Programming Languages and Computation

Week 4: Finite Automata

This problem sheet is about constructing examples of finite automata to recognise various languages.

* 1. For each of the automata in Figure 1, give (i) whether or not the word 12212 is accepted, and (ii) the language recognised by the automaton.

Solution

(a) (i) Rejected, (ii) the language is: $\{1w1 \mid w \in \{2\}^*\}$

(b) (i) Rejected, (ii) the language is $\{w \mid w$ contains 322 as a substring$\}$

(c) (i) Accepted, (ii) the language is $\{w \mid w \neq 121v$ for any $v\}$

* 2. Draw the diagram of the following automata:

(a) $(\{e, o\}, \{0, 1\}, \{((e, 0), o), ((e, 1), o), ((o, 0), e), ((o, 1), e)\}, e, \{e\})$

(b) $(Q, \{0, 1\}, \delta, q_0, Q)$ where $Q = \{q_0, q_1, q_2, q_3\}$ and δ is specified by:

$$\begin{array}{rcl} \delta(q_0, 0) & = & q_0 \\ \delta(q_0, 1) & = & q_1 \\ \delta(q_1, 0) & = & q_2 \\ \delta(q_1, 1) & = & q_3 \\ \delta(q_2, 0) & = & q_1 \\ \delta(q_3, 0) & = & q_3 \end{array}$$

(c) $(Q, \Sigma, \delta, q_0, Q)$ where:

- $Q = \{1, 2, 3, 4, 5\}$
- $\Sigma = \{a, b\}$
- $\delta = \{((i, a), i + 1) \mid 1 \leq i \leq 5\} \cup \{((j, b), j) \mid j$ is even$\}$
- $q_0 = 1$
- $F = \{1, 3, 5\}$

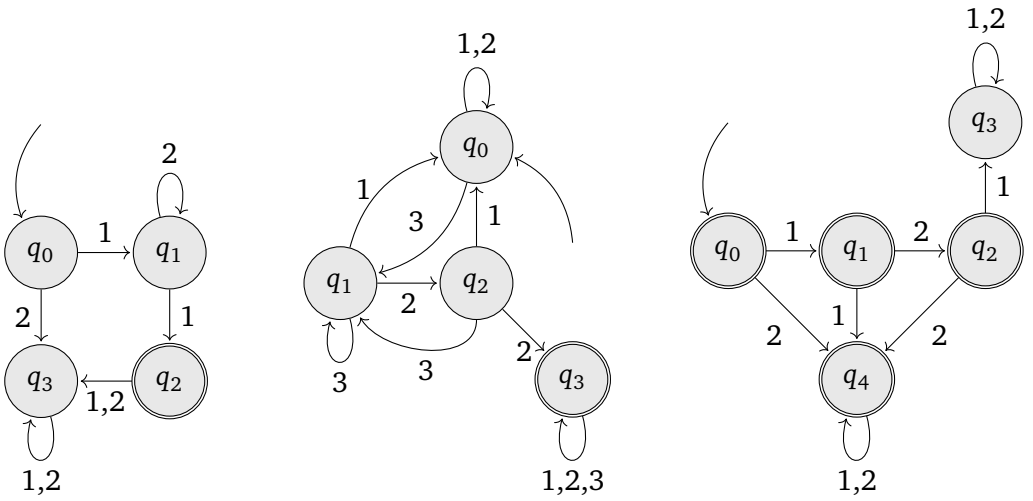
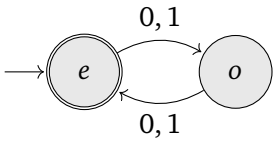


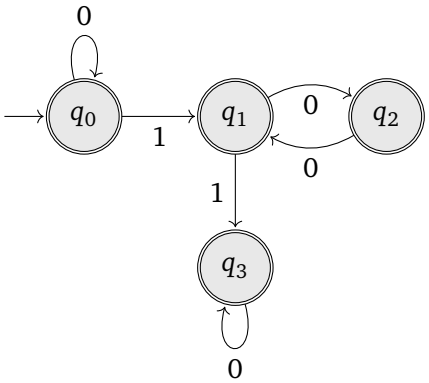
Figure 1: Automata used in Question 1.

