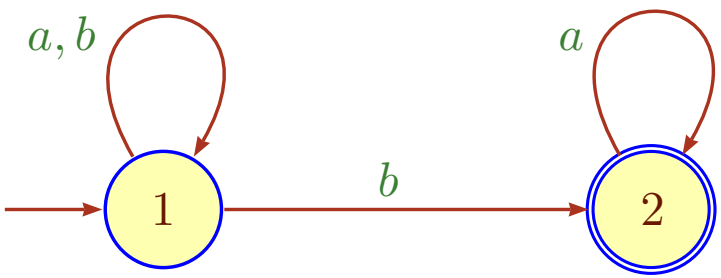
Exercise

Make sure you understand how and why this works.

Let's return to the old workhorse 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. A Büchi automaton $\mathcal{B}$ for L looks like so:



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

$$
\begin{array}{lll}
1 & (1{:}1) & \stackrel{a}{\Rightarrow} \quad (1{:}1) \\
1 & (1{:}1) & \stackrel{b}{\Rightarrow} \quad (1{:}1, 2; 2{:}2!) \\
2 & (1{:}1, 2; 2{:}2!) & \stackrel{a}{\Rightarrow} \quad (1{:}1, 2; 2{:}2!) \\
2 & (1{:}1, 2; 2{:}2!) & \stackrel{b}{\Rightarrow} \quad (1{:}1, 2; 3{:}2!) \\
3 & (1{:}1, 2; 3{:}2!) & \stackrel{a}{\Rightarrow} \quad (1{:}1, 2; 3{:}2!) \\
3 & (1{:}1, 2; 3{:}2!) & \stackrel{b}{\Rightarrow} \quad (1{:}1, 2; 2{:}2!)
\end{array}
$$

The Rabin pairs are

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

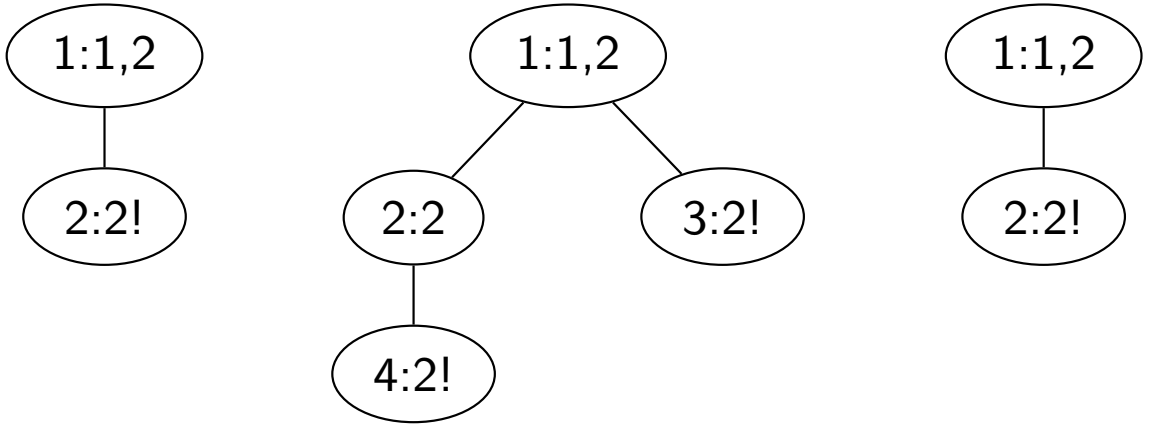
since 2 and 3 are the only marked nodes.

$\delta_b(T_1) = T_2$

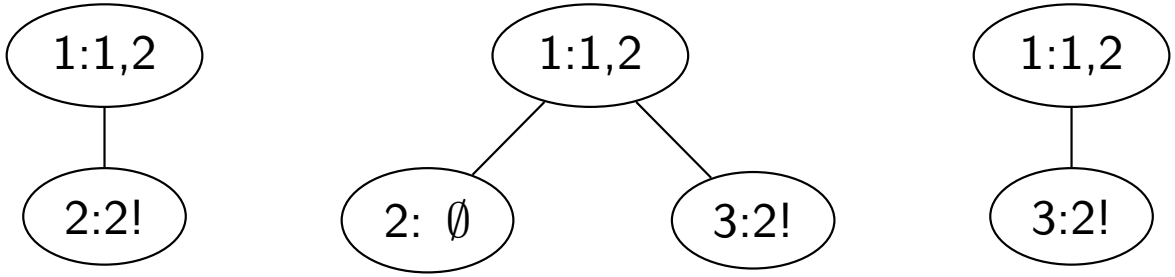
$T_1 = (1{:}1)$, $T_2 = (1{:}1, 2; 2{:}2!)$

