Informatics 1 - Computation & Logic: Tutorial 7

Computation: Non-Deterministic FSMs and Regular Expressions

Week 9: 13 - 17 November 2017

Please attempt the entire worksheet in advance of the tutorial, and bring all work with you. Tutorials cannot function properly unless you study the material in advance. Attendance at tutorials is **obligatory**; please let the ITO know if you cannot attend.

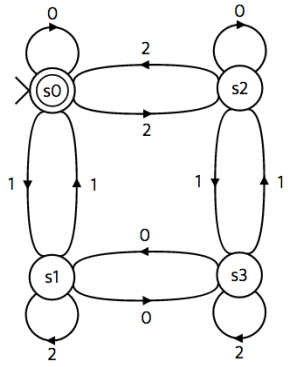
You may work with others, indeed you should do so; but you must develop your own understanding; you can't phone a friend during the exam. If you do not master the coursework you are unlikely to pass the exams.

You may find it useful to refer to the FSM Workbench question set which accompanies this tutorial at homepages.inf.ed.ac.uk/s1020995/tutorial6.

Use the workbench to check your working, but ultimately you should aim to be able to answer simple questions of this type working with just pencil and paper.

This tutorial exercise sheet was written by Matthew Hepburn and Dagmara Niklasiewicz, with additions from Michael Fourman. Send comments to `Michael.Fourman@ed.ac.uk`

In Lecture 14 we introduced this example of a DFA encoded in Haskell:

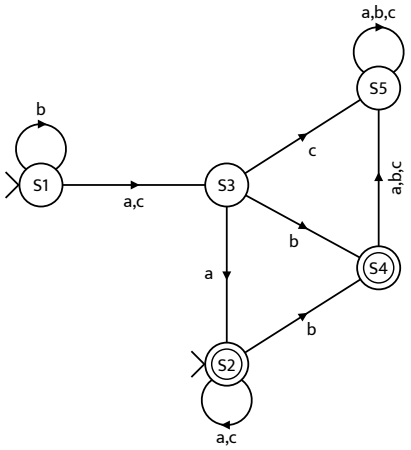



```
      s0 []  =  True          s2 []  =  False
s0 ('0':xs)  =  s0 xs    s2 ('0':xs)  =  s2 xs
s0 ('1':xs)  =  s1 xs    s2 ('1':xs)  =  s3 xs
s0 ('2':xs)  =  s2 xs    s2 ('2':xs)  =  s0 xs

      s1 []  =  False         s3 []  =  False
s1 ('0':xs)  =  s3 xs    s3 ('0':xs)  =  s1 xs
s1 ('1':xs)  =  s0 xs    s3 ('1':xs)  =  s2 xs
s1 ('2':xs)  =  s1 xs    s3 ('2':xs)  =  s3 xs
```

The code consists of four mutually recursive functions, one for each state of the machine. To check whether a string, `input`, is accepted by the machine we evaluate `s0 input` (because $S0$ is the starting state of our DFA), which returns the Boolean answer, `True` or `False`.
Check that you understand how this code relates to the DFA. How would you modify the code if the accepting state were $S3$? How could you modify the code to return the number of the final state of the trace generated by any input string?

1. Consider the finite state machine in the diagram below.



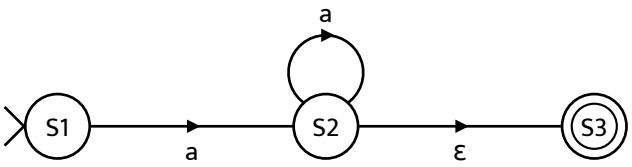
Input	Is Accepted?
⟨ ⟩	
b	
aa	
ba	
abaab	
acaca	
aaab	
bbbcb	
cacba	

   (a) For each input sequence in the table above, record whether it is accepted by the FSM.

   (b) Is the FSM deterministic? Justify your answer.

   (c) Implement a Haskell function to check your answers (by implmenting a suitable DFA).

