

CDM

Nondeterminism and Deterministic Simulation

Klaus Sutner
Carnegie Mellon University
www.cs.cmu.edu/~sutner

Battleplan

- Closure Properties
- Nondeterministic Machines
- Epsilon Elimination
- Deterministic Simulation

Closure Properties

Recall: Closure Properties

We have already seen that regular languages are closed with respect to Boolean operations.

Theorem 1. *Regular languages are closed with respect to union, intersection and complement.*

For union and intersection the proof rests on the construction of a product machine; more precisely, the product of the two given DFA, which is again a DFA.

For complementation the argument is easy: just interchange final and non-final states.

Are there any other interesting closure properties for regular languages?

Concatenation and Nondeterministic Splits

Given two languages $L_1, L_2 \subseteq \Sigma^*$ we can “multiply” them by concatenating all words:

$$L_1 \cdot L_2 = \{ xy \mid x \in L_1, y \in L_2 \}.$$

Suppose we have DFAs for L_1 and L_2 .

Can we build a DFA for L_1L_2 ?

The problem is that given a word w we need to split it as $w = xy$ and then feed x to M_1 and y to M_2 . But there are $|w| + 1$ many ways to do the split, and we have a priori no idea where the break should be.

One can also think of this as a *guess and verify* problem: guess x and y , and then check that indeed M_1 accepts x , and M_2 accepts y .

Of course, there is a slight problem: DFAs don't know how to guess.

Kleene Star and More Nondeterminism

The situation gets worse if we try to construct a DFA for the Kleene star

$$L^* = L^0 \cup L^1 \cup L^2 \dots \cup L^n \cup \dots$$

Not only do we not know where to split the string, we also don't know how many blocks there are.

Moreover, the number of blocks is unbounded (at least in general), and it is far from clear how this type of processing can be handled by a DFA.

Reversal Closure

Here is yet another example of an operation that is difficult to capture within the confines of DFAs.

Let

$$L^r = \{ x^r \mid x \in L \}$$

be the *reversal* of a language.

The direction in which we read a string should be of supreme irrelevance, so for regular languages to form a reasonable class they should be closed under reversal.

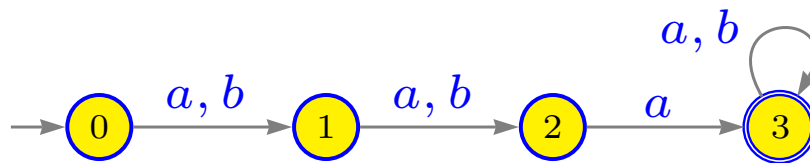
How would we go about constructing a machine for L^r ?

Well, flip all the transitions around. In spirit, that is the right answer, but, of course, the result will almost never be a DFA.

Example: Third Symbol

It is very easy to build a DFA for $L_{a,3} = \{x \mid x_3 = a\}$.

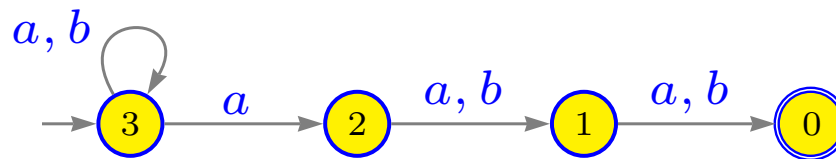
We omit the sink to keep the diagram simple.



Third Symbol From the End

But $L_{a,3}^r = \{x \mid x_{-3} = a\}$ is hard for DFAs: we don't know how far from the end we are.

By flipping transitions (and the whole machine to get the initial state over on the left) we obtain a machine that looks like so:



Now 3 is the initial state and 0 is final.

The problem with this machine is that there are now potentially many computations for the same input.

Nondeterministic Machines

Nondeterministic FSMs

So far we have only considered DFAs (complete and deterministic machines). It's time to deal with more general machines.

Definition 1. A **nondeterministic finite automaton (NFA)** is a structure

$$M = \langle Q, \Sigma, \tau; I, F \rangle$$

where $\langle Q, \Sigma, \tau \rangle$ is a transition system and $I, F \subseteq Q$ are the initial and final states, respectively.

So in general there is no unique next state in an NFA: there may be no next state, or there may be many. But note that every DFA is automatically also an NFA, albeit a very special one.

We have already seen that this is useful in constructing a machine for L^r ; we will see in a moment that concatenation and Kleene star is also easier to handle with nondeterministic machines.

Traces, Runs and Labels

In order to define acceptance for a nondeterministic machine we use traces and runs, just as before for deterministic machines.

