252

State Complexity

Klaus Sutner

Carnegie Mellon University

statecomp 2018/2/2 12:17

Rabin and Scott

In 1959, Rabin and Scott wrote the seminal paper in automata theory

M. Rabin, D. Scott

Finite Automata and Their Decision Problems

IBM Journal of Research and Development
Volume 3, Number 2, Page 114 (1959)

This paper introduces nondeterminism and the systematic study of decision problems associated with finite state machines. It's a must-read classic.

Nondeterministic FSMs

Here is a straightforward generalization of DFAs that allows for nondeterministic behavior.

Definition

A nondeterministic finite automaton (NFA) is a structure

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

consisting of

- a transition system $\langle Q, \Sigma, \tau \rangle$ (labeled digraph)
- an acceptance condition $I, F \subseteq Q$.

Some authors insist that $I = \{q_0\}$. That destroys symmetry and is unnecessary.

Sources of Nondeterminism

So nondeterminism can arise from two different sources:

- Transition nondeterminism:
  there are different transitions $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$.

- Initial state nondeterminism:
  there are multiple initial states.

In other words, even if the transition relation is deterministic we obtain a nondeterministic machine by allowing multiple initial states. Intuitively, this second type of nondeterminism is less wild.

Autonomous Transitions/Epsilon Moves

There is yet another natural generalization beyond just nondeterminism: autonomous transitions, aka epsilon moves. These are transitions where no symbol is read, only the state changes. This is perfectly fine considering our Turing machines ancestors.

Definition

A nondeterministic finite automaton with ε-moves (NFAE) is defined like an NFA, except that the transition relation has the format $\tau \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

Thus, an NFAE may perform several transitions without scanning a symbol.

Hence a trace may now be longer than the corresponding input word. Other than that, the acceptance condition is the same as for NFAs: there has to be run from an initial state to a final state.

Σ_ε

There are occasions where it is convenient to "enlarge" the alphabet Σ by adding the empty word ε:

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

Of course, ε is not a new alphabet symbol. What's really going on?

We are interested in runs of the automaton

$$p_0 \xrightarrow{a} p_1 \xrightarrow{a} p_2 \xrightarrow{b} p_3 \xrightarrow{a} p_4 \xrightarrow{b} p_5 \xrightarrow{b} p_6$$

and want to concatenate the labels: $x = aababb$.

Algebra

Concatenating letters takes place in the monoid of words over Σ, the structure

$$\langle\, \Sigma^{\star}, \cdot, \varepsilon \,\rangle$$

(which is freely generated by Σ). ε is the unit element of this monoid and we can add it to the generators without changing the monoid.

We could even allow arbitrary words and use super-transitions like

$$p \xrightarrow{aba} q$$

Exercise

Explain why this makes no difference as far as languages are concerned.

Conversion to DFA

Our first order of business is to show that NFAs and NFAEs are no more powerful than DFAs in the sense that they only accept regular languages. Note, though, that the size of the machines may change in the conversion process, so one needs to be a bit careful.

The transformation is effective: the key algorithms are

Epsilon Elimination Convert an NFAE into an equivalent NFA.

Determinization Convert an NFA into an equivalent DFA.

Size of an Automaton

There are two important measure of the size of a FSM:

- The state complexity of a FSM is the number of states of the machine.
- The transition complexity of a FSM the number of transitions of the machine.

Transition complexity is more interesting (since it corresponds more faithfully to the size of a FSM data structure), but state complexity is easier to deal with.

Epsilon Elimination

Given an NFAE $\mathcal{A}$ of state complexity n, the first step in ε-elimination is to compute the ε-closures of all states; this takes at most $O(n^3)$ steps.

Introducing new transitions preserves state complexity, but can increase the transition complexity by a quadratic factor.

Aside: If the goal is pattern matching one can try not to pre-process all of $\mathcal{A}$: instead one computes the closures on the fly and only when actually needed. This may be faster if the machine is large and only used a few times.

Laziness Axiom of CS: Don't do anything that you don't have to do.

Exponential Blow-Up

Alas, the Rabin/Scott powerset construction is potentially exponential in the size of $\mathcal{A}$, even when only the accessible part pow$(\mathcal{A})$ is constructed. The only general bound for the state complexity of pow$(\mathcal{A})$ is 2^n.

Warning: An implementation of Rabin/Scott that blindly uses the power set is useless.

Exercise

Figure out how to implement Rabin/Scott properly.

Exponential Blow-Up

In practice, it happens quite often that the accessible part is small.

Alas, there are cases when the state complexity of the deterministic machine is close to 2^n.

Even worse, it can happen that this large power automaton is already minimal, so there is no way to get rid of these exponentially many states.