2. This NFA over the alphabet {a} uses an ε transition.

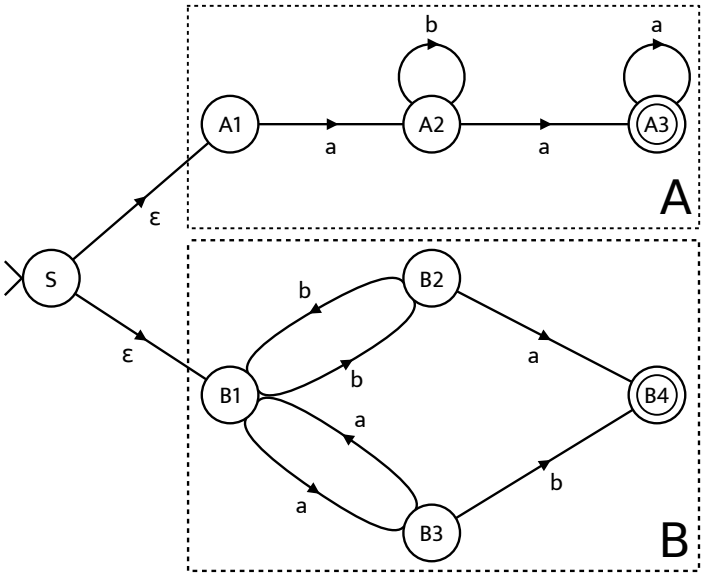


   (a) Describe the language accepted by this machine in words.

   (b) Describe the language accepted by this machine using a regular expression.

   (c) Design a deterministic machine that accepts the same language as this machine.

3. ε-transitions provide a simple way of combining FSMs. The machine below has been composed from two machines A and B, which had initial states A1 and B1.



   (a) Considering machines A and B separately, give a regular expression which describes the language they accept.

   (b) Considering the whole machine, give a regular expression which describes the language the machine accepts.

(c) L_A and L_B are the languages accepted by machines A and B. Give an expression relating L_A and L_B to L, where L is the set of input accepted by the whole machine.

(d) Construct and test a Haskell implementation of an equivalent DFA.

4. Consider the regular expression $\texttt{ab}(\texttt{a}|\texttt{b})^*$

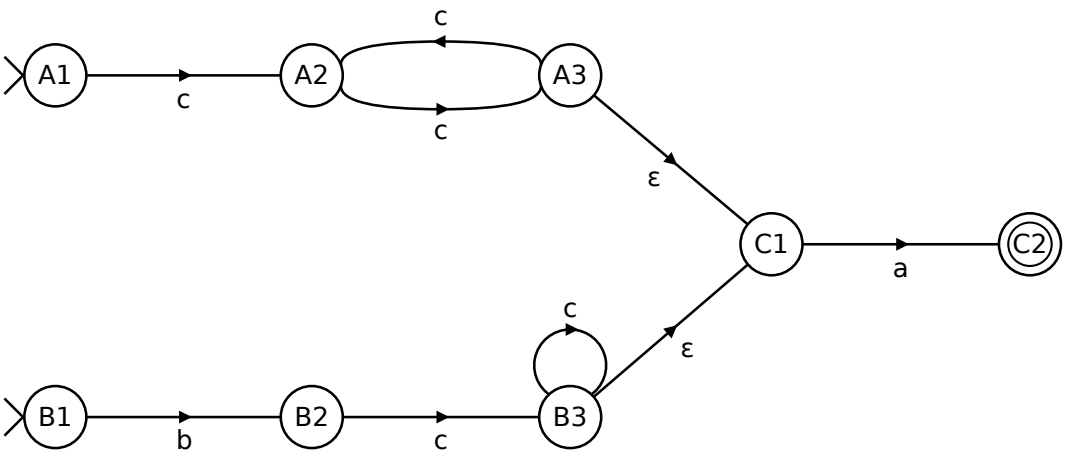
(a) Describe in words the language that the expression matches. Include two examples of strings that are matched.

(b) Design a finite state machine that accepts that language.

(c) Building on your answer to (b), design a finite state machine that accepts $\texttt{ab}(\texttt{a}|\texttt{b})^*\texttt{bb}^*(\texttt{aa})^*$.

(d) Construct and test a Haskell implementation of an equivalent DFA.