Recall that in any transition system $\langle Q, \Sigma, \tau \rangle$ a *trace* is an alternating sequence

$$\pi = p_0, a_1, p_1, \dots, a_r, p_r$$

where $p_i \in Q$, $a_i \in \Sigma$ and $\tau(p_{i-1}, a_i, p_i)$ for all $i = 1, \dots, r$.

p_0 is the *source* of the run and p_r its *target*. The length of π is r .

The corresponding *run* is the sequence p_0, p_1, \dots, p_r of states.

The corresponding *label* or *input* is the word $a_1 a_2 \dots a_r$.

The Fatal Definition

The acceptance condition is essentially the same as for DFAs, except that traces are no longer unique.

Definition 2. *An NFA $M = \langle Q, \Sigma, \tau; I, F \rangle$ **accepts** a word $w \in \Sigma^*$ if there is a run of M with label w , source in I and target in F . We write $\mathcal{L}(M)$ for the acceptance language of M .*

But note that now there may be exponentially many traces with the same label. In particular, some of the traces starting in I may end up in F , others may not.

There is a hidden existential quantifier here.

Again: all that is needed for acceptance is one accepting trace.

Example: Third Symbol from the End

Consider the input $x = baaba$. Here are the possible runs of M from above with this input (for emphasis we write the transitions with arrows). The last one leads from the initial state to the final state, so the machine accepts x .

$$\begin{array}{cccccccccc}
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{a} & 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 \\
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{a} & 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 2 \\
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{a} & 2 & \xrightarrow{b} & 1 & \xrightarrow{a} & 0
 \end{array}$$

But $x = babaa$ is not accepted, none of runs has the right source and target.

$$\begin{array}{cccccccccc}
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{a} & 3 \\
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{a} & 2 \\
 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 3 & \xrightarrow{b} & 3 & \xrightarrow{a} & 2 & \xrightarrow{a} & 1
 \end{array}$$

Acceptance Testing

Recall our first motivation for using DFAs: acceptance testing is very fast. How much of a computational hit do we take when we switch to nondeterministic machines?

In a DFA any input determines a unique run with initial state q_0 and we can simply follow this run. But in an NFA there may be multiple runs starting at several initial states.

We need to follow all these runs. We could try to enumerate the corresponding paths in the transition diagram, but that's a bad idea: there might be exponentially many.

But note that the only important quantity that we need to determine is the set of states we could reach from I with input x .

So all we need to do is to maintain a set of states (which we will do using a simple iterative algorithm scanning the input symbol by symbol).

Membership Testing

Here is a natural modification of the program that tests acceptance for a DFA.

```
P = I;
while( a = x.next() )    // next input symbol
    P = Tau[P, a];

return ( P intersect F != empty );
```

The update step uses

$$\tau(P, a) = \{ q \in Q \mid (p, a, q) \in \tau, p \in P \}$$

where P is a set of states.

Exercise 1. *Prove that this algorithm is correct.*

Running Time

The loop executes $|x|$ times as with DFA.

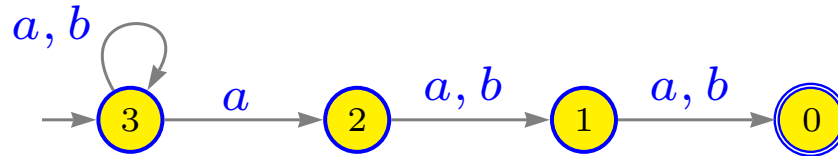
Unfortunately, the loop body is no longer constant time: we have to update a set $P \subseteq Q$.

This can certainly be done in $O(|Q|^2)$ steps.

Actually, it seems that in practice (i.e. in NFAs that appear naturally in some application such as pattern matching) one often deals with overhead that is linear in $|Q|$ rather than quadratic.

At any rate, we can check acceptance in an NFA in $O(|x| |Q|^2)$ steps.

