Non-deterministic Finite Automata

Informatics 2A: Lecture 4

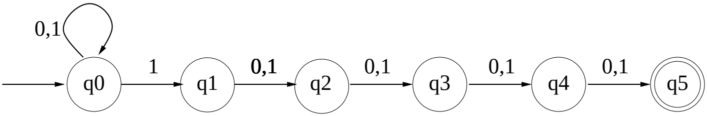
Alex Simpson

School of Informatics
University of Edinburgh
als@inf.ed.ac.uk

25 September, 2012

Variation on a theme: Non-deterministic finite automata

In an NFA, for any current state and any symbol, there may be zero, one or many new states we can jump to.



Here there are two transitions for ‘1’ from q0, and none from q5.

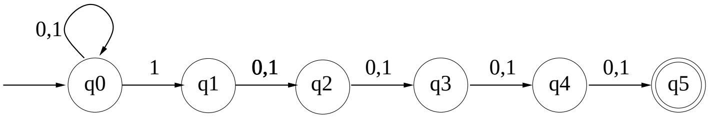
NFAs are useful because . . .

- We often wish to ignore certain details of a system, and model just the range of possible behaviours.
- Some languages can be specified much more concisely by NFAs than by DFAs.
- Certain useful facts about regular languages are most conveniently *proved* using NFAs.

The language accepted by an NFA

The language associated with an NFA is defined to consist of all strings that are accepted under some possible execution run.

Example:



The associated language is

$$\{x \in \Sigma^* \mid \text{the fifth symbol from the end of } x \text{ is } 1\}$$

To ponder: Could you design a DFA for the same language?

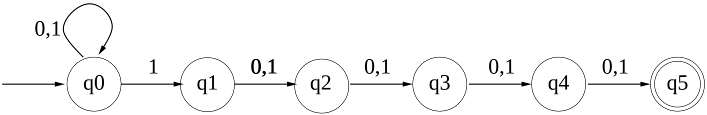
Formal definition of NFAs

Formally, an NFA N with alphabet Σ consists of:

- A set Q of states,
- A transition relation $\Delta \subseteq Q \times \Sigma \times Q$,
- A set $S \subseteq Q$ of possible starting states.
- A set $F \subseteq Q$ of accepting states.

Note: any DFA is an NFA!

Example formal definition



$$Q = \{q0, q1, q2, q3, q4, q5\}$$

$$\begin{aligned}\Delta = \{ & (q0, 0, q0), (q0, 1, q0), (q0, 1, q1), (q1, 0, q2), \\ & (q1, 1, q2), (q2, 0, q3), (q2, 1, q3), (q3, 0, q4), \\ & (q3, 1, q4), (q4, 0, q5), (q4, 1, q5) \}\end{aligned}$$

$$S = \{q0\}$$

$$F = \{q5\}$$

Formal definition of acceptance

From the formal definition of an NFA, we can define a many-step transition relation $\widehat{\Delta} \subseteq Q \times \Sigma^* \times Q$:

$$
\begin{aligned}
(q, \epsilon, q') \in \widehat{\Delta} &\quad \text{iff} \quad q' = q \\
(q, xu, q') \in \widehat{\Delta} &\quad \text{iff} \quad \exists q''.\ (q, x, q'') \in \widehat{\Delta} \ \&\ (q'', u, q') \in \Delta
\end{aligned}
$$

The language accepted by the NFA is then

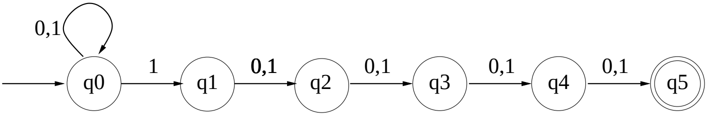
$$\mathcal{L}(N) \ = \ \{x \in \Sigma^* \mid \exists s, q.\ s \in S \ \&\ (s, x, q) \in \widehat{\Delta} \ \&\ q \in F\}$$

DFAs and NFAs

- By definition, a regular language is one that is recognized by some DFA.
- Every DFA is an NFA, but not *vice versa*.
- So you might wonder whether NFAs are 'more powerful' than DFAs. Are there languages that can be recognized by an NFA but not by any DFA?
- The main goal of the lecture is to show that the answer is No. In fact, any NFA can be *converted* into a DFA with exactly the same associated language.
- So regular languages can equally well be defined as those that are exactly recognized by some NFA. This makes it easy to prove some further useful facts about regular languages.