Good News: Determinization is quite fast as long as the resulting automaton is not too large.

General Abstract Nonsense to the Rescue

Three fundamental (if trivial) principles: characteristic functions, Currying, lifting.

$$A \subseteq B \qquad\qquad A : B \to \mathbf{2}$$

$$f : A \times B \to C \qquad\qquad f : A \to (B \to C)$$

$$f : A \to \mathfrak{P}(Q) \qquad\qquad f : \mathfrak{P}(A) \to \mathfrak{P}(A)$$

Right?

Rabin/Scott Determinization

$$\tau \subseteq Q \times \Sigma \times Q$$

$$\tau : Q \times \Sigma \times Q \to \mathbf{2}$$

$$\tau : Q \times \Sigma \to Q \to \mathbf{2}$$

$$\tau : Q \times \Sigma \to \mathfrak{P}(Q)$$

$$\tau : \mathfrak{P}(Q) \times \Sigma \to \mathfrak{P}(Q)$$

The last function can be interpreted as the transition function of a DFA on state set $\mathfrak{P}(Q)$. Done.

Again, the **Laziness Axiom:** Don't think unless you have to.

Blow-Up Example 1

Recall the kth symbol languages

$$L(a,k) = \{\, x \mid x_k = a \,\}$$

Proposition

$L(a,-k)$ can be recognized by an NFA on $k+1$ states, but the state complexity of this language is 2^k.

Proof.

There is a de Bruijn DFA for the language, a machine on state set $\mathbf{2}^k$ (at least for a binary alphabet).

It remains to show that this machine already has the smallest possible number of states.

Minimality

Suppose $\mathcal{A}$ is a DFA for $L_{0,-k}$ on less than 2^k states.

Consider all 2^k inputs $x \in \mathbf{2}^k$ and let

$$p_x = \delta(q_0, x)$$

Then $p_x = p_y$ for some $x \neq y$.

But then there is a word u such that $xu \in L_{0,-k}$ and $yu \notin L_{0,-k}$.

Contradiction.

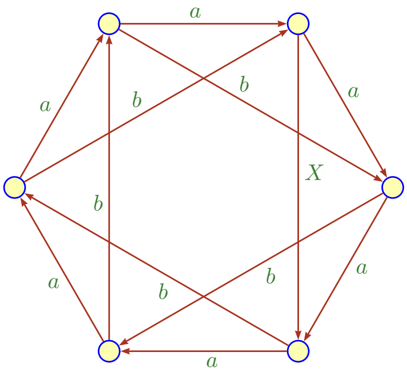
$\square$

Blow-Up Example 2

Here is a 6-state NFA based on a circulant graph. Assume $I = Q$.

If $X = b$ than the power automaton has size 1.

However, for $X = a$ the power automaton has maximal size 2^6.

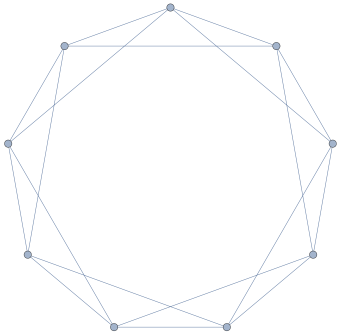


Circulants

The example generalizes to a whole group of circulant machines on n states with diagram $C(n; s, t)$.

These machines are based on circulant graphs:

Vertices	$\{0, 1, \ldots, n-1\}$
Edges	$(v, v + s \bmod n)$ and $(v, v + t \bmod n)$



Pebbling

To prove blow-up results, think of placing a pebble on each initial state in the machine.

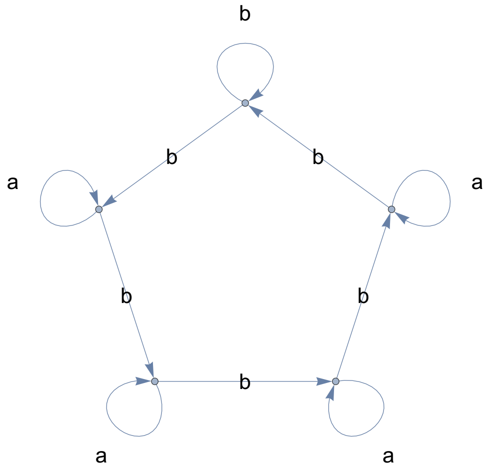
Then fire a sequence of "commands" like $aabbaba$.

- For each command a, all pebbles have to move across an a-labeled edge; otherwise they shrivel up and die.
- If there are multiple edges, the pebbles split.
- If multiple pebbles wind up on the same state, they merge.

The goal is to show that for every $P \subseteq Q$ there is a command sequence x so that x places pebbles on exactly the states in P.