Example: Third Symbol from the End

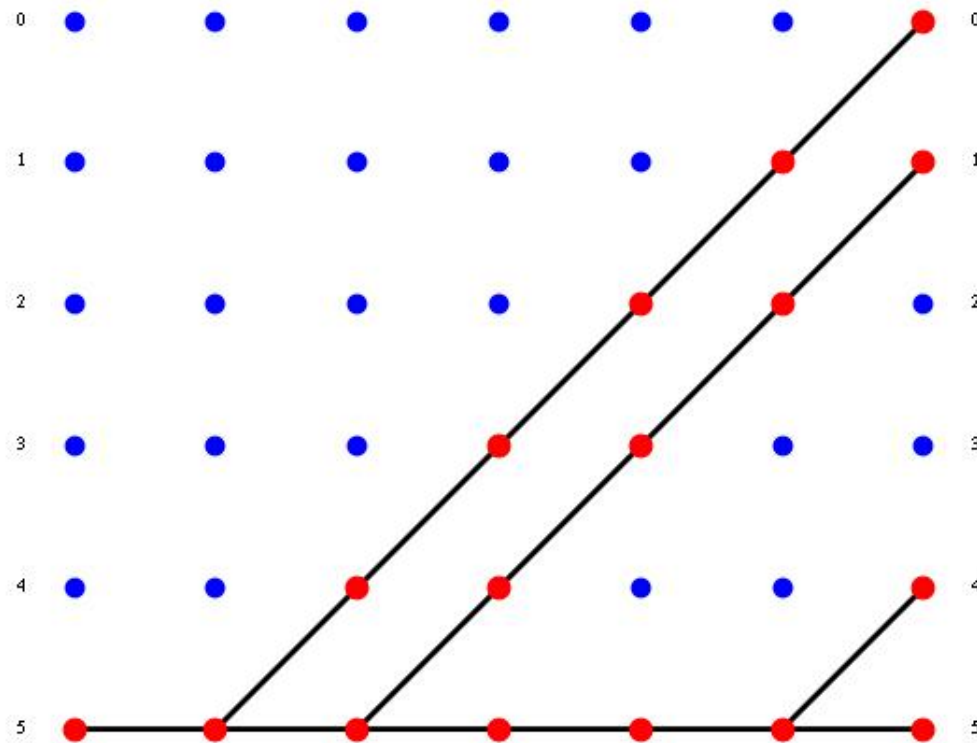


It is helpful to pre-compute the following table, $\tau(P, a)$ is then just the union of some of the sets in the table. Note how the sets in the table are small (think about the 20th symbol from the end).

	0	1	2	3
a	\emptyset	$\{0\}$	$\{1\}$	$\{2, 3\}$
b	\emptyset	$\{0\}$	$\{1\}$	$\{3\}$

Example

For the analogous machine checking for the 5th symbol from the end here is a plot of all runs of length 6 on input $aaabaa$.



So What?

All we have so far is fairly elegant way to construct finite state machines for languages obtained from regular ones by concatenation, Kleene star and reversal.

Alas, the new machines fail to be DFAs, so none of this establishes closure.

We need to show that the acceptance languages of NFAs are again regular.

Note that our constructions also produce NFAs when the original languages are given by NFAs, so things generalize nicely.

Since NFAs are a priori more general than DFAs the question arises if there is an NFA that is not equivalent to any DFA. It turns out the answer is No, but there is a price to pay.

Deterministic Simulation

Conversion of a nondeterministic machine to a deterministic one appeared first in a seminal paper by Rabin and Scott.

Theorem 2. *For every NFA there is an equivalent DFA.*

The idea is to use the Acceptance Testing algorithm for NFAs from above: compute the set of states the automaton could be in after scanning some input x .

More precisely, instead of re-calculating the sets $\tau(P, a)$ for every input we compute them once and for all.

Note that the map $(P, a) \mapsto \tau(P, a)$ is perfectly deterministic, we really obtain a DFA this way.

Proof of Rabin-Scott

Suppose $M = \langle Q, \Sigma, \tau, I, F \rangle$ is an NFA. Let

$$M' = \langle \mathfrak{P}(Q), \Sigma, \delta; I, F' \rangle$$

where $\delta(P, a) = \{ q \in Q \mid \exists p \in P (p, a, q) \in \tau \}$

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

□

The machine from the proof is the *full power automaton* of M , written $\text{pow}_f(M)$, a machine of size 2^n .

Of course, for equivalence only the accessible part $\text{pow}(M)$, the *power automaton* of M , is required.

Example: $L_{a,-3}$

Applying this construction (accessible part only) to the NFA for $L_{a,-3}$ from above we obtain a machine with 8 states

