Language Processors F29LP2, Lecture 5

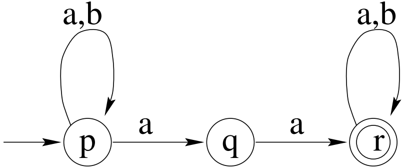
Jamie Gabbay

February 2, 2014

Nondeterministic Finite Automata (NFA)

NFA generalise deterministic finite automata (DFA). They allow several (0, 1, or more than 1) outgoing transitions with the same label.

An NFA accepts a word if some choice of transitions takes the machine to a final state (other choices may lead to a non-final state; we don't care).

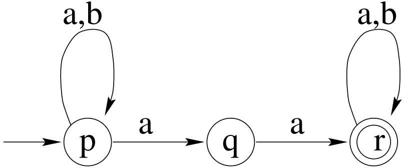


Note: two transitions out of p with input letter a, going to p and q.

Note: no transitions from q with input letter b. The number of transitions may be zero, one or more.

Nondeterministic Finite Automata (NFA)

NFA may have different computations for the same input word.



Consider $w = abaa$. The first input letter a gives two choices: go to p or go to q.

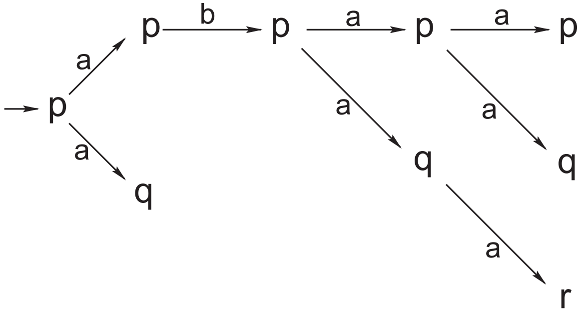
If we go to q there is no transition with b; we get stuck.

If we go to p then the second input letter b takes the machine to p. Then a and a can take the machine to q, then r. There exists a path to a final state; $abaa$ is accepted; and

$$abaa \in L(A).$$

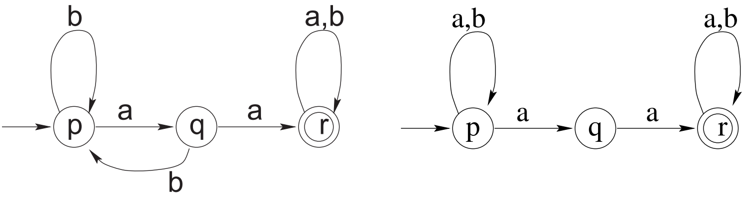
Nondeterministic Finite Automata (NFA)

A computation tree summarises computations with input *abaa*:



We see that the NFA accepts exactly those words that contain *aa* as a subword, so the NFA is equivalent to the DFA of lecture 4 slide 7.

Nondeterministic Finite Automata (NFA)



Call two automata equivalent when they recognise the same language.

Precise definition of NFA

An NFA $A = (Q, \Sigma, \delta, q_0, F)$ is specified by:

- State set Q,
- input alphabet Σ,
- transition function δ,
- initial state q_0 and
- the final state set F.

δ is defined differently than for DFAs.

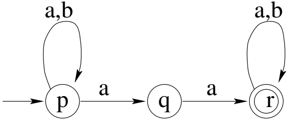
It gives for each state q and input letter a a set $\delta(q, a) \subseteq Q$ of possible next states. Using the power set notation

$$2^Q = \{S \mid S \subseteq Q\} \quad \text{we can write} \quad \delta : Q \times \Sigma \longrightarrow 2^Q.$$

Precise definition of NFA

δ for the NFA on slide 2 is given as follows:

	a	b
p	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\emptyset$
r	$\{r\}$	$\{r\}$



Extending δ

Extend δ to $\hat{\delta}$ as we did in DFA. Define

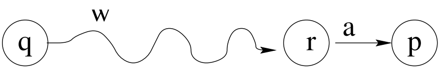
$$\hat{\delta} : Q \times \Sigma^* \longrightarrow 2^Q$$

such that $\hat{\delta}(q, w)$ is the set of all states the machine can reach from state q reading input word w.