1. Unmark: zip
2. Update: $(1{:}1, 2)$
3. Create: $(1{:}1, 2; 2{:}2!)$
4. Horizontal Merge: zip
5. Kill Empty: zip
6. Vertical Merge: zip

$\delta_a(T_2) = T_2$

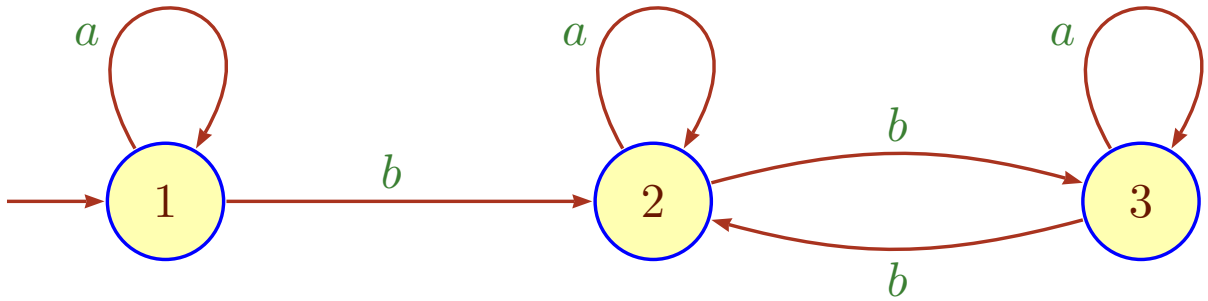


$\delta_b(T_2) = T_3$



A detailed description of the computation of the transitions with source state 2. The intermediate tree after Update and Create are shown.

The diagram should look familiar:

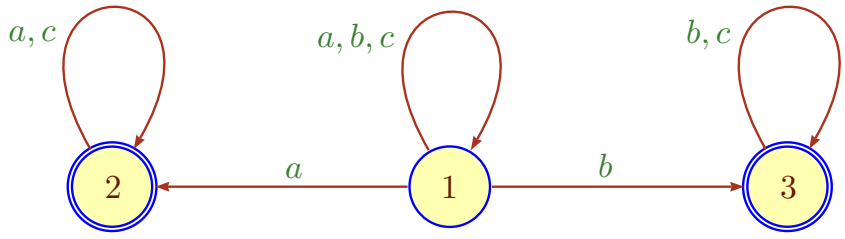


This is the machine we already saw previously.

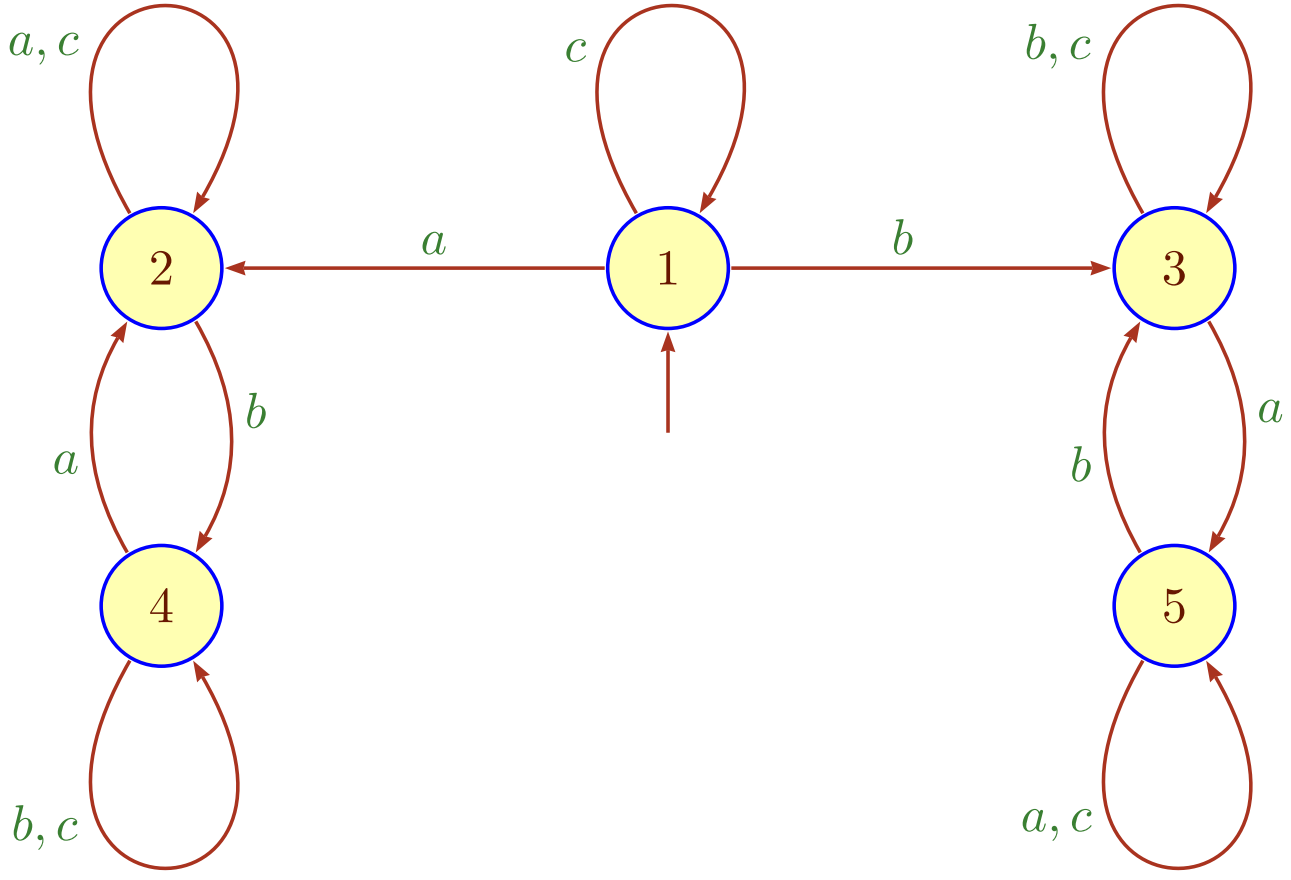
In this particular case, we have already verified that the machine behaves properly.