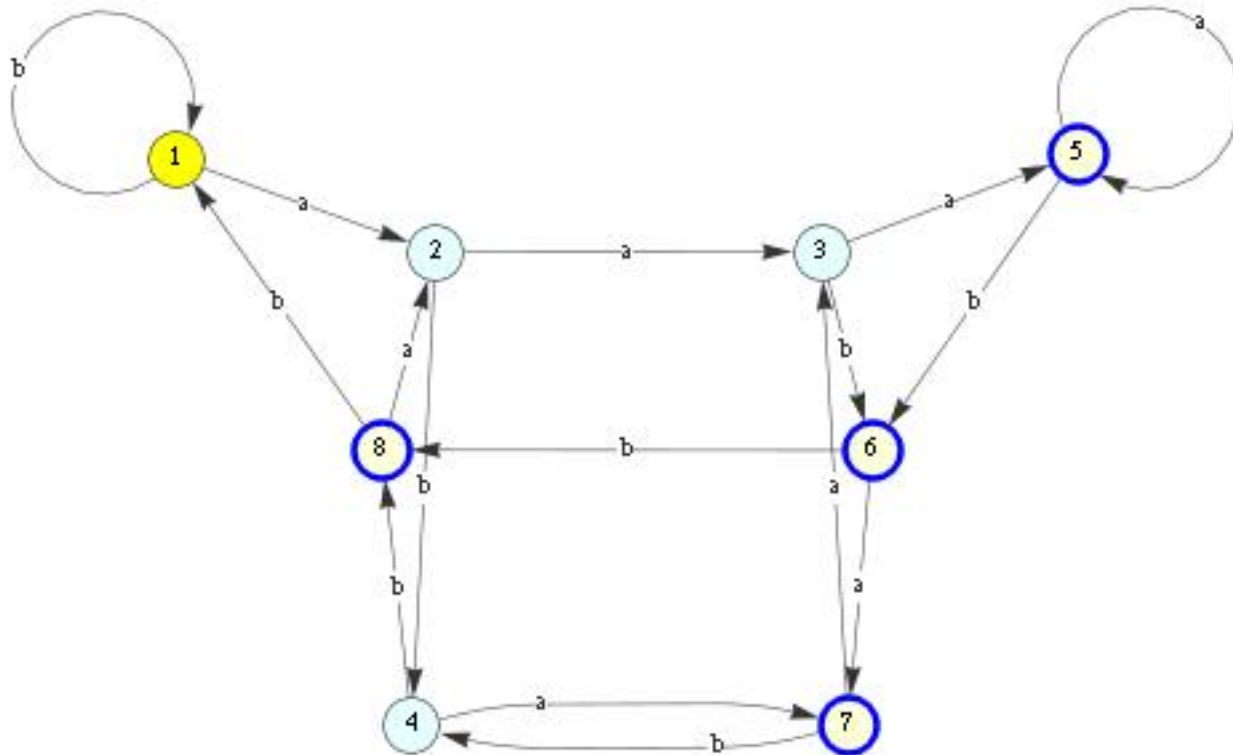
$\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$, $\{1, 3\}$, $\{1, 2, 3, 4\}$, $\{1, 3, 4\}$, $\{1, 2, 4\}$, $\{1, 4\}$

where 1 is initial and 5, 6, 7, and 8 are final. The transitions are given by

	1	2	3	4	5	6	7	8
<i>a</i>	2	3	5	7	5	7	3	2
<i>b</i>	1	4	6	8	6	8	4	1

Note that the full power set has size 16, this construction only builds the accessible part (which happens to have size 8).

Transition System



A Better Mousetrap?

Acceptance testing is slower, nondeterministic machines are not simply all-round superior to DFAs.

- Advantages:
 - Easier to construct and manipulate.
 - Sometimes exponentially smaller.
 - Sometimes algorithms much easier.
- Drawbacks:
 - Acceptance testing slower.
 - Sometimes algorithms more complicated.

Which type of machine to choose in a particular application can be a hard question, there is no easy general answer.

Reversal Closure

Let's give a somewhat formal proof that NFAs really solve the problem of closure under reversal.

Theorem 3. *For any regular language L , the reversal L^r is also regular.*

Proof.

Let $M = \langle Q, \Sigma, \tau, I, F \rangle$ be an NFA for L (perhaps a DFA).

Then

$$M^r = \langle Q, \Sigma, \tau^r, F, I \rangle$$

is an NFA for L^r where

$$\tau^r = \{ (p, a, q) \mid (q, a, p) \in \tau \}$$

Done since NFAs are equivalent to DFAs. □

Back To Concatenation

So suppose we have two NFAs for L_1 and L_2 .

Assume that the state sets are disjoint and think of both machines together as one machine.

We use I_1 as initial states, and F_2 as final states.

A computation starts in I_1 and continues until we get to F_1 .

Then we either continue in M_1 or move to M_2 .

Thus, we need to add transitions from M_1 to M_2 .

Exercise 2. *Figure out how to construct an NFA for concatenation.*

Autonomous Transitions

Epsilon Moves

The last construction of an NFA for concatenation is still a bit cumbersome. It turns out that yet another generalization makes life much easier.

Definition 3. A **nondeterministic finite automaton with ε -moves (NFAE)** is defined like an NFA, except that the transition relation has the format $\tau \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

A transition labeled ε does not consume an input symbol, think of it as an autonomous transition. 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.

NFAE versus NFA

As we will show in a moment, languages accepted by NFAEs are again regular.

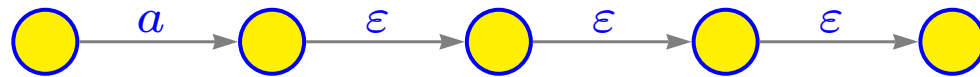
To this end, it suffices to show how any NFAE can be converted into an equivalent NFA, a process called epsilon elimination.

