

CDM

Determinizing Büchi Automata

Klaus Sutner

Carnegie Mellon University

Fall 2011

Outline

- 1 Complementation
- 2 Determinizing Büchi Automata
- 3 Safra's Algorithm

Where Are We?

We have seen three different types of machines accepting regular ω -languages (Büchi, Muller and Rabin) and various conversion algorithms.

We also know that regular ω -languages are closed under union and intersection.

Two parts are still missing:

- closure under complementation, and
- conversion from Büchi to Rabin/Muller.

Complementation Closure

Of course, it might be the case that the complement of a regular ω -language is not regular in general.

Theorem (McNaughton 1966)

Regular ω -languages are closed under complementation.

McNaughton's original proof uses Muller automata; there is also a purely algebraic, automata-free proof. Unfortunately, neither argument translates into a good algorithm.

So, we still need a reasonably efficient method to convert from a nondeterministic Büchi automaton to a deterministic Rabin or Muller automaton.

- Complementation
- ② Determinizing Büchi Automata
 - Safra's Algorithm

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 have a deterministic transition system, but it has a more expressive acceptance condition.

Theorem

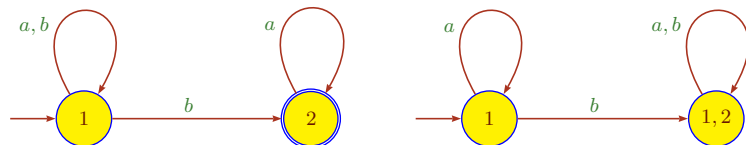
For every Büchi automaton there exists an equivalent Rabin automaton.

It is worth to think a bit about why this is difficult.

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 “at least one but finitely 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 .

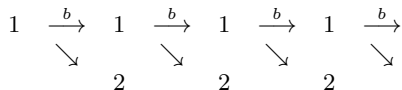
It seems that we need a construction more powerful than Rabin-Scott.

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:

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

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

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.

- Complementation
- Determinizing Büchi Automata
- ③ Safra's Algorithm

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 a name and

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

Of course, we only need to consider names whose nodes are marked in at least one tree.

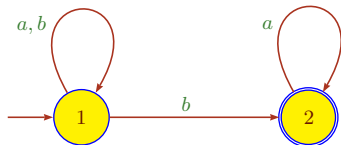
That's all.

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

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

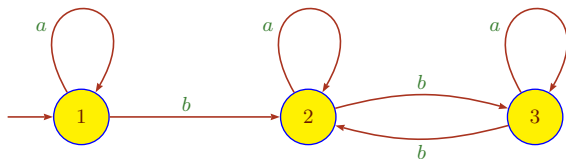
The Rabin pairs are

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

since 2 and 3 are the only marked nodes.

The Diagram

The diagram should look familiar:

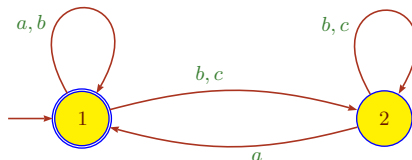


This is the machine we already saw last time.

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

Example II

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

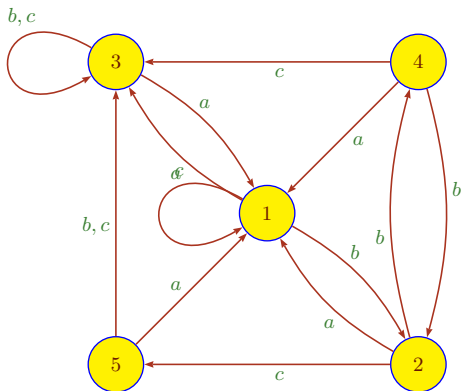


The language is

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

Example II, contd.