Easy Case

Consider $C(n;0,1)$, label all loops a and all stride 1 edges b. Then switch the label of the loop at 0.



Proof?

a kill 0

b sticky rotate

Note:

- Q is reachable from any $P \neq \emptyset$.
- If we can concoct the operation "plain rotate" we are done.

Case 1: $0 \notin P$ b works

Case 2: $0 \in P$?????

Problems

Exercise

In our automaton $C(6;1,2)$, find the shortest command sequence x that produces P, for any subset $P \subseteq Q$.

Exercise

Prove that full blow-up occurs for all the NFAs over $C(n;1,2)$ when a stride-2 label is switched.

Exercise

How about switching a stride-1 label?

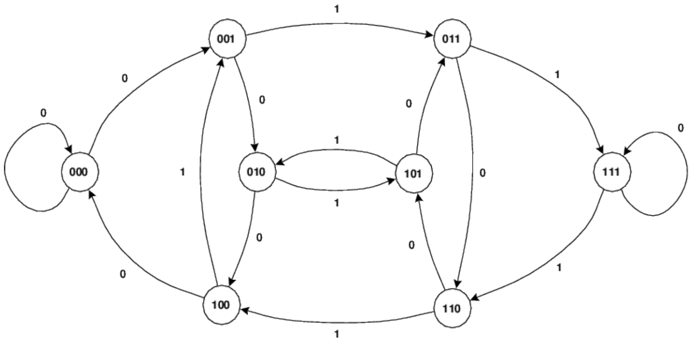
Exercise

How about other circulants $C(n;1,t)$? How about $C(n;s,t)$?

Another Blow-Up

Start with a binary de Bruijn semiautomaton where both δ_0 and δ_1 are permutations. Now flip the label of the loop at $\mathbf{0}$.

For $I = Q$, full blow-up occurs.



A Little Challenge

The loop case I can prove. But here is an open problem:

One can show that the number of permutation labelings in the binary de Bruijn graph of rank k is $2^{2^{k-1}}$. Flipping the label of an arbitrary edge will produce full blow-up in exactly half of the cases.

Conjecture

Full blow-up occurs exactly $2^k \, 2^{2^{k-1}}$ *times.*

This is of interest, since the de Bruijn automata correspond to one-dimensional cellular automata.

Verified experimentally up to $k = 5$ (on Blacklight at PSC, rest in peace). There are $8,388,608$ machines to check, ignoring symmetries.

State Complexity of Operations

	DFA	NFA
intersection	mn	mn
union	mn	$m+n$
concatenation	$(m-1)2^n - 1$	$m+n$
Kleene star	$3 \cdot 2^{n-2}$	$n+1$
reversal	2^n	n
complement	n	2^n

Worst potential blow-up starting from machine(s) of size m, n and applying the corresponding operation.

Note that we are only dealing the state complexity, not transition complexity.

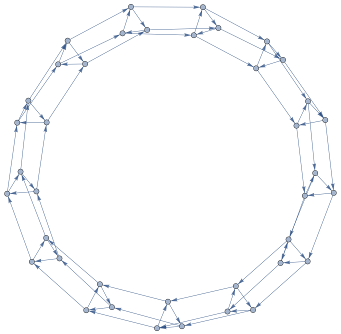
Example: Intersection

Let

$$K_{a,m} = \{\, x \in \mathbf{2}^\star \mid \#_a x = 0 \pmod{m} \,\}$$

be the "mod-counter" language. Clearly $K_{a,m}$ has state complexity m.

The intersection of $K_{0,m}$ and $K_{1,n}$ has state complexity mn.



Weird Automata

In a DFA there is exactly one trace for each input.

In an NFA there may be exponentially many, and acceptance is determined by an existential quantification: is there a run . . . . So here is a tempting question:

Is there a useful notion of acceptance based on
"for all runs such that such and such"?

One problem is whether these "universal" automata are more powerful than ordinary FSMs. As we will see, we only get regular languages.

Of course, this raises the question of how the state complexities compare.

Forall Automata

How would one formally define a type of FSM where acceptance means all runs have a certain property?

The underlying transition system will be unaffected, it is still a labeled digraph.

The acceptance condition now reads:

$\mathcal{A}$ accepts x if all runs of $\mathcal{A}$ on x starting at I end in F.

Let's call these machines ∀FA .

Read: universal FA. Actually, don't: this collides with our previous use where universal means "accepting all inputs." Just look at the beautiful symbol and don't say anything. Wittgenstein would approve.

By the same token, a NFA would be a ∃FA.