Clicker question

Consider our example NFA over $\{0, 1\}$:

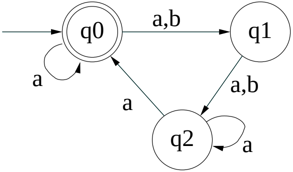


In what range is the number of states of the smallest equivalent DFA?

- A: ≤ 9
- B: 10–19
- C: 20–29
- D: 30–39

NFAs to DFAs: the idea

Given an NFA N over Σ and a string $x \in \Sigma^*$, how would you *in practice* decide whether $x \in \mathcal{L}(N)$?

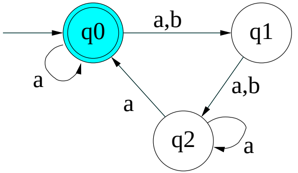


String to process: aba

Idea: At each stage in processing the string, keep track of all the states the machine might possibly be in.

Stage 0: initial state

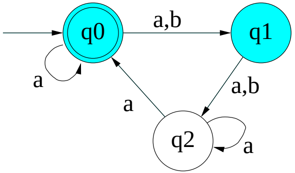
At the start, the NFA *can only be* in the initial state q0.



String to process:	aba
Processed so far:	ϵ
Next symbol:	a

Stage 1: after processing ‘a’

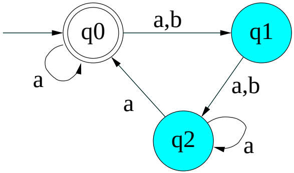
The NFA could now be in either q0 or q1.



String to process:	aba
Processed so far:	a
Next symbol:	b

Stage 2: after processing ‘ab’

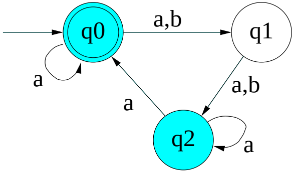
The NFA could now be in either q1 or q2.



String to process:	aba
Processed so far:	ab
Next symbol:	a

Stage 3: final state

The NFA could now be in q2 or q0. (It could have got to q2 in two different ways, though we don't need to keep track of this.)



String to process: aba
Processed so far: aba

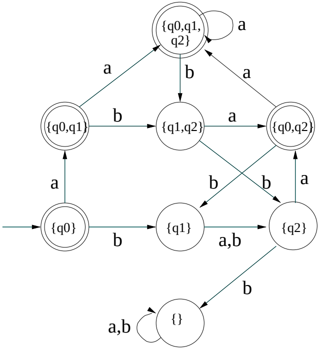
Since we've reached the end of the input string, and the set of possible states includes the accepting state q0, we can say that the string aba is accepted by this NFA.

The key insight

- The process we've just described is a completely deterministic process! Given any current set of 'coloured' states, and any input symbol in Σ, there's only one right answer to the question: 'What should the new set of coloured states be?'
- What's more, it's a finite state process. A 'state' is simply a choice of 'coloured' states in the original NFA N.
  If N has n states, there are 2^n such choices.
- This suggests how an NFA with n states can be converted into an equivalent DFA with 2^n states.

The subset construction: example

Our 3-state NFA gives rise to a DFA with $2^3 = 8$ states. The states of this DFA are subsets of $\{q0, q1, q2\}$.



(Example string: aba)

The accepting states of this DFA are exactly those that *contain* an accepting state of the original NFA.

