Finite Automata

Informatics 2A: Lecture 3

Mary Cryan

School of Informatics
University of Edinburgh
`mcryan@inf.ed.ac.uk`

21 September 2018

Languages and alphabets

Throughout this course, languages will consist of finite sequences of symbols drawn from some given alphabet.
An alphabet Σ is simply some finite set of *letters* or *symbols* which we treat as 'primitive'. These might be ...

- English letters: $\Sigma = \{a, b, \ldots, z\}$
- Decimal digits: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- ASCII characters: $\Sigma = \{0, 1, \ldots, a, b, \ldots, ?, !, \ldots\}$
- Programming language 'tokens': $\Sigma = \{\texttt{if}, \texttt{while}, \texttt{x}, ==, \ldots\}$
- Words in (some fragment of) a natural language.
- 'Primitive' actions performable by a machine or system, e.g. $\Sigma = \{\texttt{insert50p}, \texttt{pressButton1}, \ldots\}$

In toy examples, we'll use simple alphabets like $\{0, 1\}$ or $\{a, b, c\}$.

What is a 'language'?

A language over an alphabet Σ will consist of finite sequences (strings) of elements of Σ. E.g. the following are strings over the alphabet $\Sigma = \{a, b, c\}$:

$$a \qquad b \qquad ab \qquad cab \qquad bacca \qquad cccccccc$$

There's also the empty string , which we usually write as ϵ.

A language over Σ is simply a (finite or infinite) set of strings over Σ. A string s is legal in the language L if and only if $s \in L$.

We write Σ^* for the set of *all* possible strings over Σ. So a language L is simply a subset of Σ^*. ($L \subseteq \Sigma^*$)

(N.B. This is just a technical definition — any *real* language is obviously much more than this!)

Ways to define a language

There are many ways in which we might formally define a language:

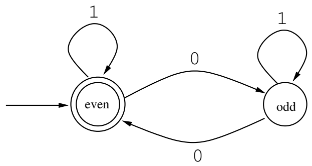
- Direct mathematical definition, e.g.

$$\begin{aligned} L_1 &= \{a, aa, ab, abbc\} \\ L_2 &= \{axb \mid x \in \Sigma^*\} \\ L_3 &= \{a^n b^n \mid n \geq 0\} \end{aligned}$$

- Regular expressions (see Lecture 5): e.g. $a(a+b)^*b$.
- Formal grammars (see Lecture 9 onwards): e.g. $S \to \epsilon \mid aSb$.
- Specify some machine for testing whether a string is legal or not.

The more complex the language, the more complex the machine might need to be. As we shall see, each level in the Chomsky hierarchy is correlated with a certain class of machines.

Finite automata (a.k.a. finite state machines)



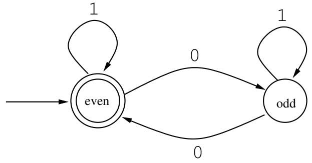
This is an example of a finite automaton over $\Sigma = \{0, 1\}$.

At any moment, the machine is in one of 2 states. From any state, each symbol in Σ determines a ‘destination’ state we can jump to.

The state marked with the in-arrow is picked out as the starting state. So any string in Σ^* gives rise to a sequence of states.

Certain states (with double circles) are designated as accepting. We call a string ‘legal’ if it takes us from the start state to some accepting state. In this way, the machine defines a language $L \subseteq \Sigma^*$: the language L is the set of all legal strings.

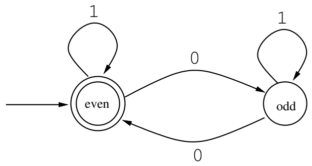
Quick test question



For the finite state machine shown here, which of the following strings are legal (i.e. accepted)?

1. ϵ
2. 11
3. 1010
4. 1101

Quick test question

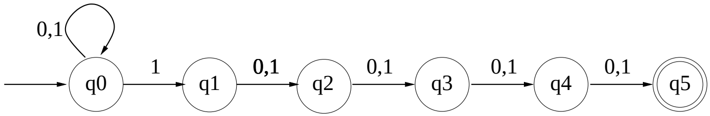


For the finite state machine shown here, which of the following strings are legal (i.e. accepted)?

1. ϵ
2. 11
3. 1010
4. 1101

Answer: 1, 2, 3, are legal, 4 isn't.

More generally, 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.

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

The language associated with the example machine above is

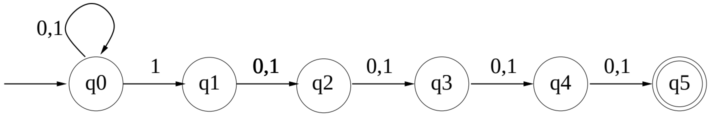
$$\{x \in \Sigma^* \mid \text{the symbol fifth from the end of } x \text{ is } 1\}$$

Formal definition of finite automaton

Formally, a finite automaton with alphabet Σ consists of:

- A finite 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.

Example formal definition



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

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

$$S = \{q0\}$$

$$F = \{q5\}$$

Regular language

Suppose $M = (Q, \Delta, S, F)$ is a finite automaton with alphabet Σ.

