Finite State Machines

Tutorial Solutions

Dr S. S. Chandra

Problem 1

Define the main components of an FSM and their use during a computation. Use these definitions to describe how a basic automatic door with pressure plates sensing users can be represented using FSMs. Be sure to include all the components necessary, critical elements of an FSM and assumptions of your system in your answer.

The main components of an FSM are:

- States. These literally define the state of the machine.
- Transitions. These define how we move between states, essentially the 'rules' of computation.
- Accept and reject states. Each of these are a subset of the state set, and being 'in' one of these states determines whether a particular input is accepted or rejected.
- Input alphabet. This is the set of symbols which the machine accepts as inputs.

To create a simple automatic door, we can consider a system whereby we have a one-way door with a pressure plate on either side (P1, P2), and define an input alphabet based on whether a particular combination of pressure plates is pressed - ie the product set (P1, !P1)x(P2,!P2) where P1 denotes P1 being pressed and !P1 denotes it not being pressed. Let's ignore that doors don't open and close instantly and say we can have two states (Door open, door closed). The key thing to note here is that we don't want the door closing on people's heads!

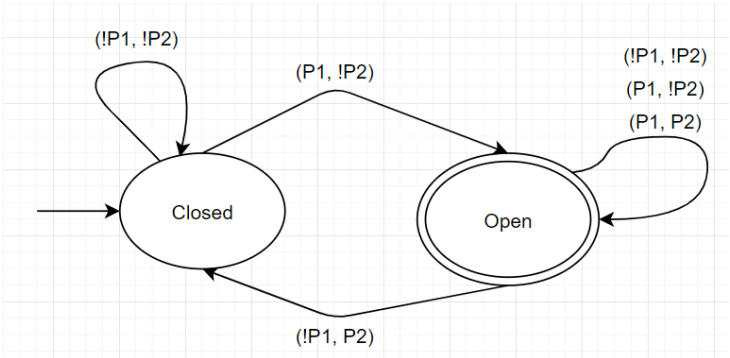
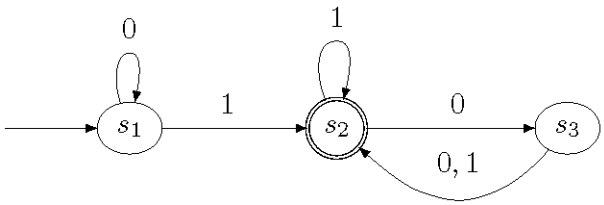


Figure 1: One-way door that opens and closes based on the inputs from two pressure plates.



Problem 2

For the given FSM above

a) List the set of all possible states.
b) Write the transition table.
c) Show that it accepts the strings w $= 10100$ and $u = 0001$.
d) What type of strings does this machine accept?

a) {s1, s2, s3}
b)

	s_1	s_2	s_3
0	s_1	s_3	s_2
1	s_2	s_2	s_2

c) This is easily shown just by inputting the strings and moving through the machine, demonstrating that we end up in an accept state each time.
d) By trying different strings, you should be able to see that this machine accepts strings which have at least one 1 symbol, and ends with either a single 1 symbol or two 0 symbols .

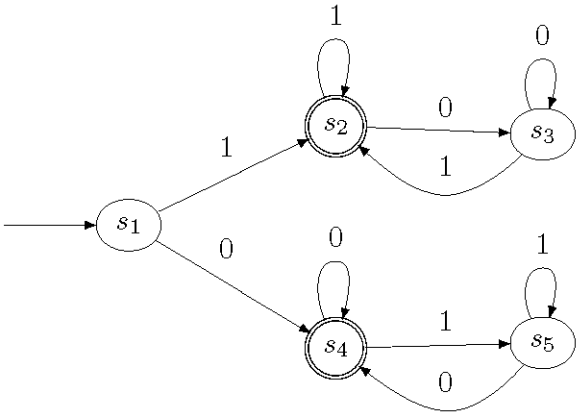
Problem 3

a) List the set of all possible states.
b) Write the transition table.
c) Show that it accepts the strings w = 101 and u = 00.
d) What type of strings does this machine accept?

a) {s1, s2, s3, s4, s5}
b)

	s_1	s_2	s_3	s_4	s_5
0	s_4	s_3	s_3	s_4	s_4
1	s_2	s_2	s_2	s_5	s_5

c) Simply input string and see that we end up in an accept state.
d) Strings which (begin with 1 and end with 1) or (begin with 0 and end with 0).



Problem 4

Design and define the following FSMs using either a transition table or state diagram

a) An FSM M_1 that only accepts set of strings containing even zeros
b) An FSM M_2 that only accepts set of strings containing odd ones
c) An FSM M_3 whose language is the union of the languages of the two machines. In other words, if the language accepted by M_1 is A and language accepted by M_2 is B then construct M_3 such that $A \cup B$, where the alphabet for $M_3$is also the same as alphabet for $M_1$and M_2.