The idea is simple: we remove all ε -transitions and introduce new ordinary transitions that have the same effect.

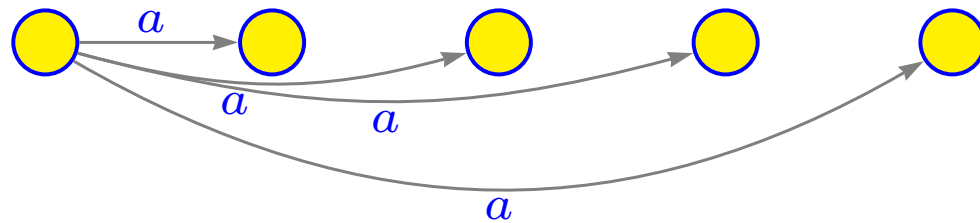
Since there may be chains of ε -transitions this is in essence a transitive closure problem.

ϵ -Closure

A transitive closure problem: we have to replace chains of transitions



by new transitions



Epsilon Elimination

Theorem 4. *For every NFAE there is an equivalent NFA.*

Proof.

This requires no new states, only a change in transitions.

Suppose $M = \langle Q, \Sigma, \tau, I, F \rangle$ is an NFAE for L . Let

$$M' = \langle Q, \Sigma, \tau', I', F \rangle$$

where τ' is obtained from τ as on the last slide.

I' is the ε -closure of I : all states reachable from I using only ε -transitions.

□

Again, there may be quadratic blow-up in the number of transitions.

Concatenation Example

Consider

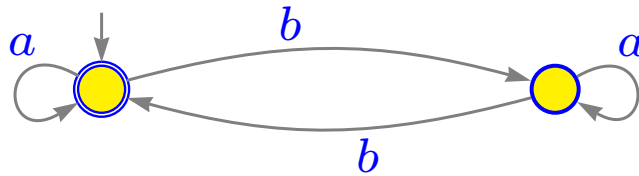
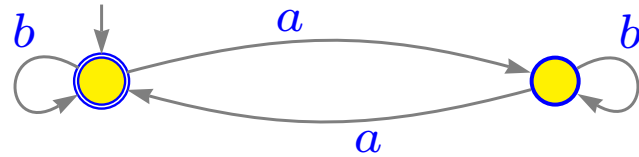
$E_a =$ even number of a 's

$E_b =$ even number of b 's

We will construct two machines M_1 and M_2 for E_a and E_b .

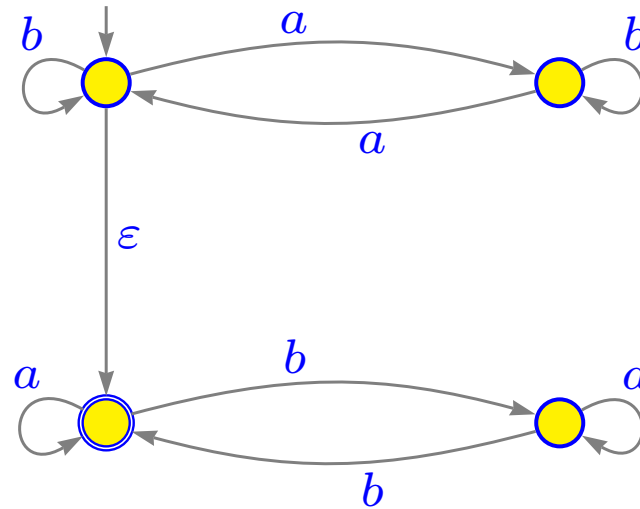
The machines can then be combined into one machine for $L = E_a E_b$.

The Two Machines



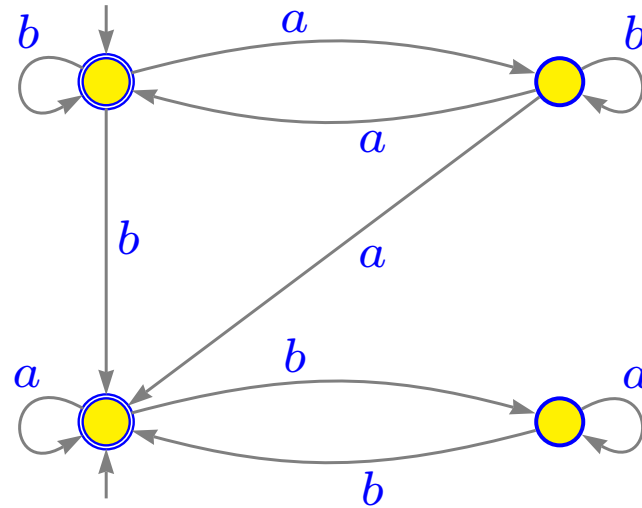
No problem. But it is not so clear what a machine for L would look like. In fact, it's not even clear what L is.

