Finite State Machines.

Note Title 3/4/2015

FSMs : Simple model of a computational device

Used in :
- Digital logic design
- Specifying network / communication protocols
- Compiler design.
- Algorithms for text search.

Finite set of states : S.
the only thing an FSM "remembers" is its current state.

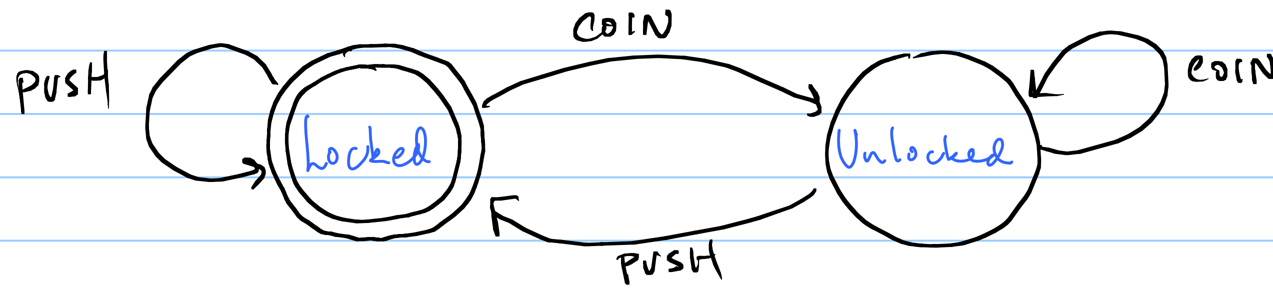
Finite input set : I
this is how the outside world (user) can interact with the device.

Transition function : describes how the device reacts to the input.

FSM diagram : directed graph.
Vertex set ⟷ states.
edges ⟷ transition function.
labels on edges ⟷ inputs.

Turnstyle example :

$S = \{ \text{Locked}, \text{Unlocked} \}$.
$I = \{ \text{Coin}, \text{Push} \}$.



Double circle denotes the initial state of the device.

There is an outgoing edge from each state labeled with each possible input action.

Transition function $\delta : S \times I \to S$

$$\left[\begin{array}{l} \delta(\text{Locked}, \text{Coin}) = \text{Unlocked}. \\ \delta(\text{Locked}, \text{Push}) = \text{locked} \\ \delta(\text{Unlocked}, \text{Coin}) = \text{Unlocked} \\ \delta(\text{Unlocked}, \text{Push}) = \text{locked}. \end{array}\right]$$

Formal Specification of an FSM:

$$(S, I, S_0, \delta)$$

transition function $\delta : S \times I \to S$.

finite set of states

finite set of input actions.

Start state $S_0 \in S$

Input: sequence of input actions from I.

(turnstyle animation).

3 different varieties of FSMs — how they interact with the outside world.

1) "output" is just the state (turnstyle example).

2) FSM has a finite set of output actions
Each transition results in one output action.

3) FSM can compute if an input string is in a set.
Finite set of "accepting" states.

Gumball Machine:

Sells gumballs for 20 ¢
Accepts nickels + dimes
Requires exact change
If buyer overshoots cost, returns last coin.