Example II

We determinize the Büchi automaton for $\#_a < \infty \vee \#_b < \infty$.

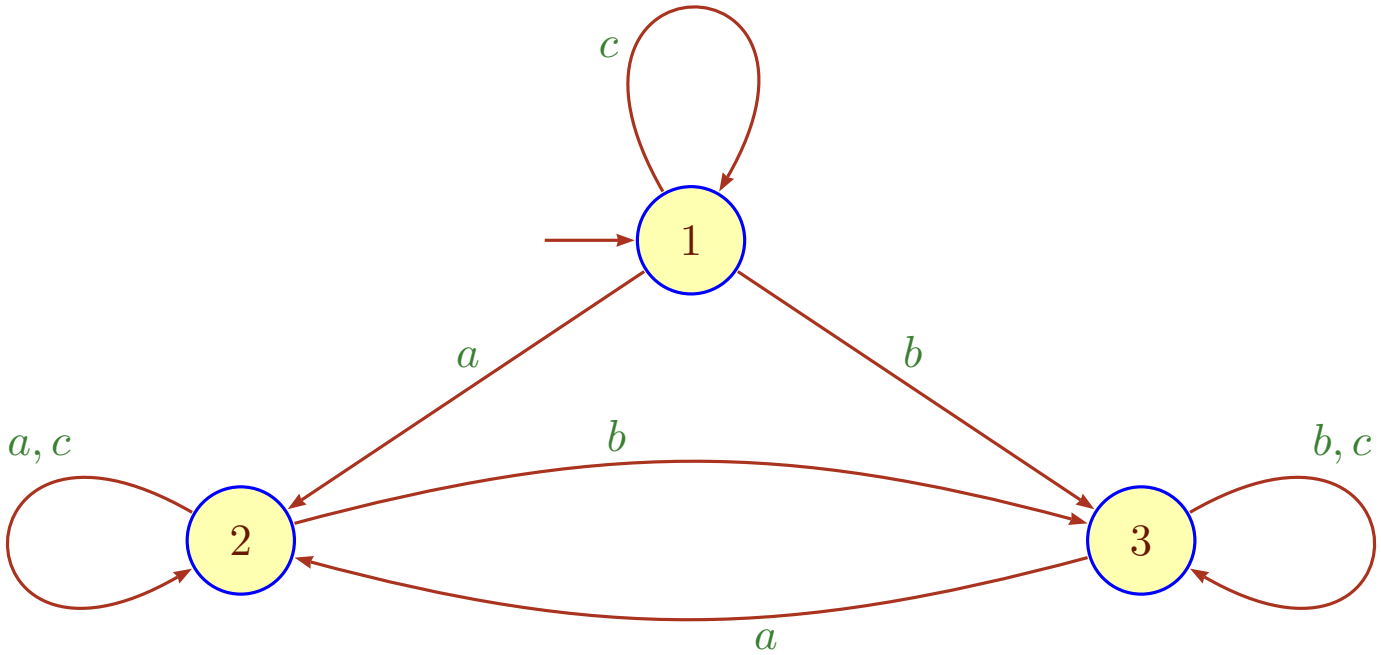


$$
\begin{array}{lll}
1 & (1{:}1) & \stackrel{a}{\Rightarrow} (1{:}1, 2; 2{:}2!) \\
1 & (1{:}1) & \stackrel{b}{\Rightarrow} (1{:}1, 3; 2{:}3!) \\
2 & (1{:}1, 2; 2{:}2!) & \stackrel{a}{\Rightarrow} (1{:}1, 2; 2{:}2!) \\
2 & (1{:}1, 2; 2{:}2!) & \stackrel{b}{\Rightarrow} (1{:}1, 3; 3{:}3!) \\
3 & (1{:}1, 3; 2{:}3!) & \stackrel{a}{\Rightarrow} (1{:}1, 2; 2{:}2!) \\
3 & (1{:}1, 3; 2{:}3!) & \stackrel{b}{\Rightarrow} (1{:}1, 2; 2{:}2!) \\
4 & (1{:}1, 3; 3{:}3!) & \stackrel{a}{\Rightarrow} (1{:}1, 2; 2{:}2!) \\