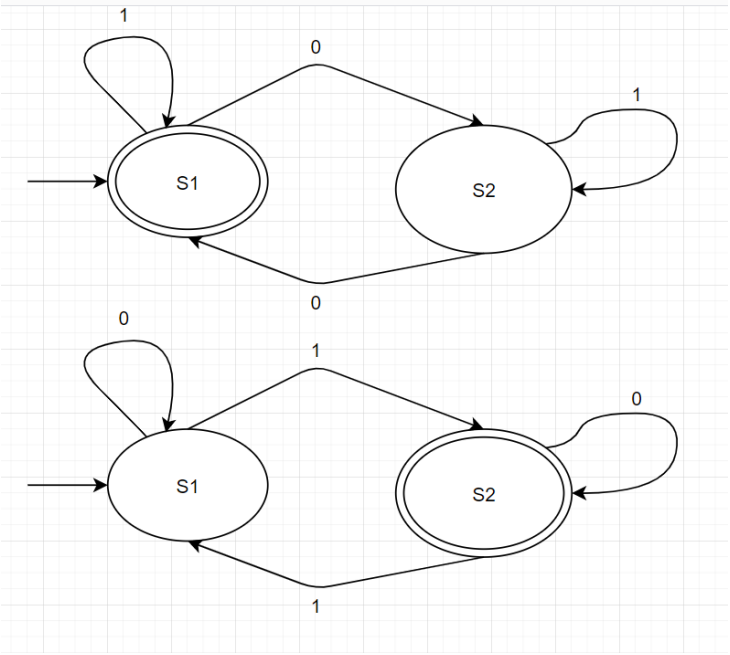


Figure 2: Top: M1. Bottom: M2.

c) A simple way to do this is to consider the product set of the state sets for each of the machines. Let's rewrite M2's states as $\{t_1, t_2\}$ for clarity here. Now we seek to effectively run the two machines in parallel on the component states of each state tuple. This is made most clear by looking at the transition table:

	(s_1, t_1)	(s_2, t_1)	(s_1, t_2)	(s_2, t_2)
0	(s_2, t_1)	(s_1, t_1)	(s_2, t_2)	(s_1, t_2)
1	(s_1, t_2)	(s_2, t_2)	(s_1, t_1)	(s_2, t_1)

You can see that the transition rules for the s components is determined by the transition function for M1, and the rules for the t components from the transition rules for M2. The accept states here would be the states where either the s or t component is an accept state in their respective machines.

Problem 5

Describe the main advantages and disadvantages of an FSM in terms of computation within roughly 250 words. Be sure to include at least one example and discussions on the consequence of having only a fixed number of finite states.

Advantages:

- Simple but quite powerful model for computation (anything we can do with a real memory limited computer, we can do with a FSM).
- A good basis for implementing simple programs (can easily translate a FSM to code), often used for simple embedded programs.
- Very intuitive and easy to design.

Disadvantages:

- Cannot solve many problems due to the finite number of states, as finite memory and unbounded inputs don't mix.
- The lack of explicit memory (eg the tape in a Turing machine) can make designing FSMs occasionally quite unwieldy.

The first disadvantage is the most important here. Consider the language $\{0^n 1^n | n \in \mathbf{N}\}$. This is an extremely simple language, but recognizing it requires that a machine can in some way remember how many 0's it has seen in order to match that with 1's. Try designing a FSM to recognise this language - you will find that no matter how many states you use, there were always be a sufficiently large input which your machine fails to recognise!

Problem 6

Design a simple non-playable character (NPC) or an animal's artificial intelligence (AI) system based on FSMs (without any code) for a game in a genre from the list in the appendix. For example, an AI system for a (non-interactable) mouse living in a dungeon part of an RPG game. The desired FSM AI system is to (at least) be able to handle the NPC's/animal's interaction after seeing a player and/or approaching it, as well as when the player is not detected or around. Ensure to include the transition diagram, assumptions and other important elements necessary for your AI system.

Let's design a simple enemy which has five states - fleeing, healing itself (which increases HP), wandering, attacking, and being dead. We will implicitly assume that the enemy has some HP statistic. We want our NPC to flee when it's HP is low so it can go heal, to wander when it can't see the player, to attack the player if it sees them, and to die when its HP reaches zero.

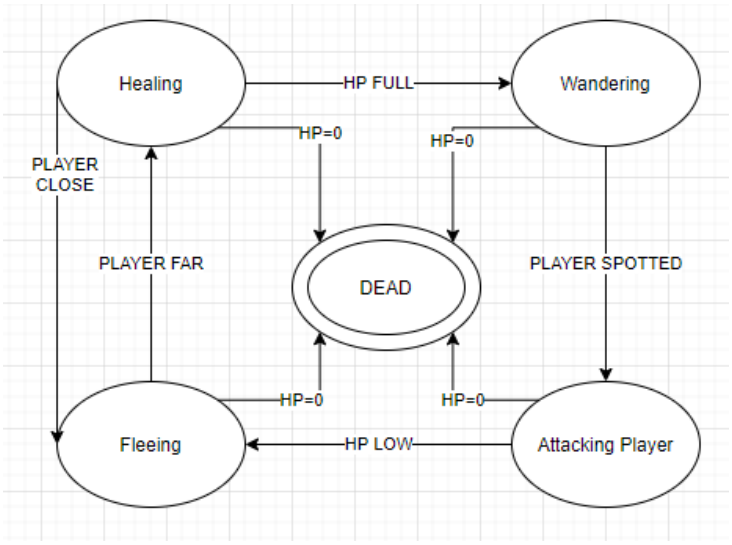


Figure 3: A simple enemy NPC