Solution

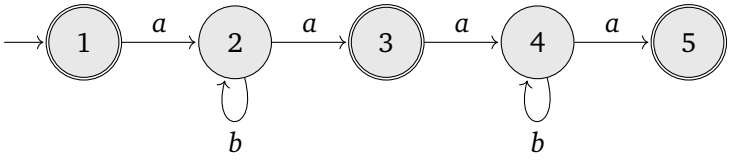
(a) Diagram:



(b) Diagram:



(c) Diagram:



* 3. Give finite automata to recognise each of the following, it suffices to draw the diagram of the automata:

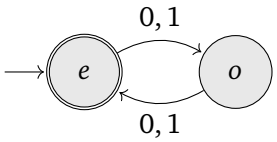
(a) The language of all strings over $\{0, 1\}$ such that each string has even length.

(b) The language of all strings over $\{0, 1\}$ such that each string has a number of 1s that is a multiple of three.

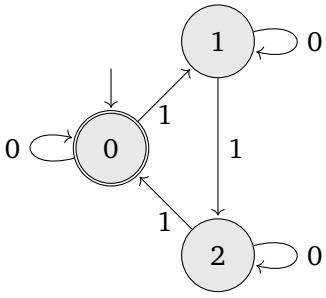
(c) The language of all strings over $\{0, 1\}$ such that each string has even length and a number of 1s that is a multiple of three.

Solution

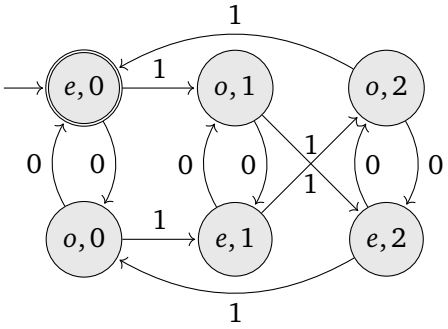
(a)



(b)



(c) In the following, the idea is that the automaton will be in a state labelled e if it has seen an even number of letters so far, and o otherwise. It will be in a state labelled $i \in \{0, 1, 2\}$ if it has seen a number of 1s whose remainder is i after dividing by 3.



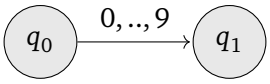
** 4. Construct a finite automaton to recognise Haskell floating point literals, e.g. `2.99`, `23.09e+34`, `0.12E-200`, `1.4e1`.

A general description is as follows. A *decimal literal* is a non-empty sequence of digits (0–9). A *floating point literal* is either:

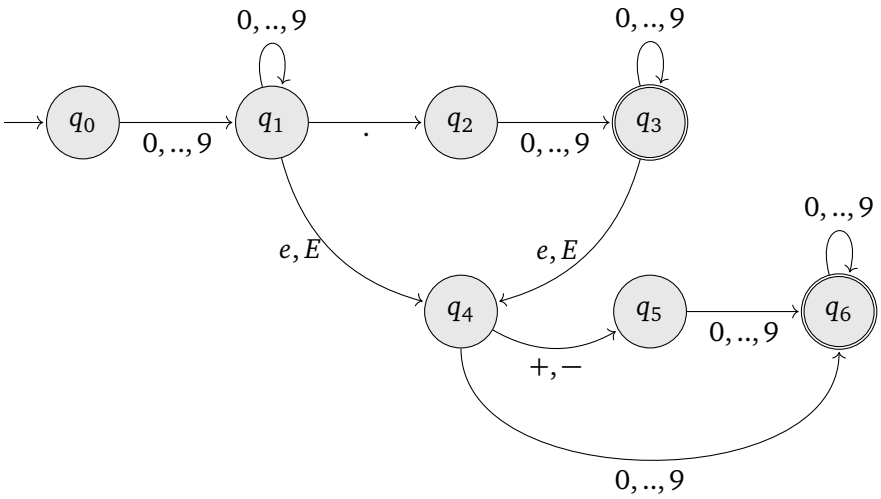
- a decimal literal followed by a decimal point followed by a decimal literal, optionally followed by an exponent
- or, a decimal literal followed by an exponent.

An exponent is either the character e or the character E; optionally followed by the character + or the character −; followed in all cases by a decimal literal.