4 & (1{:}1, 3; 3{:}3!) & \stackrel{b}{\Rightarrow} (1{:}1, 3; 3{:}3!) \\
5 & (1{:}1, 2; 3{:}2!) & \stackrel{a}{\Rightarrow} (1{:}1, 2; 3{:}2!) \\
5 & (1{:}1, 2; 3{:}2!) & \stackrel{b}{\Rightarrow} (1{:}1, 3; 2{:}3!)
\end{array}
$$



Rabin pairs $((1, 4, 5; 2, 3), (1, 2, 3; 4, 5))$.

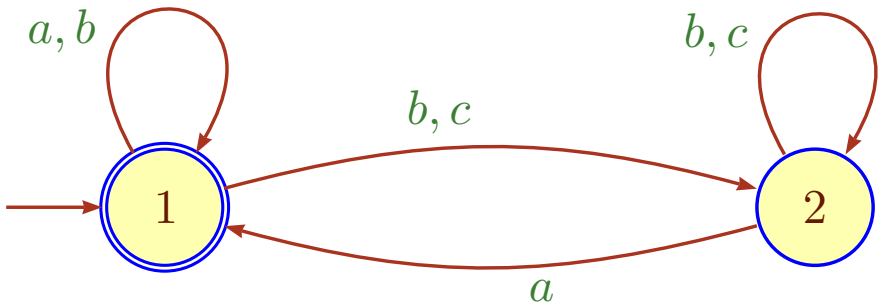
We can do a bit of state merging (but note this is tricky):



Rabin pairs $((1, 2; 3), (1, 3; 2))$.

Here is another Büchi automaton $\mathcal{B}$ on alphabet $\{a, b, c\}$.

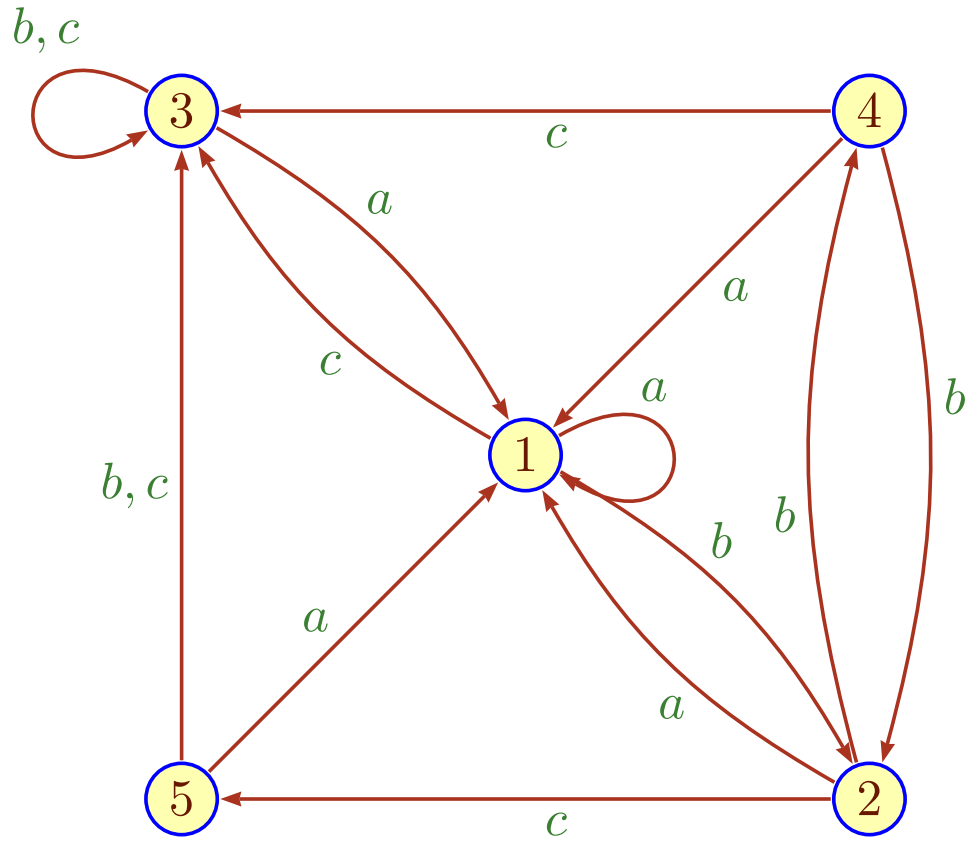
This one is slightly more complicated.



