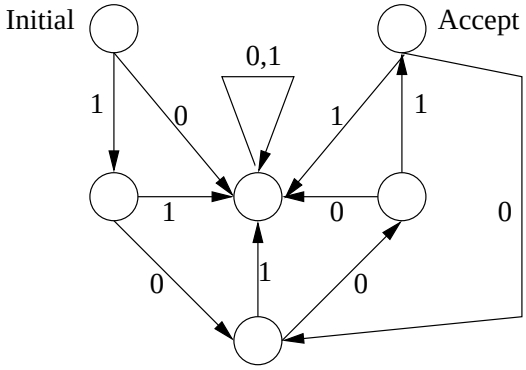
Worksheet 5 (Tutorial)

Introduction

This worksheet is about finite state machines (Lecture 9 and 10). FSMs have many applications throughout computer science.

Simple FSMs

The following transition diagram defines an FSM. The initial state and the accepting state are indicated. The input alphabet is $\{0, 1\}$, i.e. the machine receives a sequence of binary digits.



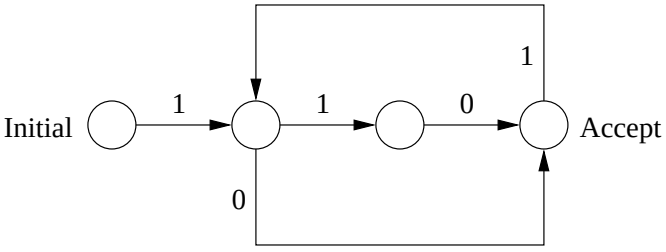
A state from which all transitions lead back to itself, for example the state in the centre of the diagram, is called an *absorbing* state.

1. Which of the following sequences are accepted?
   (a) 101
   (b) 1001
   (c) 10011
   (d) 1001001
2. Describe the set of all sequences accepted by this FSM.
3. Draw the transition diagram for an FSM which accepts all sequences of the form: 011, repeated any number of times.

More Complex FSMs

The following transition diagram defines another FSM. Again the input alphabet is $\{0, 1\}$. If any state has a missing transition (for example, the initial state has no transition for 0),

this means that there should be a transition to an absorbing state (not shown). This is a way of simplifying the diagram.



4. Which of the following sequences are accepted?

   (a) 110

   (b) 10110

   (c) 1010

   (d) 111

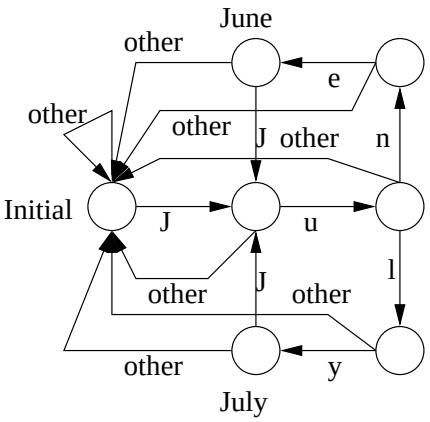
5. Draw the full version of the transition diagram, including an absorbing state and showing all of the missing transitions.

6. Describe the set of all sequences accepted by this FSM.

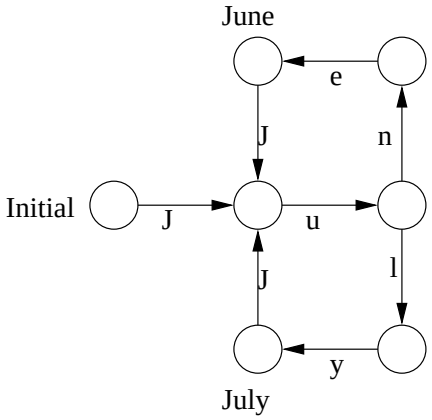
7. Draw the transition diagram for an FSM which accepts all sequences of the form: any number (at least one) of blocks, where a block is either 101 or 01.

Text Searching

Imagine that we want to search a text file for occurrences of the words "June" and "July". We can do it with an FSM. The input alphabet consists of all upper and lower case letters. The accepting state is reached by any sequence of letters ending with "June" or "July". Here is the transition diagram. The label "other" means that the transition corresponds to all letters except for those which label other transitions from the same state. The states labelled "June" and "July" are accepting states showing when one of the words has been found.



To make the diagram less cluttered, we could remove all of the "other" transitions. Just remember that from every state there should be an "other" transition going back to the initial state.



This idea is used in programming language compiler systems (for example, the Ada compiler in the lab) to recognise keywords in the program file. The first step in translating an Ada program into machine language is to convert the file from a sequence of characters into a sequence of "tokens": "if", "then", "end", "array", and so on. This is done by implementing an FSM, giving it the characters from the file, one by one, and taking note of every time an accepting state is reached.

8. The diagram above is not complete. Find an example of an input sequence which contains the word "June" but which is not detected by the FSM shown.

9. Which extra transitions are needed in order to correct this problem?

10. Draw the transition diagram for a similar FSM which recognises the words "carrot", "cat" and "dog".

11. How would you design an Ada program to detect "June" and "July" in a file? The output should be a sequence of lines of the form "June at 20", meaning that the 20th, 21st, 22nd and 23rd characters in the file form the word "June".