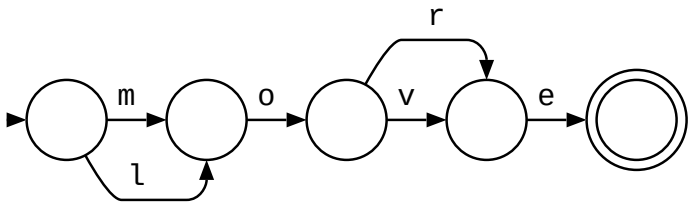
Computer Mathematics

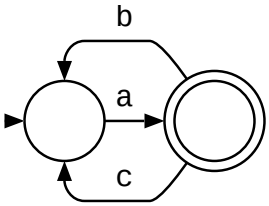
Week 11 Examples

1. What strings (sequences of symbols) are generated (accepted) by the following FSMs?

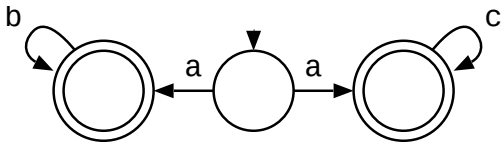
a)



b)



c)



2. For each set of strings, draw a FSM such that it accepts (or generates) at least the indicated strings.

a)
- bag
- beg
- big
- bog
- bug

b)
- bg
- big
- biig
- biiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiig

c)
- luck
- lurk
- lark
- lack

3. Draw a FSM that generates an even parity bit for an input sequence of binary digits. (The machine will be in one of two states, either 0 or 1. If the number of '1's seen so far is even, the machine will be in state 0. If the number of '1's seen so far is odd, the machine will be in state 1.)

4. Draw a FSM that accepts the following pattern:

- any number of (zero or more) '0's
- an odd number of '1's
- an odd number of '0's
- any number of (zero or more) '1's

5. Draw a state machine that controls a digital combination lock. To open the lock, four digits have to be entered in the correct order (e.g., 1 2 3 4) followed by the # key. If a mistake is made while entering the digits, all previous input should be discarded and the lock reset. Once open, pressing any digit or the # key closes and resets the lock.

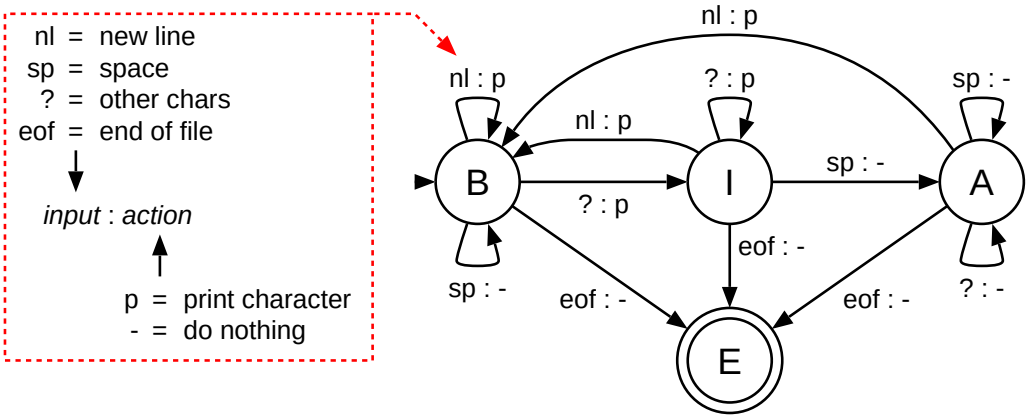
Draw the transition table for your state machine.

6. An elevator door has four states: closed, opening, open, and closing. There are two buttons labelled `open` and `close`, motors that can be commanded to open or close the doors, and two detectors that indicate when the doors are fully open or fully closed.

Design a state machine to control the elevator door. Be sure to properly handle the cases when a button is pushed while the doors are in the middle of opening or closing.

7. Design and write a program that uses a state machine to count and print the number of *words* in a file, where a *word* is defined as: one letter (`a` through `z`, `A` through `Z`, and _) followed by zero or more letters or digits (`0` through `9`). All other characters should be treated as separators between words.

8. Consider this 'first word printing' finite state machine:



This is an example of a *Mealy machine* (named after the person who first described it). In a Mealy machine the outputs (which in this case are the actions that print the current input character) are associated with the current state and the *current* input(s). The outputs (actions) are valid (executed) *before* (during) the transition to the next state; the outputs (actions) are therefore attached to the *transitions* between states.

There is another kind of finite state machine called a *Moore machine* (named after the person who first described a way to simplify Mealy machines). In a Moore machine the outputs depend only on the current state, or (equivalently) on the current state and the *previous* input(s). The ouputs (actions) are valid (executed) *after* a transition to the next state is complete; the ouputs (actions) are therefore attached to the *states* themselves, and are valid (executed) when their associated state becomes the current state.

Moore machines are simpler than Mealy machines, but the price paid for simplicity is that they typically require more states than an equivalent Mealy machine. For example, when leaving the start state B the Mealy machine shown above will print the NL character during the associated transition back to B, but it does *not* print the SP character when making a different transition that also leads back to B. In the equivalent Moore machine, state B can have only one action that either does, or does not, print the previous input character. State B in the Mealy machine must therefore be split into two states in the equivalent Moore machine (one state prints the previous input, the other does not, to deal correctly with NL and SP, respectively).

Construct a Moore machine that performs the same function as the Mealy machine shown above.