Since this automaton is relatively complex, I suggest you abbreviate multiple transitions between a given pair of states with a single transition that has multiple labels. For example, 10 transitions labelled 0, 1, ..., 9 between states q_0 and q_1 could be written:



Solution



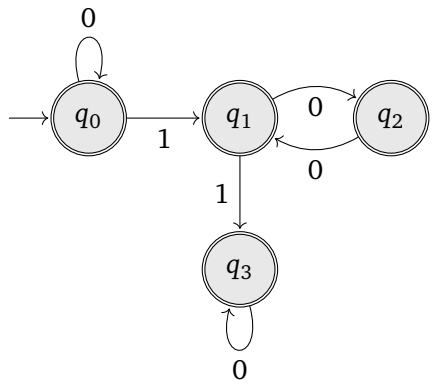
** 5. Construct a finite automaton to recognise the language of strings over {0, 1} such that any two occurrences of 1 are separated by an even number of letters.

Hint how many 1s can occur in a word of this language?

Solution

Some thought about the hint reveals that at most two 1s can occur in a word of this language. To see why, suppose s is a string containing at least two 1s and in which every pair of 1s is separated by an even number of letters. What does such a word look like? The word can be seen to have shape $u1v1w$, in which u doesn't contain any 1 and w doesn't contain any 1 and we allow that any of u, v or w could be empty. In other words, it contains at least two 1s and the first occurs after u and the last one is followed by w. Now, we know that the two 1s highlighted by this decomposition are separated by an even number of letters, i.e. v is of even length. Now suppose there are *more* 1s than just these two, so there is another 1 inside the v part. But the v part is of even length, so wherever this additional 1 occurs within v, there will be an odd number of letters between it and the first 1 in s or between it and the last 1 in s. This contradicts the assumption that all occurrences of 1 in s are separated by an even number of letters.

On the other hand, it is clearly possible to have words with 0, 1 or 2 occurrences of 1 that satisfy the requirement. So we design an automaton that recognises the language of words containing at most two occurrences of 1 separated by an even number of 0s.



The meaning of the states are:

- q_0: Read in word without any occurrence of 1
- q_1: Read in word containing a single occurrence of 1 and in which there are an even number of letters read since the 1.
- q_2: Read in word containing a single occurrence of 1 and in which there are an odd number of letters read since the 1.
- q_3: Read in word containing exactly two occurrences of 1 separated by an even number of letters.

A good way to think about designing finite automata is to think: "what should the states of the automaton represent?". This means attaching some informal meaning to each state in terms of the kinds of words that have been read at the point at which the automaton enters that state. Often, once you know what the states of the automaton represent, it is easy to fill in the transitions. The following question helps to illustrate this.

** 6. Construct an automaton that recognises the language of bit strings (strings over $\{0,1\}$) that represent numbers written in binary that are divisible by three. For example, 10010 should be accepted because it represents the decimal number 18 written in binary and this number is divisible by 3. However, 101 should not be accepted, because this is the binary representation of the number 5, which is not divisible by 3. The empty word ϵ can be considered as representing 0 and hence should be accepted.

Your automaton should have three states, labelled 0, 1 and 2, with meaning as follows:

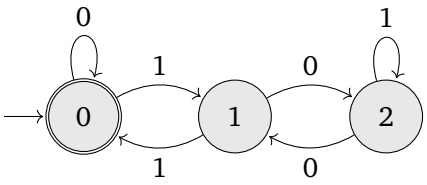
- When the automaton is in state 0, it has (so far) read in a bit string which represents a number divisible by 3, written in binary.

- When the automaton is in state 1, it has (so far) read in a bit string which represents a number which has remainder 1 when divided by 3.
- When the automaton is in state 1, it has (so far) read in a bit string which represents a number which has remainder 2 when divided by 3.

Use the meanings of the states in order to guide how you fill out the transitions of the automaton. Which is the initial state, which are the final states?

Hint For example, there is a transition from state 1 to state 0 on reading a 1, this is why. Suppose the automaton is in state 1 – this means it has so far read in a bit string, say w, which represents a number, say n, with remainder 1 when divided into 3. Now suppose it next reads in a 1, so the word read in so far is now $w1$, the number represented by this string is $2n+1$ because it arises by shifting w up one place (which corresponds to multiplying the value of w, which is n, by 2) and adding 1 on the end. A bit of modulo arithmetic tells you that if the remainder on dividing n by 3 is 1, then the remainder on dividing $2n+1$ by 3 is 0. For example: with $w = 111$ we have $n = 7$ so the automaton should be in state 1 after reading this word. Suppose it next reads 1, so the word is now $w1 = 1111$. This word represents $2n+1 = 2 \cdot 7 + 1 = 15$ and so the remainder is 0. Hence, the automaton should move to state 0.