The subset construction in general

Given an NFA $N = (Q, \Delta, S, F)$, we can define an equivalent DFA $M = (Q', \delta', s', F')$ (over the same alphabet Σ) like this:

- Q' is 2^Q, the set of all subsets of Q. (Also written $\mathcal{P}(Q)$.)
- $\delta'(A, u) = \{q' \in Q \mid \exists q \in A.\ (q, u, q') \in \Delta\}$. (Set of all states reachable via u from *some* state in A.)
- $s' = S$.
- $F' = \{A \subseteq Q \mid \exists q \in A.\ q \in F\}$.

It's then not hard to prove mathematically that $\mathcal{L}(M) = \mathcal{L}(N)$. (See Kozen for details.)

The subset construction: Summary

- We've shown that for any NFA N, we can construct a DFA M with the same associated language.
- So an alternative definition of 'regular language' would be 'language recognized by some NFA'.
- Often a language can be specified more concisely by an NFA than by a DFA.
- We can automatically convert an NFA to a DFA any time we want, at the risk of an exponential blow-up in the number of states.

  In practice, DFA minimization will often mitigate this.

The subset construction: Summary

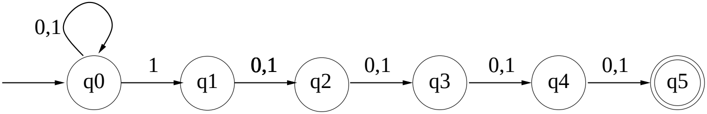
- We've shown that for any NFA N, we can construct a DFA M with the same associated language.
- So an alternative definition of 'regular language' would be 'language recognized by some NFA'.
- Often a language can be specified more concisely by an NFA than by a DFA.
- We can automatically convert an NFA to a DFA any time we want, at the risk of an exponential blow-up in the number of states.

  In practice, DFA minimization will often mitigate this.

  But not always!

Exponential blow-up: an example

Recall the example NFA from earlier:



Associated language:

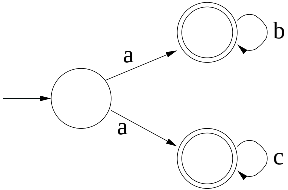
$$\{x \in \Sigma^* \mid \text{the fifth symbol from the end of } x \text{ is } 1\}$$

Any DFA for recognizing this language will need at least $2^5 = 32$ states, since in effect such a machine has to 'remember' the last five symbols seen.

In fact the minimal DFA has exactly 32 states.

Clicker question

Consider the following NFA over $\{a, b, c\}$:



What is the *minimum* number of states of an equivalent DFA?

A: 3

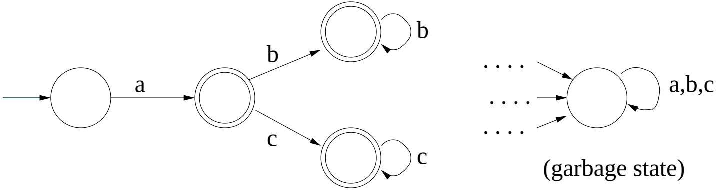
B: 4

C: 5

D: 6

Solution

An equivalent DFA must have at least 5 states!



NFAs: a first application

Consider the following little theorem:

If L_1 and L_2 are regular languages over Σ, so is $L_1 \cup L_2$.

This *can* be shown using DFAs ... but it's dead easy using NFAs.

Suppose $N_1 = (Q_1, \Delta_1, S_1, F_1)$ is an NFA for L_1, and $N_2 = (Q_2, \Delta_2, S_2, F_2)$ is an NFA for L_2.

We may assume $Q_1 \cap Q_2 = \emptyset$ (just relabel states if not).

Now consider the NFA

$$(Q_1 \cup Q_2,\ \Delta_1 \cup \Delta_2,\ S_1 \cup S_2,\ F_1 \cup F_2)$$

This is just N_1 and N_2 'side by side'. Clearly, this NFA recognizes precisely $L_1 \cup L_2$.

(Quite useful in practice — no state explosion!)

Reading

Relevant reading:

- Kozen chapters 5 and 6;
  J & M section 2.2.7 (very brief).

Next time: Yet another way of specifying regular languages: via regular expressions (cf. Inf 1 Computation & Logic).