The language is given be the rational expression

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

Example III, contd.



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

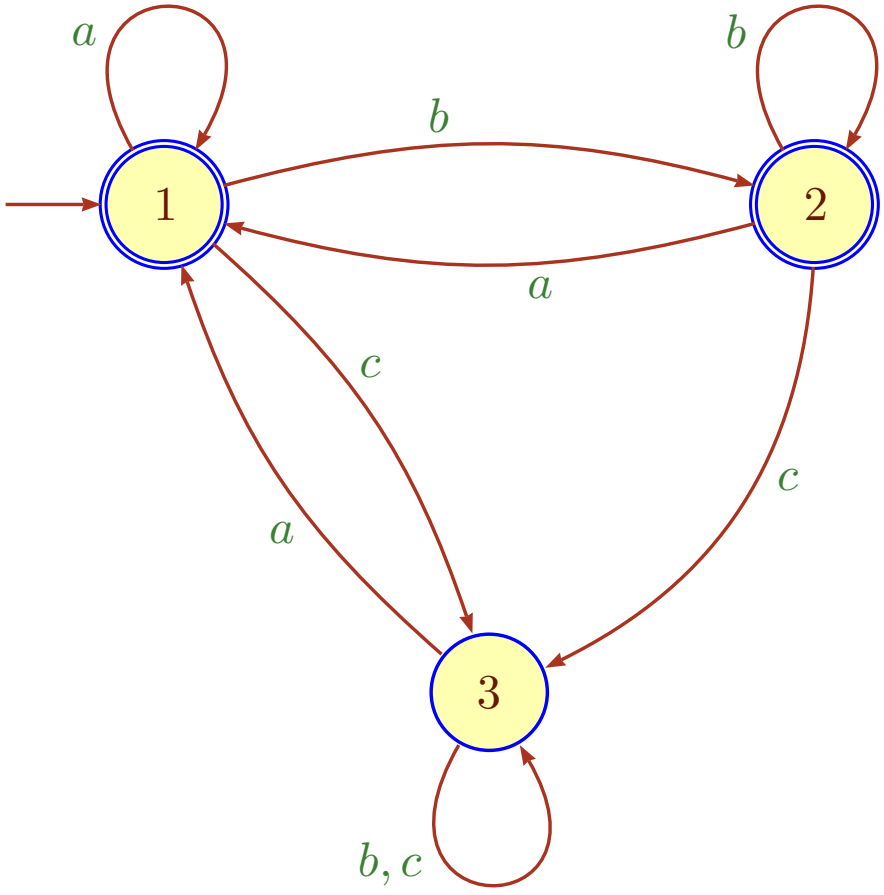
The Safra trees corresponding to the 5 states are

$$\begin{array}{ll} 1 & (1{:}1!) \\ 2 & (1{:}1, 2; 2{:}1!) \\ 3 & (1{:}2) \\ 4 & (1{:}1, 2!) \\ 5 & (1{:}2!) \end{array}$$

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, $F = \{1, 2\}$.

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 $x = x_1 x_2 \dots x_k$ 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.

