Informatics 1 - Computation & Logic: Tutorial 6

Computation: Introduction to Finite State Machines

Week 8: 9 - 13 November 2015

Please attempt the entire worksheet in advance of the tutorial, and bring with you all work, including (if a computer is involved) printouts of code and test results. Tutorials cannot function properly unless you do the work in advance.

You may work with others, but you must understand the work; you can't phone a friend during the exam.

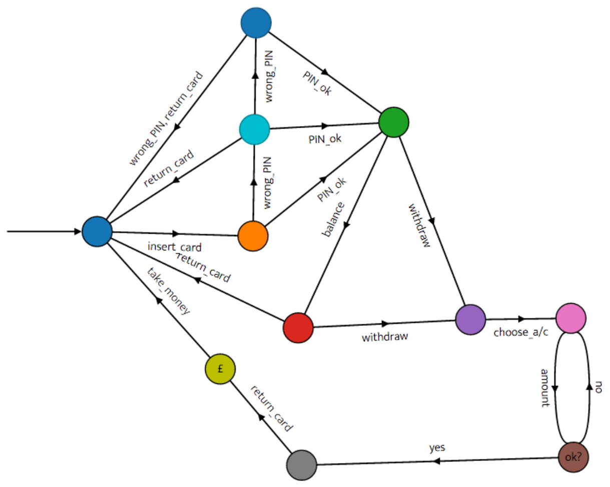
Assessment is formative, meaning that marks from coursework do not contribute to the final mark. But coursework is not optional. If you do not do the coursework you are unlikely to pass the exams.

Attendance at tutorials is **obligatory**; please let your tutor know if you cannot attend.

For this tutorial you may find it helpful to use Matthew Hepburn's online FSM tool.

1. The ATM design at
   `http://homepages.inf.ed.ac.uk/s1020995/atm.html`
   is somewhat unforgiving. If you enter the wrong PIN it keeps your card.

   Modify the machine so that it gives you three chances to enter the right PIN, before keeping the card, and so it allows you to get the card back after either of the first two failures, if you want to.



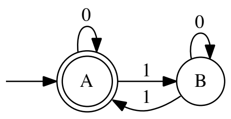
   For the remainder of this tutorial, you will probably find it helpful to use
   `http://homepages.inf.ed.ac.uk/s1020995/create.html`

   You can toggle each tool on and off by clicking it. When no tool is selected you can test your FSM with different inputs - and step through an input string (you can step backwards, as well as forwards) If you have problems using the tool please start a thread on Piazza.

2. Consider the following Finite State Machine (FSM) over the (input) alphabet $\{0, 1\}$:

   There are just two states, called state A and state B. State A is the initial state. State A is the only accept state. There are transitions labelled '0' from state A to itself and from state B to itself. There are transitions labelled '1' from state A to state B and from state B to state A.

   (a) Draw this machine.



   (b) Which of the following strings are accepted by this machine? (Y/N)

ϵ	Y
0	Y
1	N
00	Y
01	N
10	N
11	Y
101	Y
010	N
0011	Y
1011	N
1010	Y
10001	Y
11011000101	Y

   (c) What property is shared by all the strings which are accepted by this machine?

   They contain an even number of 1's.

(d) Does the machine accept all strings with this property?

Yes.

(e) What do the two states 'mean'?

State A means that the machine has seen an even number of 1's so far. State B means that it has seen an odd number of 1's so far.

(f) Fill out the following table so that the entry in row x column y contains the names of all states that can be reached from state x by a transition labelled y in the above machine:

	0	1
state A	A	B
state B	B	A

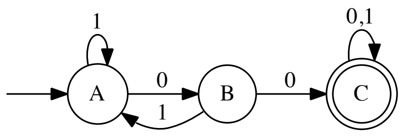
Use the table to decide whether or not the machine is deterministic, and explain why.

Yes, it is deterministic since every cell contains exactly one state transition.

3. Consider the following FSM over alphabet $\{0, 1\}$:

   There are just three states — state A, state B and state C. State A is the initial state. State C is the only accept state. There are transitions labelled 0 *from state A to state B, from state B to state C, and from state C to itself. There are transitions labelled* 1 *from state A to itself, from state B to state A, and from state C to itself.*

   (a) Draw this machine.

   (b) What language is defined by this machine? (Think about what the states 'mean'.)