Using ε -Moves



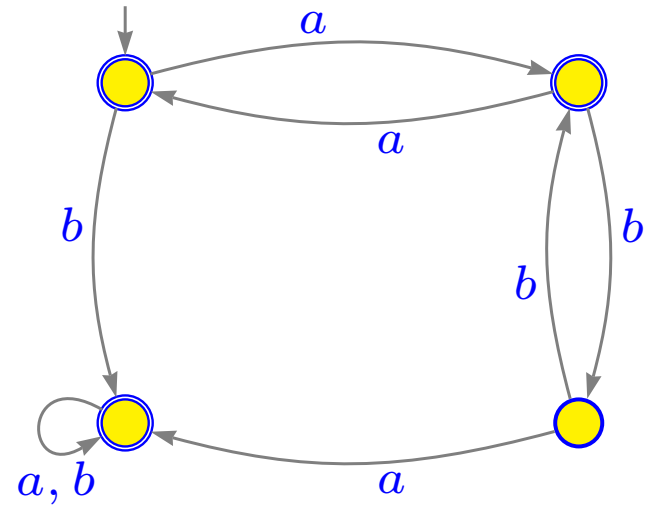
From the final state of the first machine, allow for autonomous transitions to the initial state of the second.

No ε -Moves



We can eliminate the ε -transition by adding appropriate honest transitions.

No Nondeterminism



Lastly, we can eliminate nondeterminism using the Rabin-Scott construction.

The Product Language

Here is the complement of $L = E_a E_b$ up to words of length 8.

ab

$aaab, abbb$

$aaaaab, aaabbb, abbaab, abbbbb$

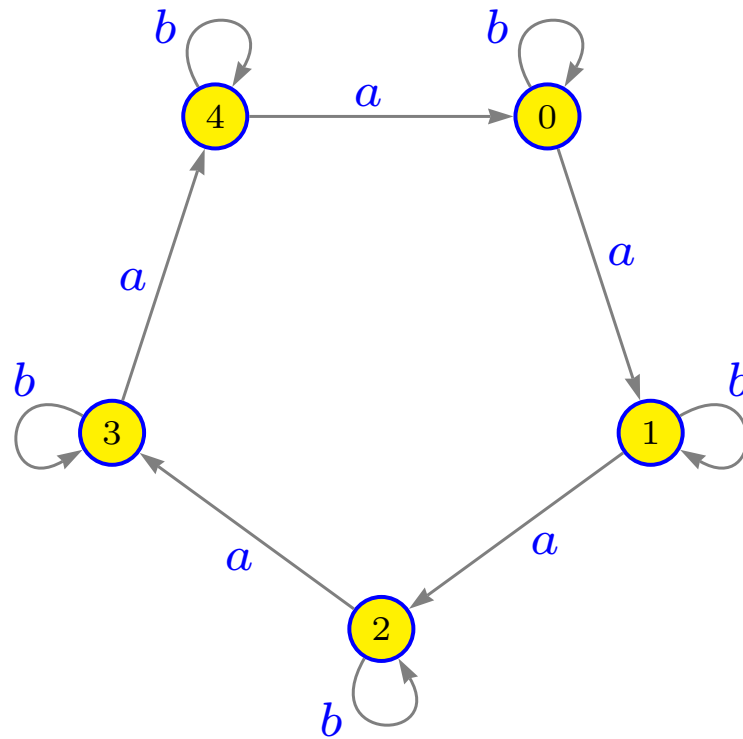
$aaaaaab, aaaaabbb, aaabbaab, aaabbbbb, abbaaab, abbaabbb, abbbbaab, abbbbbb$

Sizes of the complement of L for words up to length 18.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\gamma(n)$	0	0	1	0	2	0	4	0	8	0	16	0	32	0	64	0	128

Exercise 3. Explain the growth rate of L (or, alternatively, of its complement).

Exercise



It is easy to generalize from counting modulo 2 to any other modulus. What would the last construction look like in the general case?

Concatenation Closure

Theorem 5. *For any two regular languages L_1 and L_2 , their concatenation product L_1L_2 is again regular.*

Proof.

Suppose we have NFAEs M_1 and M_2 for L_1 and L_2 . We may safely assume that the state sets are disjoint, set $Q = Q_1 \cup Q_2$.

Add ε -transitions from F_1 to I_2 , choose I_1 as the set of initial states and F_2 as the set of final states. □

Note that the number of states increases linearly, but the number of transitions may increase quadratically. This indicates that state complexity is not a good measure for the size of a nondeterministic machine.

Kleene Star

How do we establish closure with respect to the Kleene star operation?

$$L^* = L^0 \cup L^1 \cup L^2 \dots \cup L^n \cup \dots$$

Note that star is not a finitary operation, so it is not entirely clear how a machine for L^* could be constructed from a machine for L .