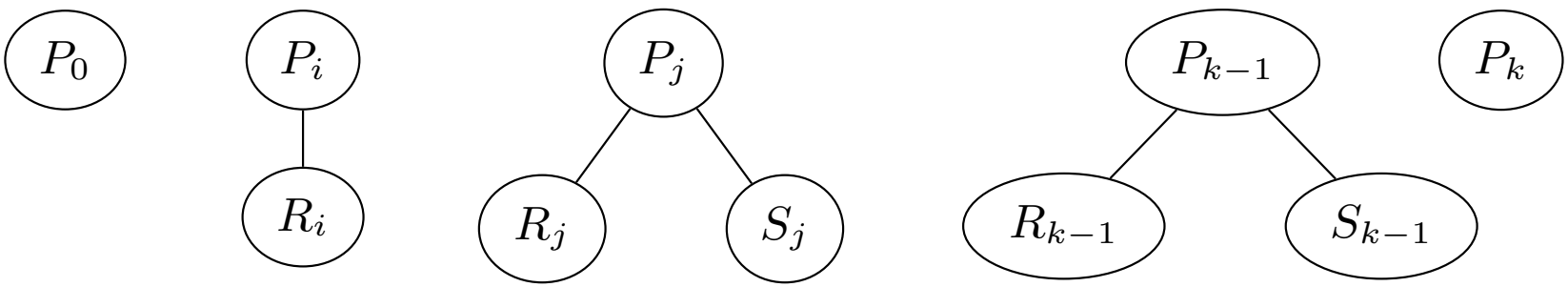
Sketch of proof.

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 P_k was obtained by a Vertical Merge, and any run from P_0 to P_k passes through a final state: $R_i = P_i \cap F$ and $S_j \subseteq P_j \cap F$.

□

Recall: Kőnig's Lemma

Theorem

Any infinite, finitely-branching tree must have an infinite branch.

Proof.

Start with r_0, the root. Since the tree is finitely-branching, one of the children of the root must span an infinite subtree. Let r_1 be one of these fat children.

Done by induction. □

We can think of the lemma as a weak choice principle. This is more powerful than plain Peano arithmetic (which suffices for ordinary finite state machines).

Note that the construction is non-constructive: if the tree were, say, computable, we would not know how to actually determine r_1.

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 I 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 Kőnig's lemma it must contain an infinite branch – which branch corresponds to an accepting computation of $\mathcal{B}$ on x.

For the opposite direction, let $\mathcal{B}$ accept x and let π be a corresponding run; say, p is a final state that appears infinitely often in π. Then the corresponding 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 p appears infinitely often in π, it must appear in some 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.

□

Lemma

Recognizable ω-languages of are closed under complementation. Hence they form an effective Boolean algebra.

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 and convert back to Büchi. □

Effective just means: there is an algorithm, efficiency is another matter. As it turns out, things a re much worse than the simple exponential blow-up for complementation of ordinary regular languages.

Exercise

Modify Safra's algorithm so that it produces directly a Muller automaton.

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

Note the horrendous upper bound for complementation. Alas, there are fairly simple, albeit entirely artificial examples that realize this bound, see below.

Thus any decision algorithm based on ω-automata is likely going to encounter substantial efficiency problems.

Safra's algorithm has the vexing property there are several reasonable versions that differ slightly in their behavior, but are all covered by essentially the same correctness proof.

- Transitions first: apply the transition function to the state sets before branching.
- Branch first: first handle children, then update the state set labels.
- Lazy names: try to minimize the number of names used.

The standard choice of name for a new node during the computation of $T' = \delta_a(T)$ is the least available one:

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

Here is a "better" approach: during the construction of the new tree T', first assign a symbolic name. Upon completion, try to assign actual names in such a way that T' has already been encountered earlier on.

Exercise

Implement this algorithm so that it beats the standard one, at least on occasion.

Let $\Sigma_k = \{0, 1, 2, \ldots, k\}$ and let $\boldsymbol{a} = a_0, a_1, \ldots$ be an infinite word over Σ_k. Think of two consecutive letters ab as an edge $a \to b$. $\boldsymbol{a}$ has an infinite path if there is an infinite subword