Logical And

As an example consider again the mod counter languages

$$K_{a,m} = \{\, x \in \mathbf{2}^\star \mid \#_a\, x = 0 \pmod m \,\}$$

with state complexity m. For the union $K_{0,m} \cup K_{1,n}$ we have a natural NFA of size $m + n$. However, for the intersection $K_{0,m} \cap K_{1,n}$ we only have a product machine that has size mn.

More importantly, note that nondeterminism does not seem to help with intersection: there is no obvious way to construct a smaller NFA for $K_{0,m} \cap K_{1,n}$.

But we can build a ∀FA of size just $m + n$: take the disjoint union and declare the acceptance condition to be universal.

That's Not Insane

Note that acceptance testing for a ∀FA is no harder than for an NFA: we just have to keep track of the set of states $\delta(I, x) \subseteq Q$ reachable under some input and change the notion of acceptance: this time we want $\delta(I, x) \subseteq F$.

For example if some word x crashes all possible computations so that $\delta(I, x) = \emptyset$ then x is accepted.

Likewise we can modify the Rabin-Scott construction that builds an equivalent DFA: as before calculate the (reachable part of the full) powerset and adjust the notion of final state:

$$F' = \{ P \subseteq Q \mid P \subseteq F \}$$

Wisdom

Good mathematicians see analogies between theorems or theories; the very best ones see analogies between analogies.

S. Banach

An Analogy (pace Banach)

We can think of the transitions in a NFA as being disjunctions:

$$\delta(p, a) = q_1 \vee q_2$$

We can pick q_1 or q_2 to continue. Likewise, in a ∀FA we are dealing with conjunctions:

$$\delta(p, a) = q_1 \wedge q_2$$

meaning: We must continue at q_1 and at q_2. So how about

$$\delta(p, a) = (q_1 \vee q_2) \wedge (q_3 \vee q_4)$$

Or perhaps

$$\delta(p, a) = (q_1 \vee \neg q_2) \wedge q_3$$

Does this make any sense?

Threads

Think of threads: both $\wedge$ and $\vee$ correspond to launching multiple threads. The difference is only in how we interpret the results returned from each of the threads.

For $\neg$ there is only one thread, and we flip the answer bit.

In other words, a "Boolean" automaton produces a computation tree rather than just a single branch. This is actually not much more complicated than an NFA.

For historical reasons, these devices are called alternating automata.

Alternating Automata

In an alternating automaton (AFA) we admit transitions of the form

$$\delta(p, a) = \varphi(q_1, q_2, \ldots, q_n)$$

where φ is an arbitrary Boolean formula over Q, even one containing negations.

How would such a machine compute? Initially we are in “state”

$$q_{01} \lor q_{02} \lor \ldots \lor q_{0k}$$

the disjunction of all the initial states.

One Step and Acceptance

Suppose we are in state Φ, some Boolean formula over Q. Then under input a the next state is

$$\Phi[p_1 \mapsto \delta(p_1, a), \ldots, p_n \mapsto \delta(p_n, a)]$$

Thus, all variables $q \in Q$ are replaced by $\delta(q, a)$, yielding a new Boolean formula. In the end we accept if

$$\Phi[F \mapsto 1, \overline{F} \mapsto 0] = 1$$

Exercise

Verify that for NFA and $\forall$FA this definition behaves as expected.

Why Alternating?

The name “alternating automaton” may sound a bit strange.

The original paper by Chandra, Kozen and Stockmeyer that introduced these machines in 1981 showed that one can eliminate negation without reducing the class of languages.

One can then think of alternating between existential states (at least one spawned process must succeed) and universal states (all spawned processes must succeed).

Also note that alternation makes sense for other machines. For example, one can introduce alternating Turing machines (quite important in complexity theory).

Theorem

Alternating automata accept only regular languages.

Proof.

We can build a DFA over the state set $\text{Bool}(Q)$ (the collection of all Boolean formulae with variables in Q) where one representative is chosen in each class of equivalent formulae (so there are at most 2^{2^n} states). For example, we could choose conjunctive normal form.

The transitions are described as above, except that after the substitution we must bring the formula back into the chosen normal form.

The final states are the ones that evaluate to true as above.

$\square$

Why Would Anyone Care?

Because an AFA can be much, much smaller that the minimal DFA. In fact, the 2^{2^n} bound is tight: there are AFAs on n states where the minimal equivalent DFA is doubly exponential in n.

So we have a very concise representation for a regular language but one that still behaves reasonably well under the usual algorithms. Avoiding the standard DFA representation is often critical for feasibility: in reality we cannot actually construct the full DFA. Laziness is a good idea in this context.

BTW, this is even true in pattern matching: DFAs should be avoided unless they are absolutely necessary (because the pattern contains a negation).