Of course, we can build machines for L^n for any n , and thus for

$$L^0 \cup L^1 \cup L^2 \dots \cup L^n$$

Alas, that's neither here nor there: these machines get bigger and bigger and certainly do not have a finite state machine as limit.

Star Closure

Theorem 6. *For any regular language L , L^* is also regular.*

Proof.

Suppose $M = \langle Q, \Sigma, \tau, I, F \rangle$ is an NFAE for L .

Let b and e be two new states and set

$$M' = \langle Q \cup \{b, e\}, \Sigma, \tau', \{b\}, \{e\} \rangle$$

where τ' is τ plus ε -transitions from b to I , F to b , and I to e .

□

This is more complicated than necessary, but adding begin/exit states keeps the number of new transitions linear.

Machines and Complexity

Algorithmic Issues

So we have three increasingly complicated types of machines: DFAs, NFAs and NFAEs, that all accepted exactly the regular languages. There are two conversion algorithms:

- Elimination of ε -moves: conversion from NFAE to NFA.
- Elimination of nondeterminism: conversion from NFA to DFA.

The first one comes down to computing transitive closure of the ε -transitions and can be handled efficiently using standard graph algorithms.

But nondeterminism is more difficult to get rid of: there may be an exponential blow-up in the state complexity of the deterministic machine.

Exponential Blow-Up

From an algorithmic point of view, ε -elimination is no problem: we can compute all the ε -closures in time at most $O(n^3)$.

But the powerset construction is potentially exponential in the size of M .

Of course, it may happen that the accessible part is small, but sometimes (a large part of) the full powerset is accessible. Even worse, it can happen that this large power automaton is already reduced, so there is no way to get rid of these exponentially many states.

Exercise 4. *Determine the running time of a reasonable implementation of the Rabin-Scott construction. Make sure to build only the accessible part.*

Blow-Up Example 1

Recall the languages

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

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

Proof.

Applying the power automaton construction to the canonical NFA produces a DFA of size 2^k , and one can show that this machine is reduced.

□

Constructing a DFA by Hand

Of course, one can also build a DFA for $L(a, -k)$ from scratch. Let's assume $k = 3$.

We need to remember the last 3 symbols of the input: if we've reached the end we have enough information to decide whether we should accept.

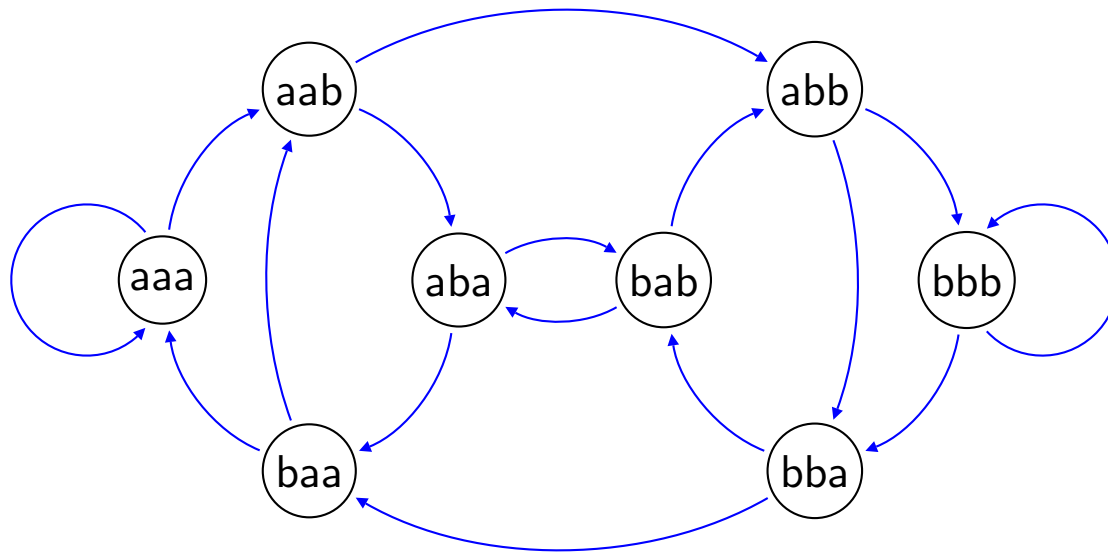
So we use $Q = \{a, b\}^3$ and transitions

$$xyz \xrightarrow{s} yzs$$

Final states are $\{aaa, aab, aba, abb\}$.

Initial state is bbb (a clever hack, otherwise we would have to add some more states $\varepsilon, a, b, aa, ab, \dots$).

