

Finite State Machines (mostly revision)

Informatics 2A: Lecture 3

John Longley

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

23 September, 2011

- 1 Languages and Finite State Machines
 - What is a 'language'?
 - Regular languages and finite state machines: recap
- 2 Some formal definitions
 - Deterministic finite automata
 - Non-deterministic finite automata
- 3 DFA minimization
 - The problem
 - An algorithm for minimization

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

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 b ab cab $bacca$ $ccccccc$

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.

$$L_1 = \{a, aa, ab, abbc\}$$

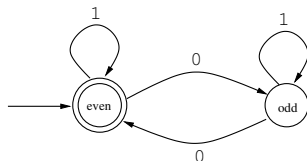
$$L_2 = \{axb \mid x \in \Sigma^*\}$$

$$L_3 = \{a^n b^n \mid n \geq 0\}$$

- Regular expressions (see Lecture 5).
- Formal grammars (see Lecture 9 onwards).
- 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 state machines



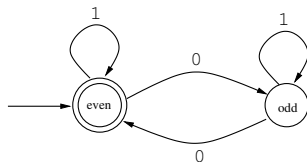
This is an example of a **finite state machine** over $\Sigma = \{0, 1\}$.

At any moment, the machine is in one of 2 **states**. For any state, and any symbol in Σ , there's just one 'new 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) may be 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 $L \subseteq \Sigma^*$.

Finite state machines: Clicker question(?)



For the finite state machine shown here, which of the following strings is *not* legal (i.e. not accepted)?

- 1 ϵ
- 2 101
- 3 1010
- 4 11

Deterministic finite automata (DFAs)

Formally, a **DFA** with alphabet Σ consists of:

- A **finite** set Q of **states**,
- A **transition function** $\delta : Q \times \Sigma \rightarrow Q$,
- A 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

$$s = \text{even}$$

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

The language associated with a DFA

Suppose $M = (Q, \delta, s, F)$ is a DFA with alphabet Σ .

We can define a **many-step transition function** $\widehat{\delta} : Q \times \Sigma^* \rightarrow Q$ by

$$\begin{aligned}\widehat{\delta}(q, \epsilon) &= q \\ \widehat{\delta}(q, xu) &= \delta(\widehat{\delta}(q, x), u)\end{aligned}$$

for $q \in Q$, $x \in \Sigma^*$, $u \in \Sigma$.

Example: $\widehat{\delta}(\text{odd}, 101) = \text{even}$.

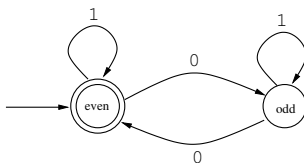
This enables us to define the **language accepted by** M :

$$\mathcal{L}(M) = \{x \in \Sigma^* \mid \widehat{\delta}(s, x) \in F\}$$

We call a language $L \subseteq \Sigma^*$ **regular** if $L = \mathcal{L}(M)$ for **some** DFA M .
Regular languages will occupy us for the next five lectures.

Example

Let M be the DFA shown earlier:



Then $\mathcal{L}(M)$ can be described directly as follows:

$$\mathcal{L}(M) = \{x \in \Sigma^* \mid x \text{ contains an even number of 0's}\}$$

More examples

Which of these three languages do you think are regular?

$$L_1 = \{a, aa, ab, abbc\}$$

$$L_2 = \{axb \mid x \in \Sigma^*\}$$

$$L_3 = \{a^n b^n \mid n \geq 0\}$$

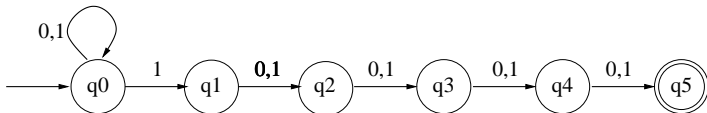
If they are, can you design a DFA that shows this?

If not, can you see why not? (We'll revisit this in Lecture 8.)

Aside: 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 .) So it's worth knowing when some task *can* be performed by a DFA.

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 q_0 , and none from q_5 .

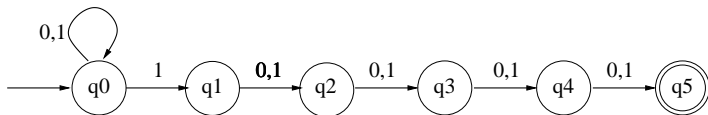
NFAs are useful because ...

- 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 (see next lecture).

The language accepted by an NFA

The language associated with an NFA is defined to consist of all strings that **might** be accepted by it (i.e. 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!)