5. Consider this NFA over the alphabet $\{\texttt{a}, \texttt{b}, \texttt{c}\}$.

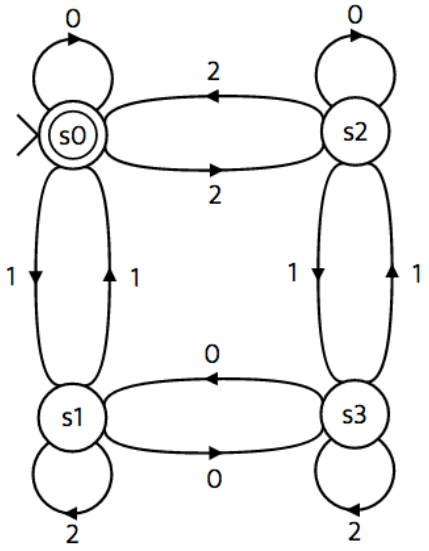


(a) Describe, both in words and with a regular expression, the language accepted by this machine. Hint: think about the sequences that end in A3 and B3.

(b) Design a DFA that accepts the same language.

(c) Are there any NFAs that cannot be converted into an equivalent DFA?

6. Consider this DFA over the alphabet $\{0, 1, 2\}$. It should be familiar.



(a) Describe, in words, the language accepted by this machine. Hint: Your description in words should refer to ternary numbers.

(b) Replace each transition labelled 0 by a transition labelled ε, between the same two states. The resulting automaton is not a DFA. (Why not?)

i. Construct an equivalent DFA.

ii. Describe, both in words and with a regular expression, the language accepted by this machine.

(c) Next, replace each transition (of the original machine) labelled 1 by a transition labelled ε, between the same two states.

i. Again, construct an equivalent DFA, and, ii, describe the language it accepts.

(d) Repeat the exercise replacing each transition (of the original machine) labelled 2 by a transition labelled ε, between the same two states.

i. Construct an equivalent DFA,

ii. describe the language it accepts.

(e) BONUS QUESTION: Give a regular expression that describes the language accepted by the original machine. Test your answer using the grep utility.

This bonus question goes somewhat beyond the call of duty. Feel free not to attempt it. That said, by the end of week 7 you should have all the tools required to complete it. If you do choose to try it, I suggest you use cut and paste in some suitable editor to make, and keep track of the algebraic substitutions that are required.

7. Use the FSM workbench to construct a machine that accepts ternary strings with an even number of 1s and an even number of 2s (and any number of 0s) that represent a number that is not a multiple of four.

   Hint: *start by constructing a machine that accepts ternary strings S that satisfy at least one of the following three conditions:*

   (a) S includes an odd number of 1*s,*

   (b) S includes an odd number of 2*s,*

   (c) S represents a multiple of 4*.*

   You are asked to build a machine that accepts strings that satisy none of these conditions.

Tutorial Activity

String matching algorithms try to find a place where one or several strings (also called patterns) are found within a larger string or text.
String matching algorithms play a vital role in a host of applications ranging, for example, from the detection of plagiarism, to the analysis of protein and DNA sequences in Computational Biology.
In computational biology DNA – the stuff of which the double helixes that carry genetic information are made – consists of two chains of bases.[1] There are four types of base: cytosine (C), guanine (G), adenine (A) or thymine (T). The two chains are matched together A–T and C–G, so one chain determines the other. For example, if one chain is AATCAG the other must be TTAGTC.
In real-life problems biologists may search for patterns with thousands of letters in genomes with billions of base-pairs.
For this exercise you should consider strings on the alphabet with four symbols AGCT.

1. (a) Draw an NFA that will accept any string that includes the pattern CACAT. Name each state with the string it is looking for – a string that has a trace from that state to an accepting state. So, the starting state will have the name CACAT and the accepting state has the name ϵ.

   (b) List the reachable states of the NFA, as constructed by the subset procedure.

2. (a) Draw an NFA that will accept any string that includes a close match to the pattern CACAT, where a close match is either exactly this string, or a string CACATof the same length that differs from the given pattern at at most one letter. Hint: some states will be looking for an exact match, and some for a close match.

[1] Biology is infinitely complex. This is a simplified account for the purposes of this exercise.

3. If time permits, derive a Haskell program inplementing the DFA for the first of these tasks. Each state of the DFA corresponds to a set of states of the NFA – each state of the DFA is looking for any one of the strings corresponding to the NFA states it includes. If we fail to find the first letter of the shortest string we are looking for, we can fall back to look for the next shortest string.