   The set of all and only strings over $\{0, 1\}$ containing two successive 0's. State A means that the machine hasn't yet seen two successive 0's and the last symbol was a 1. State B means that the machine hasn't yet seen two successive 0's and the last symbol was a 0. State C means that the machine has seen two successive 0's.

   (c) Fill out its transition table:

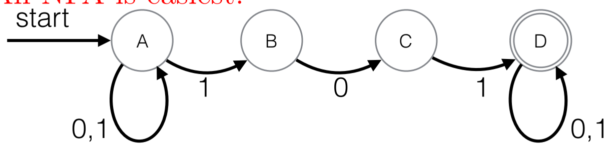
	0	1
state A	B	A
state B	C	A
state C	C	C

   Is the machine deterministic? Why?

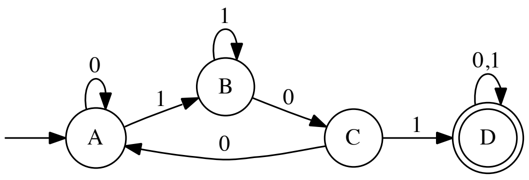
   Yes, it is deterministic since every cell contains exactly one state.

4. Draw a finite state machine over alphabet $\{0, 1\}$ which accepts all and only those strings of which the string 101 is a substring.

An NFA is easiest.



Here is a DFA:



State A: I haven't yet seen 101, and the last thing I saw was 00 (if at least 2 symbols have been seen). State B: I haven't yet seen 101 and the last thing I saw was 1. State C: I haven't yet seen 101 and the last thing I saw was 10. State D: I've seen 101! Transitions labelled 0 from A to A, from B to C, from C to A, and from D to D. Transitions labelled 1 from A to B, from B to B, from C to D, and from D to D.

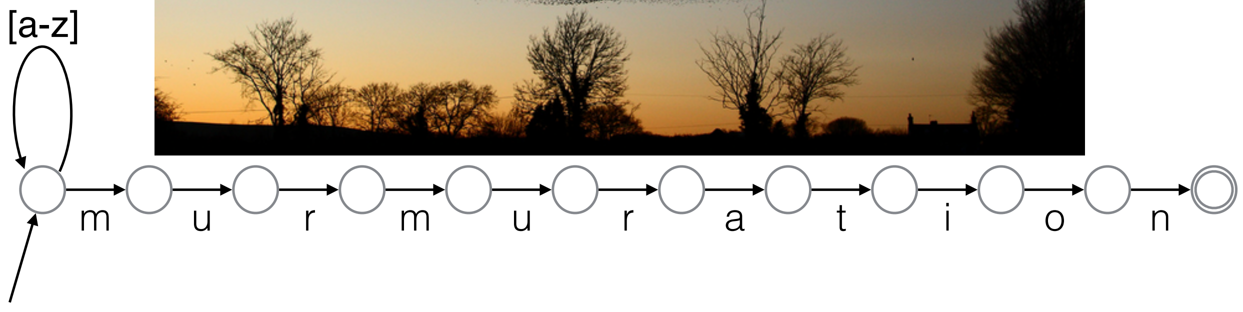
5. Sketch a finite state machine over alphabet $\{0, 1\}$ which accepts all and only those strings consisting of a 42 zeros followed by *the same number* of ones?

For any given number n, for example, 42, this is simple.

6. Can you design a more general finite state machine over alphabet $\{0, 1\}$ which accepts all and only those strings consisting of some number of 0's followed by *exactly the same* number of 1's?

We cannot produce a single FSM that works for all n. This is not possible because FSMs only have finitely many states.
Let M be a DFA with $k(> 0)$ states. Let $n = 2 \times k$. Then any trace of the states $(x_1, \dots, x_n)$ visited for an input of n zeros in succession, must visit some state more than once, say $s = x_p = x_q$, where $p < q$. Now consider the traces for the following two input sequences: let a_i be the trace of states visited for the sequence of q zeros followed by q ones; let b_j be the trace of states visited for the sequence of p zeros followed by q ones. By construction $a_q = b_p = s$. Beyond this state, both inputs continue with q ones, so, by induction on i, $a_{q+i} = b_{p+i}$ for all $i \leq q$. In particular, the two traces end in the same state, but one input has equal numbers of ones and zeros, while the other does not.