Input = { Nickel, Dime, Buy }

Output = { Return, Message, Gumball, None }

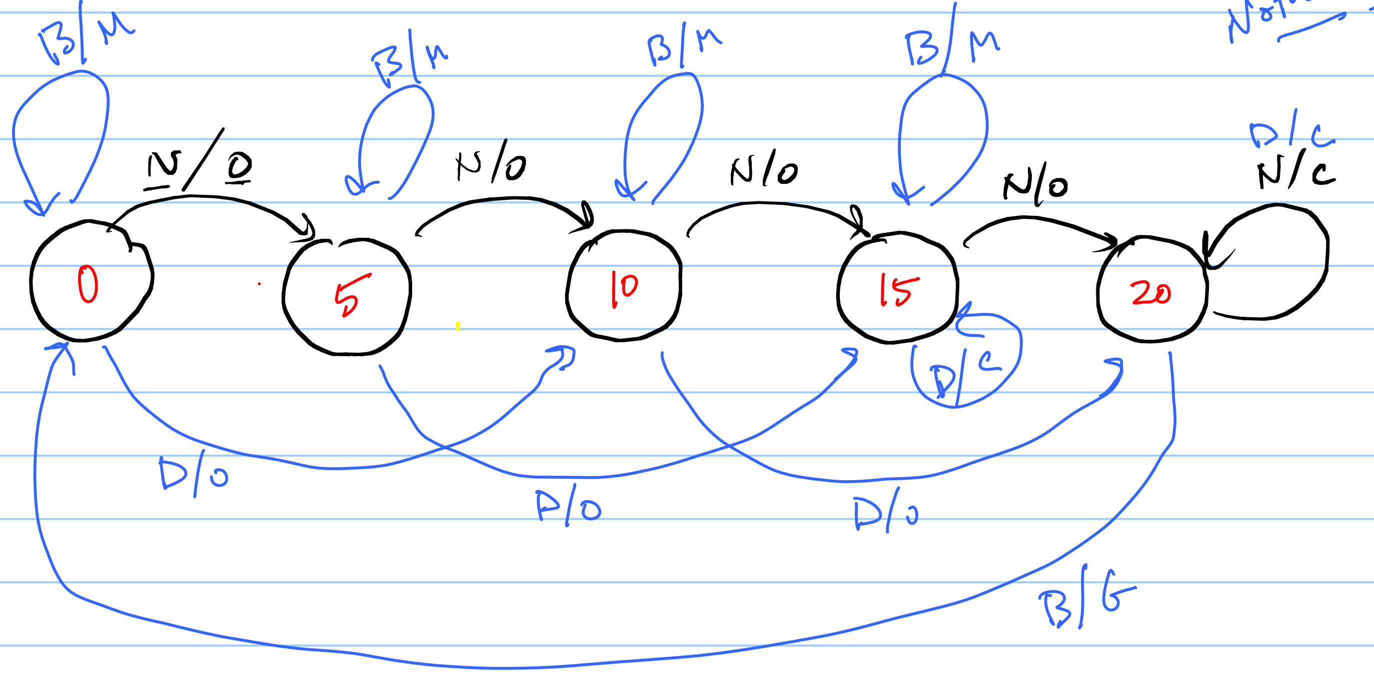
return last coin — "Need more money" — Release gumball.

State: encodes amount input so far

Input: N, D, B

Output: M, G, C, O

Do Nothing



(animation).

Formal description of an FSM with output:

$$(S,\ I,\ O,\ s_0,\ \delta)$$

finite set of states

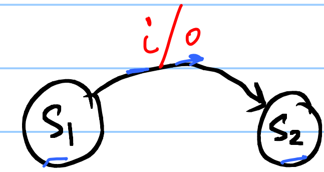
finite set of input actions.

finite set of output actions.

$s_0 \in S$, start state

$\delta : S \times I \rightarrow \underline{S \times O}$.

$\delta(s_1, i) = (s_2, o)$



Finite State Machines that recognize properties of strings.

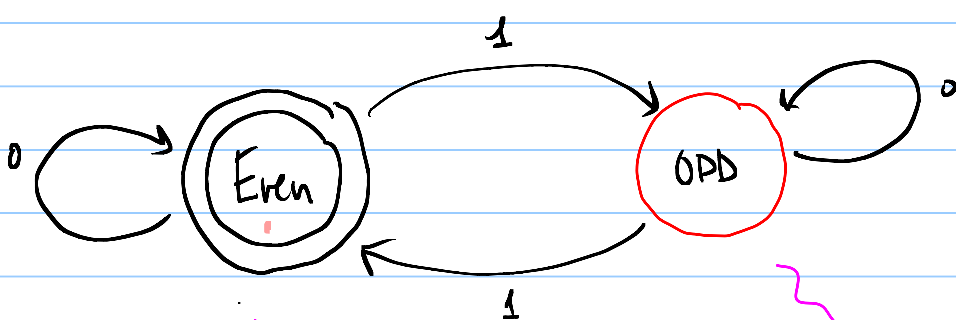
not as applicable to digital logic design.
more about guiding algorithm design

- recognize valid passwords.
- recognize correct program syntax
- recognize the occurrence of a string in a text file.