From this, we can define a **many-step transition relation**

$\widehat{\Delta} \subseteq Q \times \Sigma^* \times Q$:

$$(q, \epsilon, q') \in \widehat{\Delta} \quad \text{iff} \quad q' = q$$

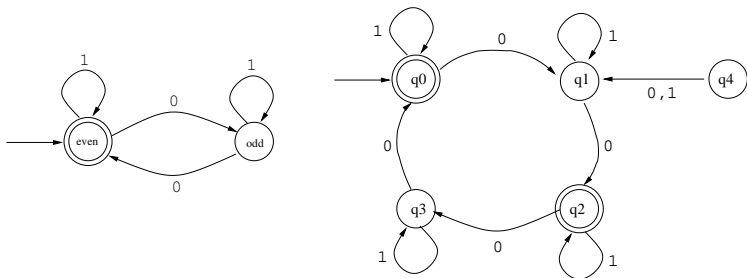
$$(q, xu, q') \in \widehat{\Delta} \quad \text{iff} \quad \exists q''. (q, x, q'') \in \widehat{\Delta} \ \& \ (q'', u, q') \in \Delta$$

The language accepted by the NFA is then

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

The Minimization Problem

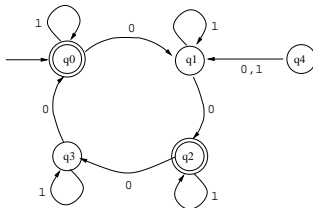
Think about DFAs again. Many different DFAs can give rise to the same language, e.g.:



This raises the question: Is there a **'best'** or **'smallest'** choice of DFA for a given regular language?

DFA minimization

Suppose $M = (Q, \delta, s, F)$ is the following DFA:



Informally, we can perform the following steps to 'reduce' M :

- Throw away **unreachable** states (in this case, q_4).
- Squish together **equivalent** states, i.e. states q, q' such that

$$\forall x \in \Sigma^*. \widehat{\delta}(q, x) \in F \Leftrightarrow \widehat{\delta}(q', x) \in F$$

In this case, q_0 and q_2 are equivalent, as are q_1 and q_3 .

Let's write $\text{Min}(M)$ for the resulting reduced DFA. In this case, $\text{Min}(M)$ is essentially the two-state machine on the previous slide.

Properties of minimization

The minimization operation on DFAs has the following pleasing properties:

- $\mathcal{L}(\text{Min}(M)) = \mathcal{L}(M)$.
- $\text{Min}(M) \cong \text{Min}(M')$ **if and only if** $\mathcal{L}(M) = \mathcal{L}(M')$.
- (Consequence.) $\text{Min}(\text{Min}(M)) \cong \text{Min}(M)$.
- $\text{Min}(M)$ is the **smallest** DFA (in terms of number of states) with the same language as M .

Here ' \cong ' means the two DFAs are **isomorphic**: that is, the same apart from a possible renaming of states.

So up to isomorphism, minimization gives a 'best', or **standard**, choice of a DFA for a given regular language.

For a formal treatment of minimization, see Kozen chapters 13–16.

Minimization in practice

Let's look again at our definition of **equivalence** of states:

$$q \approx q' \quad \text{iff} \quad \forall x \in \Sigma^*. \widehat{\delta}(q, x) \in F \Leftrightarrow \widehat{\delta}(q', x)$$

This is fine as an abstract **mathematical** definition of equivalence, but it doesn't seem to give us a way to **compute** which states are equivalent: we'd have to 'check' infinitely many strings $x \in \Sigma^*$.

Fortunately, there's an actual **algorithm** for DFA minimization that works in reasonable time.

This is useful in practice: we can specify our DFA in the most convenient way without worrying about its size, then minimize to a more 'compact' DFA to be implemented e.g. in hardware.

An algorithm for minimization

First eliminate any unreachable states (easy).

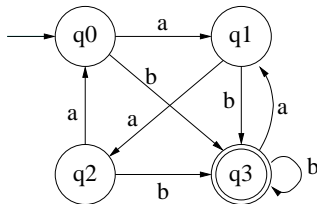
Then create a table of all possible pairs of states (p, q) , initially **unmarked**. (E.g. a two-dimensional array of booleans, initially set to false.) We **mark** pairs (p, q) as and when we discover that p and q *cannot* be equivalent.

- 1 Start by marking all pairs (p, q) where $p \in F$ and $q \notin F$, or vice versa.
- 2 Look for unmarked pairs (p, q) such that for some $u \in \Sigma$, the pair $(\delta(p, u), \delta(q, u))$ is marked. Then mark (p, q) .
- 3 Repeat step 2 until no such unmarked pairs remain.

If (p, q) is still unmarked, can collapse p, q to a single state.

Clicker question

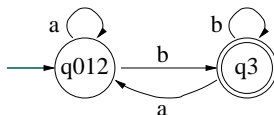
Apply minimization to the following DFA over $\{a, b\}$.



How many states does the minimized DFA have? 1, 2, 3 or 4?

Solution

The minimized DFA has just **2 states!** q_0 , q_1 and q_2 can all be identified, but clearly q_3 must be kept distinct from these.



Notice that the corresponding language consists of *all strings ending with b*.

Reading

Relevant reading:

- DFAs and regular languages: Kozen chapter 3; J & M section 2.2.
- NFAs: parts of Kozen chapters 5 and 6; J & M section 2.2.4–2.2.5.
- Minimization: Kozen chapters 13–16.

Next time: Are NFAs really 'more powerful' than DFAs? (See rest of Kozen chapters 5 and 6.)