7. A flock of starlings is called a **murmuration**.



The NFA shown recognises strings, in the alphabet [a-z], that end with this word. The transition from the start state to itself can be taken with any letter of the alphabet.

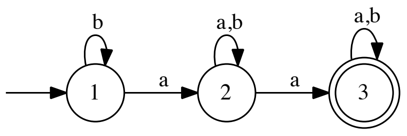
Construct a DFA that recognises these strings.

	m	u	r	a	t	i	o	n	[ˆ muration]
0	0,1	0	0	0	0	0	0	0	0
0,1	0,1	0,2	0	0	0	0	0	0	0
0,2	0,1	0	0,3	0	0	0	0	0	0
0,3	0,1,4	0	0	0	0	0	0	0	0
0,1,4	0,1	0,2,5	0	0	0	0	0	0	0
0,2,5	0,1	0	0,3,6	0	0	0	0	0	0
0,3,6	0,1,4	0	0	0,7	0	0	0	0	0
0,7	0,1	0	0	0	0,8	0	0	0	0
0,8	0,1	0	0	0	0	0,9	0	0	0
0,9	0,1	0	0	0	0	0	0,10	0	0
0,10	0,1	0	0	0	0	0	0	0,11	0
0,11	0,1	0	0	0	0	0	0	0	0

Searching for strings in a long sequence, for example, searching for words in text, or genes in DNA sequences, is an important problem.

One of the best algorithms for many applications, and one of the most commonly used, was invented by Bob Boyer and J Moore, in Edinburgh, in the 1970s. If you want to learn more you can look at J Moore's page
`http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/`
although he works in Texas he is still a frequent visitor to the Forum.

8. Consider the Finite State Machine in the diagram:



(a) Draw its transition table:

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

(b) Explain why this is a non-deterministic automaton (NFA)?

It is a non-deterministic FSM because there are two possible transitions from node 2 given the a input symbol.

(c) Give a simple english description of the language accepted by this FSM.

A finite sequence of as and bs that contains at least two as.

(d) Give a regular expression for the language accepted by this FSM.

$b^*a(a|b)^*a(a|b)^*$

(e) Construct an equivalent DFA.

	a	b
1	2	1
2	$2,3$	2
$2,3$	$2,3$	$2,3$

9. Draw a finite state acceptor which accepts an infinite set of strings each of which consists of an *odd number of* 0*s* and an *even number greater than* 0 *of* 1*s* . To be more precise, if string s is accepted by the machine, then s must:

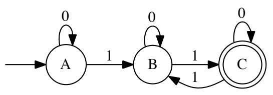
   (a) consists of an odd number of symbols;

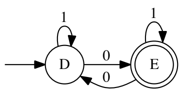
   (b) contain only 0s and 1s;

   (c) contain an odd number of 0s; and,

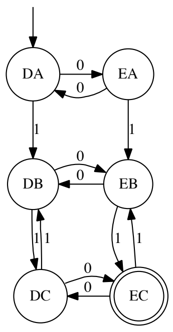
   (d) contain an even number of 1s greater than 0.

   To create such acceptor, first create an acceptor for the even number (> 0) of 1s, then another acceptor for the odd numbers of 0s, and then put the two acceptors together using the correct operator. Explain your choice of the operator.





The first accepts an even number of 1s greater than 0, the second an odd number of 0s.



Intersection: a string x will be accepted by the intersection of the two machines only if it is accepted by both machines:

- 1100 is accepted by the first (even 1s), but not by the second (odd 0s)
- 1000 is accepted by the second, not by the first
- 11000 is accepted by both, and it is accepted by the intersection

10. (a) Do DFAs and NFAs recognise the same languages?

Yes, they recognise the same class of languages, the class of *regular languages*.

(b) What advantages do NFAs have over DFAs, and vice-versa?

The determinism of DFAs makes it easy to implement them in software or hardware, whereas the reverse is true for NFAs - their nondeterminism makes them difficult to implement directly. However, NFAs have the advantage that they are easier to construct). As we have seen, we can build build complex NFAs in a simple piecewise fashion — fitting smaller FSMs together. NFAs are also useful for modelling real systems with inherent nondeterminism.

This tutorial exercise sheet was written by Paolo Besana, and revised by Thomas French, Areti Manataki, and Michael Fourman. Send comments to `Michael.Fourman@ed.ac.uk`