A recursive definition goes as follows:

1. For every state q, $\qquad \hat{\delta}(q, \varepsilon) = \{q\}$.
2. For every state q, word w and letter a,

$$\hat{\delta}(q, wa) = \{p \in Q \mid \exists r \in \hat{\delta}(q, w) : \ p \in \delta(r, a)\} = \bigcup_{r \in \hat{\delta}(q,w)} \delta(r, a).$$



Extending δ

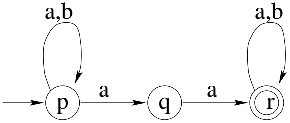
On single input letters, δ and $\hat{\delta}$ are equal:

$$\delta(q, a) = \hat{\delta}(q, a).$$

So there's no confusion if we drop the hat and write δ instead of $\hat{\delta}$.

In the NFA of slide 2

$$\begin{array}{lcl} \delta(p, a) & = & \{p, q\}, \\ \delta(p, ab) & = & \{p\}, \\ \delta(p, aba) & = & \{p, q\}, \\ \delta(p, abaa) & = & \{p, q, r\}. \end{array}$$



Language recognised by an NFA

The language recognised by NFA $A = (Q, \Sigma, \delta, q_0, F)$ is

$$L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}.$$

That is, $L(A)$ contains the w such that some final state is reachable from q_0 using w.

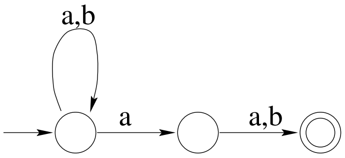
There might also be non-final states reachable from q_0, but we don't care.

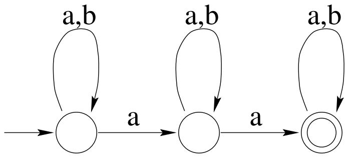
Some exercises:

Construct NFAs over $\Sigma = \{a, b\}$ that recognise the following languages:

1. Words ending in *ab*.
2. Words containing *aba* as a subword.
3. Words starting with *ab* and ending with *ba*.
4. Words containing two *b*s separated by an even number of *a*s.

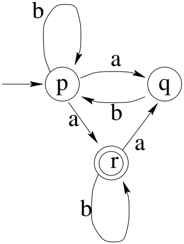
Conversely, determine (and describe in English) the languages recognised by the following NFA:





Converting an NFA to a DFA

How to check whether this NFA



accepts $w = abbaabb$?

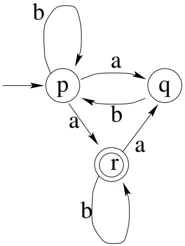
We check whether any computation paths for w end in a final (accepting) state. This can grow exponentially with the length of the input!

Better to scan the input once, keeping track of the set of possible states.

Converting an NFA to a DFA

For example, with the example of the previous slide and input $w = abbaabb$ we have

$$\{p\} \xrightarrow{a} \{q, r\} \xrightarrow{b} \{p, r\} \xrightarrow{b} \{p, r\}$$
$$\xrightarrow{a} \{q, r\} \xrightarrow{a} \{q\} \xrightarrow{b} \{p\} \xrightarrow{b} \{p\}$$



so $\delta(p, w) = \{p\}$, and word w is not accepted because p is not a final state.

Testing is deterministic (even though the NFA is nondeterministic).

Converting an NFA to a DFA

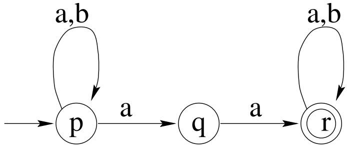
Makes it 'obvious' that any NFA can be converted to a DFA.

The states of the DFA are sets of states of the NFA; transitions are such that the state of the DFA after input w is $\delta(q_0, w)$ the set of states that can be reached in the NFA with input w.

So NFAs are just a convenient shorthand for a (possibly much larger) DFA. NFA only recognise regular languages.