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



Comments

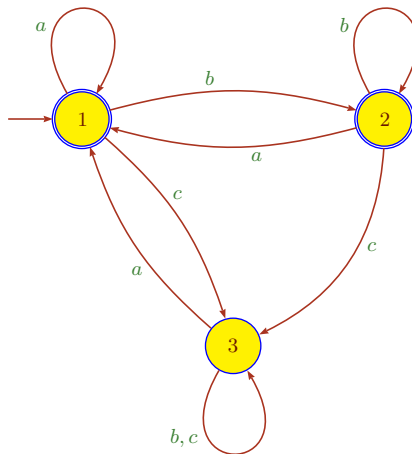
The Safra trees corresponding to the 5 states are

- 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.

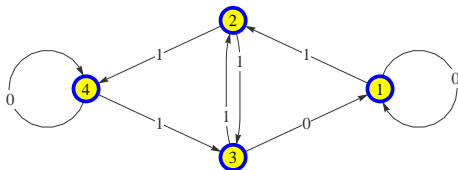
After Merging



As a Büchi automaton $F = \{1, 2\}$.

Example III

Here is a de Bruijn automaton (which describes a cellular automaton, but that doesn't matter here). If we construe one of these as a Büchi automaton, what is the corresponding Rabin automaton going to look like?



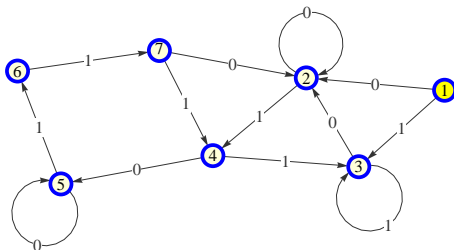
So there are 4 states, all initial and final.

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

Think, Don't Just 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 the famous elementary cellular automaton 110 (computationally universal) the algorithm produces the following machine:



States

The 8 states (labels of the root) are:

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

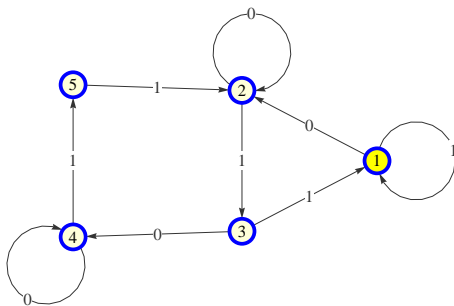
State number 1 (corresponding to $\{1, 2, 3, 4\}$ is initial), everybody except for the sink 7 (corresponding to \emptyset) is final.

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.

Incidentally ...

The DFA above is not minimal, here is the minimized version:



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 $q \in P_0$ and ending at p that touches a final state.

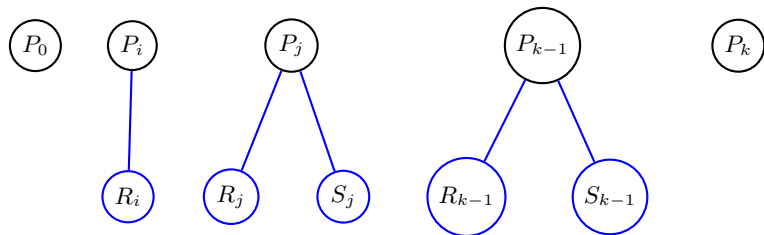
Proof.

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 $S_j \subseteq P_j \cap F$.

□

Total Recall: König's Lemma

A tree is *finitely branching* if every node in the tree has only finitely many children (though there need not be a global bound on the number of children).

Lemma (König's Lemma)

Every infinite but finitely branching tree has an infinite branch.

Proof. Call a node u *fat* if the subtree rooted at u is infinite. So the root u_0 is fat. Suppose we have a path u_0, u_1, \dots, u_n of fat nodes. Since u_n has finitely many successors at least one of them must be fat; let u_{n+1} be any one of those. \square

Note that the proof is non-constructive: there is no algorithmic way to choose u_{n+1} even if the tree has a relatively simple description.

This combinatorial principle is hugely important in reverse mathematics.

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 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 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.

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.