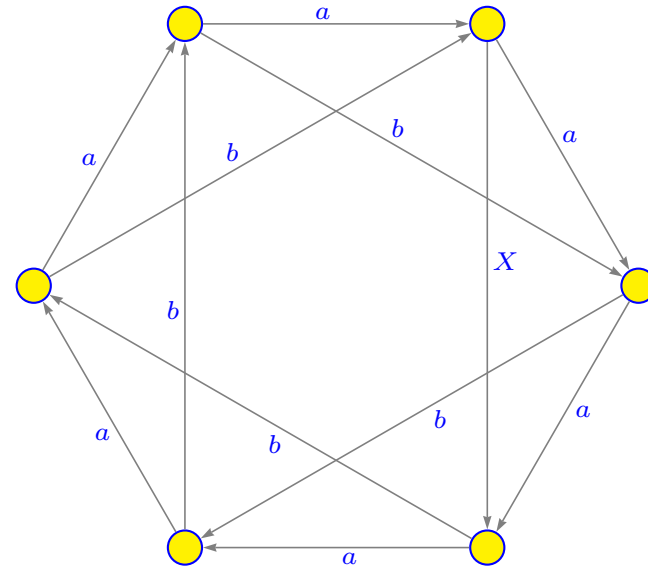
The Diagram



This is a so-called *de Bruijn graph* (of rank 3).

These graphs have lots of interesting properties.

Blow-Up Example 2



Here is a 6-state NFA based on a circulant graph.

Assume $I = F = Q$.

Note that if $X = b$ then the power automaton has size 1.

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

Tip of an Iceberg

The example generalizes to a whole group of circulant machines on n states with diagram $C(n; 1, 2)$.

Start with a labeling where the edges with stride 1 are labeled a and the edges with stride 2 are labeled b .

Then change exactly one of these edge labels: the resulting nondeterministic machines have power automata of size 2^n and the power automata are already reduced.

Exercise 5. *Full blow-up means that for any subset $P \subseteq [n]$ there is some word x such that $\delta([n], x) = P$. Determine such a word x .*

Exercise 6. *Prove that full blow-up occurs for all these NFA.*

Decision Problems

The paper by Rabin and Scott also introduced the study of the computational complexity of various decision problems associated with finite state machines.

All the problems we consider here are easily decidable for any kind of finite state machine, but there can be a big gap in the low-level complexity of a problem depending on whether the machine is deterministic or not: conversion from NFAE to NFA is cheap (at least up to polynomial time computation), but the step from NFA to DFA can be exponentially expensive.

We are particularly interested whether a problem is solvable in polynomial time or whether it is hard (in some suitable complexity class).

For problems in \mathbb{P} highly efficient algorithms are of interest.

Classical Problems

Problem: **Emptiness Problem**

Instance: A regular language L .

Question: Is L empty?

Problem: **Finiteness Problem**

Instance: A regular language L .

Question: Is L finite?

Problem: **Universality Problem**

Instance: A regular language L .

Question: Is $L = \Sigma^*$?

Emptiness and Finiteness are easily polynomial time for DFAs and NFAs.

Universality is polynomial time for DFAs but PSPACE-complete for NFAs.

Classical Problems

Problem: **Equality Problem**

Instance: Two regular languages L_1 and L_2 .

Question: Is L_1 equal to L_2 ?

Problem: **Inclusion Problem**

Instance: Two regular languages L_1 and L_2 .

Question: Is L_1 a subset of L_2 ?

Emptiness and Finiteness are polynomial time for DFAs.

Both problems are PSPACE-complete for NFAs.

Predicting Blow-Up

Many algorithms in particular in the area of pattern matching naturally produce nondeterministic machines.

Exponential blow-up makes it somewhat difficult to decide whether it is advantageous to compute the corresponding power automaton: the actual matching process is faster but the machine may be too large.

Likewise, it is not clear in general whether minimization is preferable: the cost of minimization is significant, the speed-up in acceptance testing essentially nil.

It would be nice if one could perform a simple, cheap test to determine what the size of the power automaton would if one were to go ahead with conversion. Unfortunately, the following problem is PSPACE-hard:

Problem: **Power Automaton Size**

Instance: A nondeterministic machine M , a bound B .

Question: Is the size of the power automaton of M bounded by B ?

Summary

- Deterministic finite state machines can be used to recognize some patterns (certain classes of input strings) very efficiently.
- In order to deal with more complicated patterns it is convenient to generalize to nondeterministic machines and even machines with autonomous transitions.
- Acceptance testing for these generalized machines is slower than for DFAs.
- All these generalized machines turn out to be equivalent to the original DFAs, albeit at a potentially exponential blow-up in size.
- Some decision problems are polynomial time solvable for deterministic and nondeterministic machines, but some problems for nondeterministic machines are PSPACE-hard (Equality, Universality).