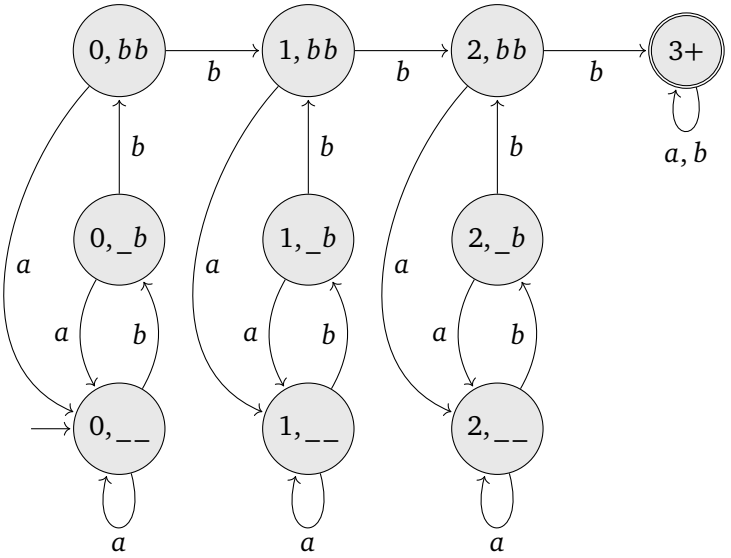
Solution



** 7. Make sure you have completed all parts of the Turtle compiler project: https://github.com/uob-coms20007/turtle

*** 8. Construct a finite automaton to recognise the language of strings over the alphabet $\{a, b\}$ containing at least three occurrences of three consecutive b, overlapping permitted (e.g. the word $bbbbb$ should be accepted).

Solution



The key to this automaton is to recognise that you have to remember *two* things at any point in time:

- The number of sequences of 3 consecutive letter b.
- For those states in which we have not yet seen at least 3 consecutive subsequences of b, we should additionally remember whether the two letters immediately preceding were both b, or (if not) was the immediately preceding letter a b, or is it neither of the above.

We write the former as a number 0, 1, 2 or 3+; and the latter as bb, $\_b$ and $\_\_$ respectively. The important thing is to know how many of the immediately preceding letters were b, so the initial state is associated with $\_\_$ as there have been no letters at all yet.

Hopefully the need to remember the first of these is quite clear, the second helps to track progress towards completing a subsequence. When we have overlapping subsequences, as in $bbbb$ - after reading the first three b we can note that we have read one sequence of three consecutive b and, since the preceding two letters (i.e. the second and third letters of the word) were also b, reading in one more b will give another complete subsequence.

**** 9. (Optional) Let Σ be the following set of binary vectors:

$$\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}$$

We use each word w over alphabet Σ to encode a pair of natural numbers, written $[\![w]\!]$, which is defined by the following recursive function:

$$\begin{aligned} [\![\epsilon]\!] &= (0,0) \\ [\![\binom{b_1}{b_2} \cdot w]\!] &= (2 * m + b_1,\ 2 * n + b_2) \\ &\qquad \textit{where } (m, n) = [\![w]\!] \end{aligned}$$

For example:

$$[\![\binom{0}{1}\binom{0}{0}\binom{1}{1}\binom{0}{1}]\!] = (4, 13)$$
$$[\![\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{1}{0}]\!] = (26, 3)$$
$$[\![\binom{1}{1}\binom{0}{1}\binom{0}{1}\binom{0}{0}\binom{0}{1}]\!] = (1, 23)$$

Construct a finite automaton to recognise the language of "multiplication by 3":

$$\{w \in \Sigma^* \mid \exists m \in \mathbb{N}.\ [\![w]\!] = (m, 3 * m)\}$$

Solution

To answer this question, it is best to think of multiplying by three as doing three binary additions of the given number in one go. This will require an extended notion of "carry bit" because adding three numbers together can result in carrying forward by two columns. The automaton needs only to keep track of this carry.