We say that a string $x \in \Sigma^*$ is accepted if there exists a path through the set of states Q, starting at some state $s \in S$, ending at some state $f \in F$, with each step taken from the Δ relation, and with the path as a whole spelling out the string x.

This enables us to define the language accepted by M:

$$\mathcal{L}(M) \;=\; \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$$

We call a language $L \subseteq \Sigma^*$ regular if $L = \mathcal{L}(M)$ for *some* finite automaton M.

Regular languages are the subject of lectures 4–8 of the course.

DFAs and NFAs

A finite automaton with alphabet Σ is deterministic if:

- It has exactly one starting state.
- For every state $q \in Q$ and symbol $a \in \Sigma$ there is exactly one state q' for which there exists a transition $q \xrightarrow{a} q'$ in Δ.
  (in some texts/definitions, this is relaxed to "at most one state")

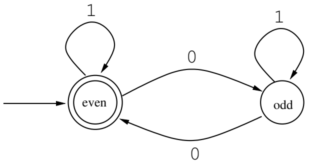
The first condition says that S is a singleton set.
The second condition says that Δ specifies a function $Q \times \Sigma \to Q$.

Deterministic finite automata are usually abbreviated DFAs.

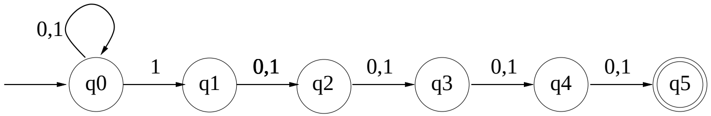
General finite automata are usually called nondeterministic, by way of contrast, and abbreviated NFAs.

Note that every DFA is an NFA.

Example



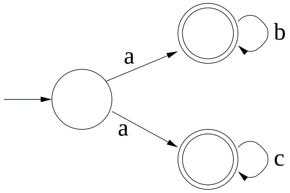
This is a DFA (and hence an NFA).



This is an NFA but not a DFA.

Challenge question

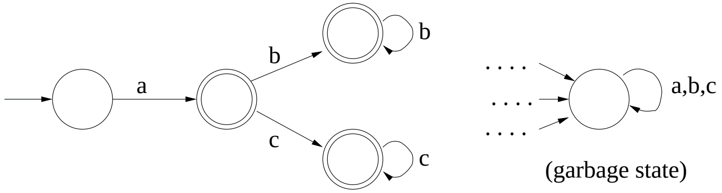
Consider the following NFA over $\{a, b, c\}$:



What is the *minimum* number of states of an equivalent DFA?
(well, first we should ask - *is* there an equivalent DFA?)

Solution

An equivalent DFA must have at least 5 states!



Specifying a DFA

Clearly, a DFA with alphabet Σ can equivalently be given by:

- A finite set Q of states,
- A transition function $\delta : Q \times \Sigma \to Q$,
- A single designated starting state $s \in Q$,
- A set $F \subseteq Q$ of accepting states.

Example:

$$Q = \{\text{even}, \text{odd}\}$$

δ :

	0	1
even	odd	even
odd	even	odd

$$s = \text{even}$$

$$F = \{\text{even}\}$$

Running a finite automaton

DFAs are dead easy to implement and efficient to run. We don't need much more than a two-dimensional array for the transition function δ. Given an input string x it is easy to follow the unique path determined by x and so determine whether or not the DFA accepts x.

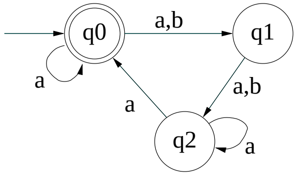
It is by no means so obvious how to run an NFA over an input string x. How do we prevent ourselves from making incorrect nondeterministic choices?

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

Executing an NFA: example

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

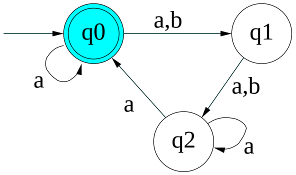
We illustrate with the running example below.



String to process: aba

Stage 0: initial state

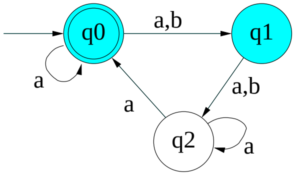
At the start, the NFA *can only be* in the initial state q0.



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

Stage 1: after processing ‘a’

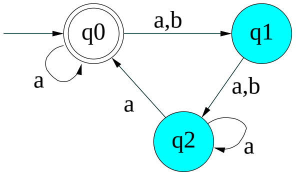
The NFA could now be in either q0 or q1.



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

Stage 2: after processing ‘ab’

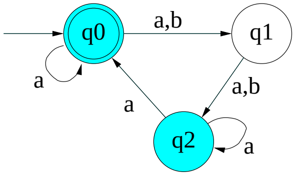
The NFA could now be in either q1 or q2.



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.

Reference material

- Kozen chapters 3 and 5.
- Jurafsky & Martin section 2.2 (rather brief).