$$a_{n_0}, a_{n_0+1},\ a_{n_1}, a_{n_1+1},\ a_{n_2}, \ldots$$

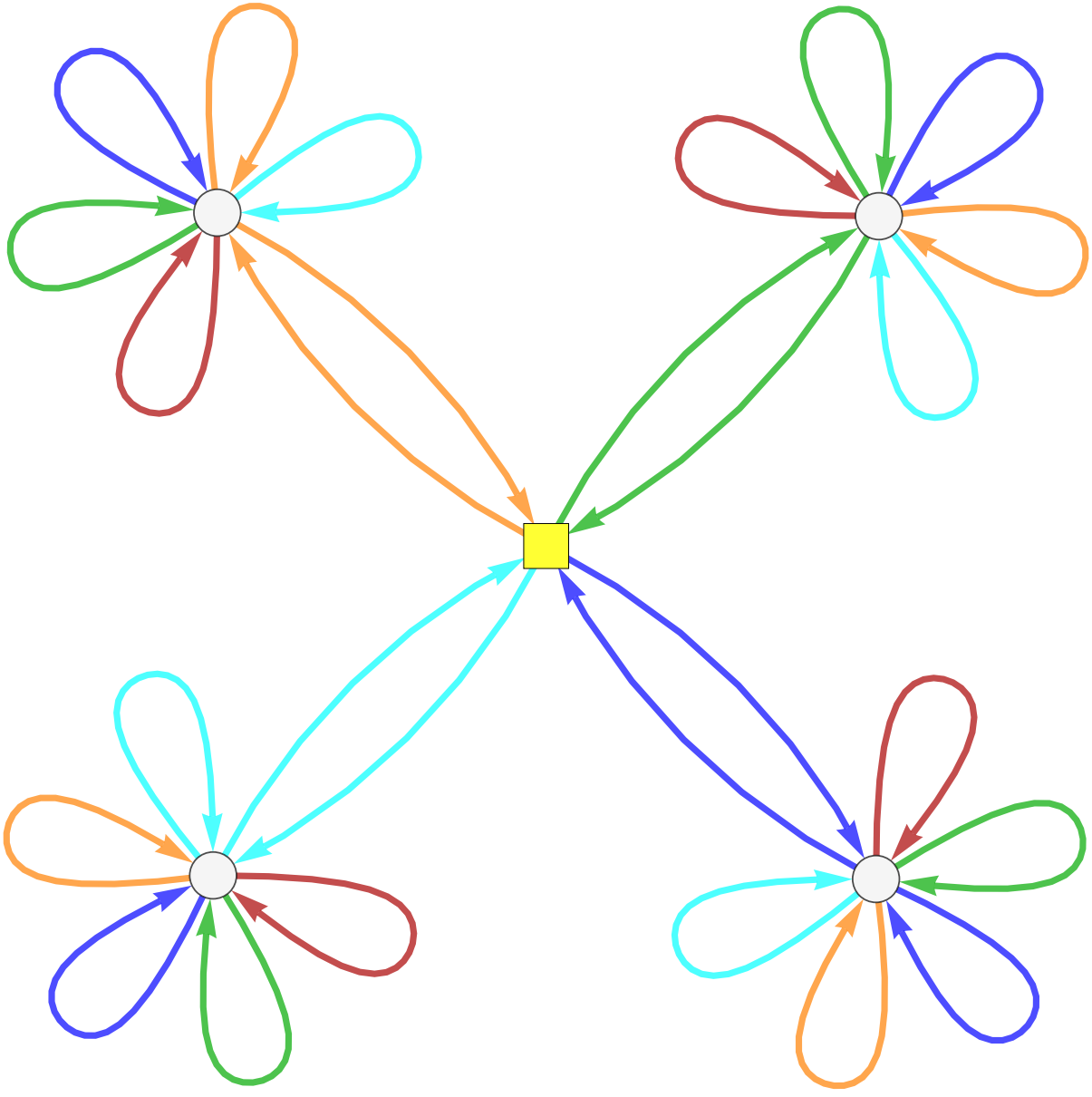
where $1 \leq a_{n_i+1} = a_{n_{i+1}} \leq k$.

Example: $\ldots ab \ldots bc \ldots cd \ldots de \ldots$ These letters will not all be distinct.

Let $L_k = \{\, \boldsymbol{a} \in {\Sigma_k}^\omega \mid \boldsymbol{a} \text{ has an infinite path} \,\}$.

Lemma (Michel 1988)

- *L_k has a nondeterministic Büchi automaton of size $k+1$.*
- *Every Büchi automaton for $\Sigma^\omega - L_k$ requires at least $k!$ states.*



1 Determinizing Büchi Automata

2 Safra's Algorithm

3 **Presburger Arithmetic**

We have the counterpart of "NFA Emptiness" for infinite words.

Problem:	**Büchi Emptiness**
Instance:	A Büchi automaton $\mathcal{A}$.
Question:	Is $\mathcal{L}^\omega(\mathcal{A})$ empty?

This is easily decidable: there has to be a path from an initial state to a final state p such that p lies in a non-trivial strongly connected component of the diagram of $\mathcal{A}$.

Exercise

What is the complexity of Büchi Emptiness? Explore Emptiness tests for Muller and Rabin automata.