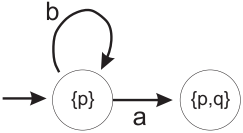
Converting an NFA to a DFA

Consider



Initially the NFA is in state p so the corresponding DFA is initially in state $\{p\}$.

With input letter a the NFA may move to either state p or q, so after input a the DFA will be in state $\{p, q\}$. With input b the NFA remains in state p:



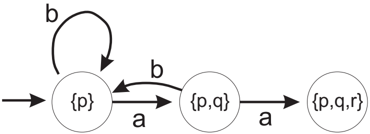
Converting an NFA to a DFA

Next figure out the transitions from state $\{p, q\}$.

If the DFA is in state $\{p, q\}$ it means that the NFA can be in either state p or q. With input a the NFA can move to p or q (if it was in state p) or to r (if it was in state q).

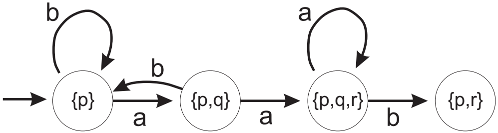
Therefore, with input letter a the machine can move to p or q or r, and so the DFA must move to $\{p, q, r\}$.

With input b the only transition from states p and q is into p. That is why DFA has transition from $\{p, q\}$ back into $\{p\}$:

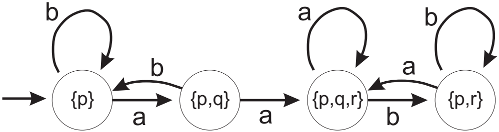


Converting an NFA to a DFA

Consider $\{p, q, r\}$. With input a the NFA can reach any state so the DFA loops to $\{p, q, r\}$. With input b the NFA can move to p (if it was at p) or to r (if it was at r) so the DFA has a transition from $\{p, q, r\}$ into $\{p, r\}$:



We again have a new state $\{p, r\}$ to process. From p and r the NFA can go to any state with input a, and to p and r with input b:

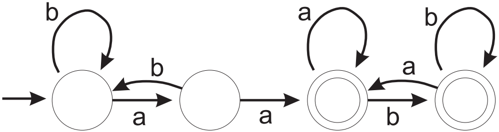


Converting an NFA to a DFA

No new states introduced! All necessary transitions in place!

We still have to calculate final states. The NFA word w if it leads to at least one final state.

The DFA should accept w when it ends to a state S containing at least one final state of the NFA. Our sample NFA has only one final state r, so every set containing r is final:



Converting an NFA to a DFA

The construction is complete. We have a DFA that accepts the same language as the original NFA.

This construction is called a powerset construction. It is plausible (and true) that the powerset construction can be applied to any NFA to convert it into a DFA.

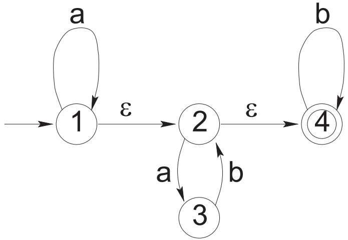
There may be other DFA accepting the same language, some of which may be simpler than the powerset DFA.

We don't care: we only wanted to demonstrate that there exists at least one such DFA.

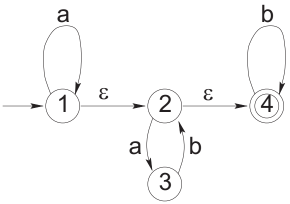
ε-moves

Surprisingly useful and convenient: extend NFA by allowing spontaneous transitions.

When an NFA executes a spontaneous transition, known as an ε-move, it changes its state without reading any input letter. Any number of ε-moves are allowed.



ε-moves



Word $w = aabbb$ is accepted as follows: The first a keeps the machine in state 1. An ε-move to state 2 is executed without reading any input. Next ab is consumed through states 3 and 2, followed by another ε-move to state 4. The last bb keeps the automaton in the accepting state 4.

The automaton of this example accepts any sequence of as followed by any repetition of abs followed by any number of bs:

$$L(A) = \{a^i(ab)^j b^k \mid i, j, k \geq 0\}.$$