Input: finite set of symbols (alphabet).

Subset of the states are "accepting states".

Start @ start state
Process input string, one symbol at a time.
follow the outgoing edge from current state labeled with the next symbol.
If you end up in an accepting state at the end of the input string, accept the string.
Otherwise, reject.



red circle: accepting state.

0 1 1 0 reject.
1 0 1 1 accept.

Accepts iff: # 1's is odd.

Formal specs of an accepting FSM:

$$(S, I, S_0, \delta, A)$$

S: finite set of states

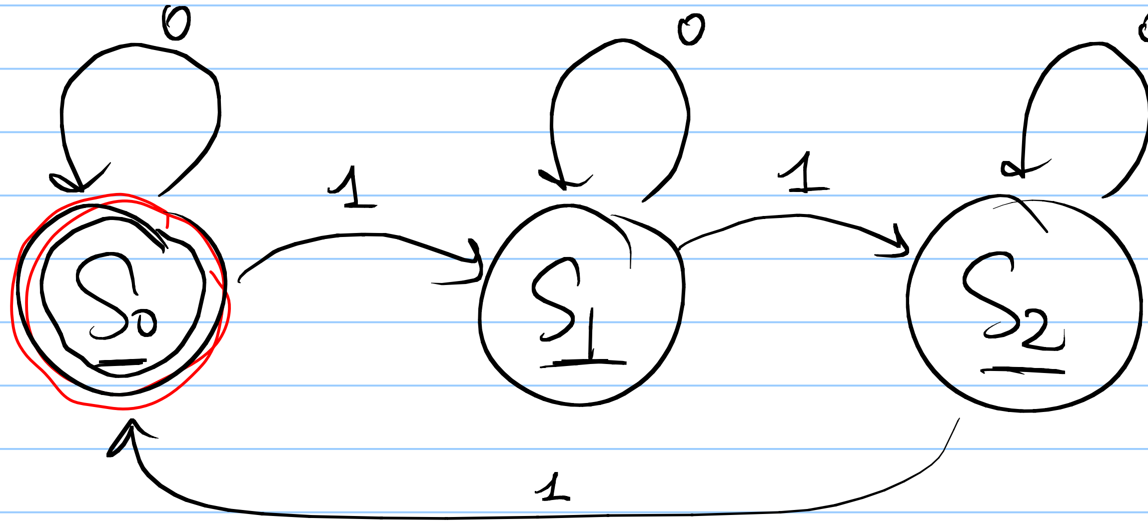
I: finite set of input symbols

$S_0 \in S$ start state.

$\delta : S \times I \to S$

$A \subseteq S$ set of accepting states.

Design an FSM that accepts a binary string ($I = \{0,1\}$) iff the number of 0's is a multiple of 3:

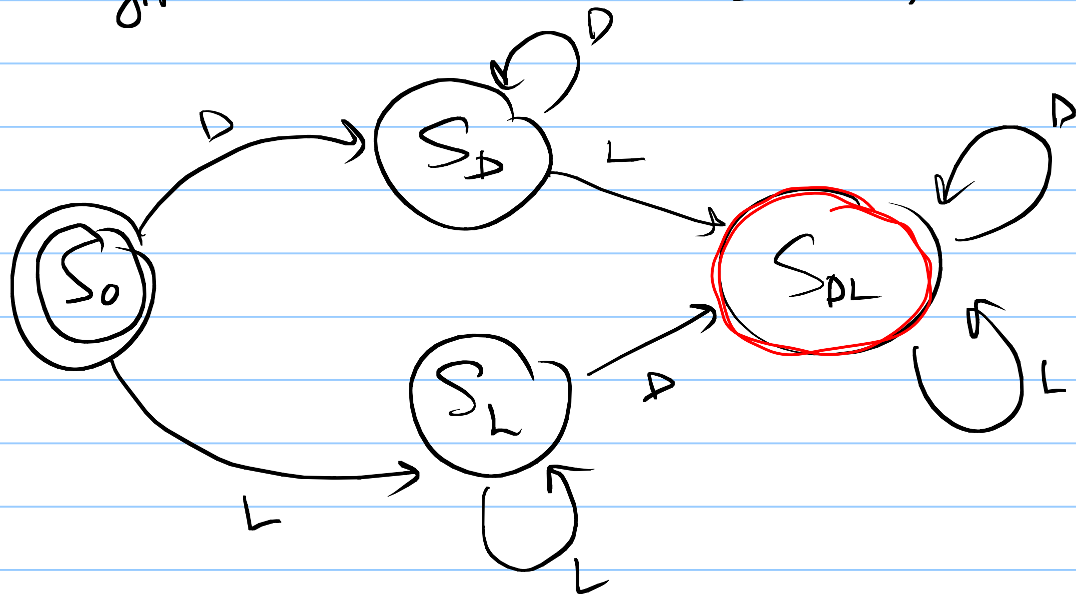


(# 1's mod 3)

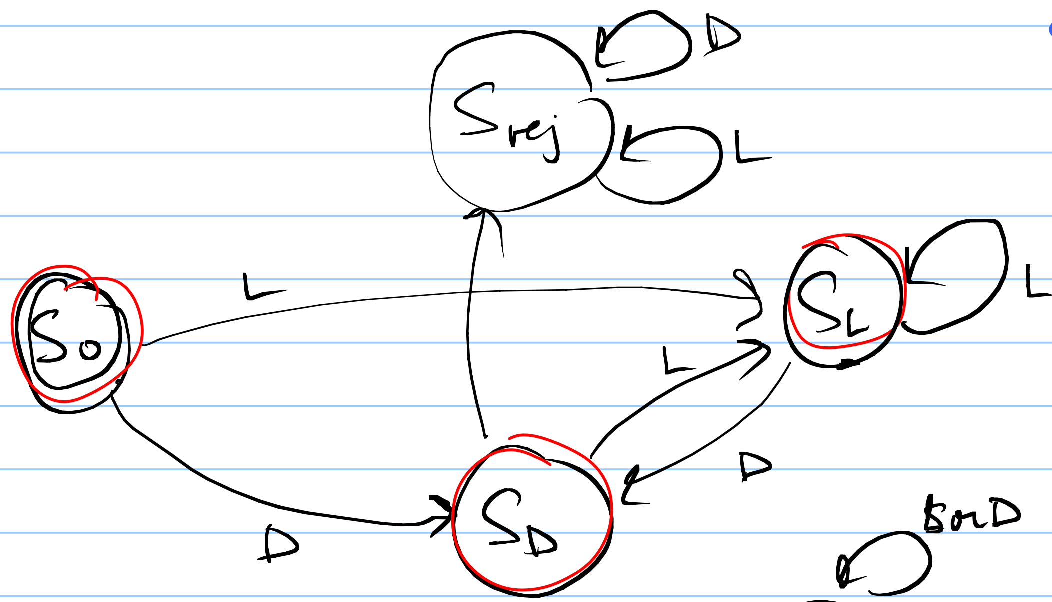
S_i : mult of 3 + i.

1 1 0 1 0 1

Design an FSM that takes in a sequence of digits or letters and accepts if there is at least one digit and one letter. $I = \{D, L\}$.



Design an FSM that takes in a sequence of digits or letters and accepts if the string does not have two consecutive digits. $I = \{D, L\}$.



S_L: last char was L

S_D: last char was D

S_{rej}: reject for sure.