Theorem (Büchi 1960)

For every sentence φ of $\mathrm{MSO}[<]$, one can effectively construct a Büchi automaton $\mathcal{A}_\varphi$ whose ω-acceptance language is the collection of all words w that satisfy φ:

$$\mathcal{L}^\omega(\mathcal{A}) = \{\, w \in \Sigma^\omega \mid w \models \varphi \,\}.$$

Corollary

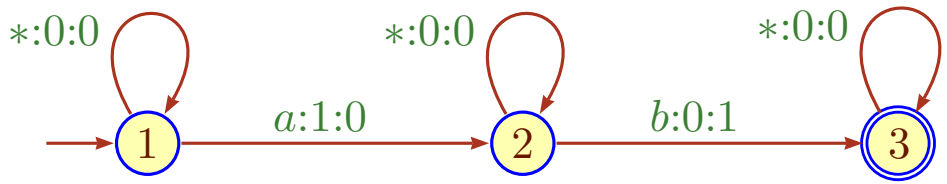
$\mathrm{MSO}[<]$ *is decidable over Σ^ω.*

To check whether a sentence φ is valid we only need to test whether $\mathcal{A}_\varphi$ is universal. Equivalently, we can check whether $\mathcal{A}_{\neg\varphi} = \overline{\mathcal{A}_\varphi}$ is non-empty.

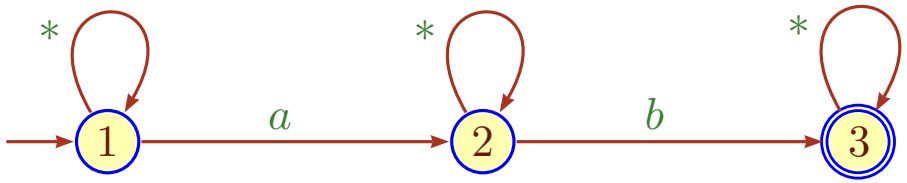
As it turns out, it may be algorithmically advantageous to use a universality testing algorithm and not deal with the last negation in the standard way.

Trivial Example

Here is a trivial example: $\exists\, x, y \left(x < y \wedge Q_a(x) \wedge Q_b(y) \right)$.



Projecting away the first-order variable tracks produces



The last automaton accepts the language $\Sigma^\star a \Sigma^\star b \Sigma^\omega$.

Weak monadic second order, WMSO, logic is define like MSO, except that we quantify over **finite** subsets of the domain. For example,

$$\forall\, X\, (\exists\, u\, X(u) \Rightarrow \varphi(X))$$

means that, for any non-empty finite set of positions P, $\varphi(P)$ holds. So if $\varphi(X)$ is

$$\exists\, u\, (X(u) \wedge \forall\, v\, (X(v) \Rightarrow v \leq u))$$

we get a valid formula (which is invalid in full second-order).

As already mentioned, Büchi automata can easily deal with the additional finiteness condition for the second-order tracks. Even better, the construction is essentially the same as in the full second-order case.

It follows that weak monadic second order logic (with $<$) is also decidable, using essentially the same algorithm.

One might wonder why Büchi's theorem is important outside of pure theory.

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.

Ordinary arithmetic is the study of the structure

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

Alas, even Σ_1 statements of the form

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

are already undecidable in general over $\mathfrak{N}$ (where φ has only bounded quantifiers): we can express Diophantine equations this way.

And truth of all of first-order logic over $\mathfrak{N}$ is highly undecidable.

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 and there is some hope that a decision algorithm might exist.

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 slightly non-trivial example of a Presburger formula:

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

Is it valid?

Without multiplication, arithmetic is much less complicated.

Theorem (M. Presburger 1929)

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

This result seemed like a major boost to Hilbert's program: first-order logic is sound and complete, and it can handle an interesting fragment of arithmetic.

Of course, what was really needed is a similar result for all of arithmetic. Alas ...

In 1929, Presburger showed that Peano arithmetic without multiplication (Presburger arithmetic) is decidable.

In 1930, Skolem proved that Peano arithmetic without addition (Skolem arithmetic) is decidable.

In 1931, Gödel showed that full Peano arithmetic is incomplete by expressing computation in the system. As a consequence, Peano arithmetic is undecidable.

Logicians have studied lots of other, related structures.

There are at least three ways to tackle this problem.

- Quantifier elimination
- Automaticity and ordinary finite state machines
- Monadic second-order logic and ω-automata

Presburger's original algorithm is based on quantifier elimination. Büchi developed the MSO approach and automaticity essentially dates back to Nerode in the 1990s.

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

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

$\mathrm{WMSO}[<]$ 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 using an ω-automaton: a machine that accepts three track binary words of the form

$$0^i 1 0^\omega : 0^j 1 0^\omega : 0^{i+j} 1 0^\omega$$

But this is impossible by a standard pumping argument.

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

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

Thus, X is essentially just the standard reverse binary expansion.

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

X	1	0	1	1	1	0	1	0	0	0	$\ldots$
Y	0	0	1	0	1	0	0	0	0	0	$\ldots$
Z	1	0	0	0	1	1	1	0	0	0	$\ldots$

This is really the same argument that shows that addition in reverse binary is synchronous.

Similarly we can check $\mathrm{val}(X) < \mathrm{val}(Y)$ and so forth.

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.