

Finite State Machine (FSM)

by a safe lock design example



FSMs are used to model systems which have a limited number of **states**, **transitions** between these states and the **actions** taken as a result.

```

when I receive state0
  wait until sensor button pressed
  if a = sliderSensorValue
    wait 1 secs
    broadcast state1
  else
    broadcast state0
  stop script
    
```

```

when clicked
  set a to 7.0
  set b to 23.0
  set c to 61.0
  set d to 93.0
  broadcast state0
  forever
    set sliderSensorValue to round slider sensor value
    
```

```

when I receive state1
  wait until sensor button pressed
  if b = sliderSensorValue
    wait 1 secs
    broadcast state2
  else
    broadcast state0
  stop script
    
```

```

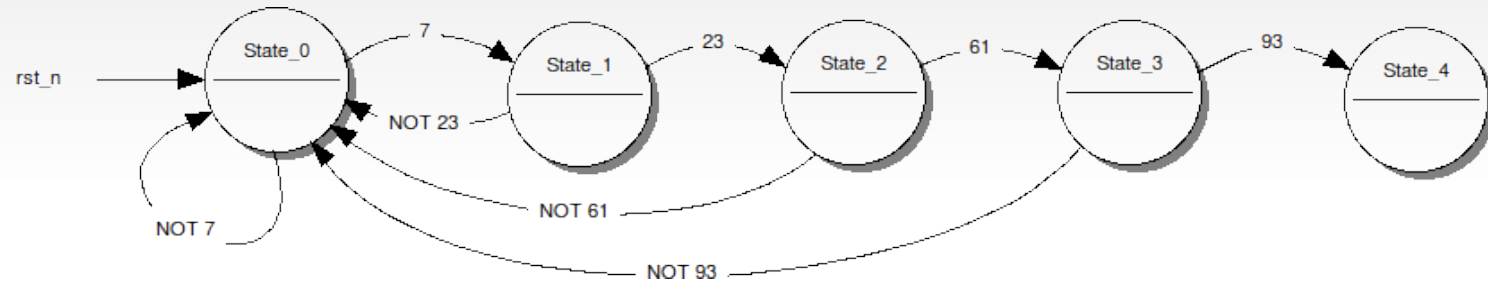
when I receive state4
  say Safe is unlocked !... for 2 secs
  stop all
    
```

```

when I receive state2
  wait until sensor button pressed
  if c = sliderSensorValue
    wait 1 secs
    broadcast state3
  else
    broadcast state0
  stop script
    
```

```

when I receive state3
  wait until sensor button pressed
  if d = sliderSensorValue
    wait 1 secs
    broadcast state4
  else
    broadcast state0
  stop script
    
```



- The numbers on the arrows represent what the system senses as an **input**, thus a **transition** from one **state** to another can take place. When there is no entry, the system keeps the **current state**.
- The example below represents an acceptor FSM, parsing the secret **combination** which, in our example, would un-lock a safe: 7, 23, 61, and 93. In case an unexpected number is entered, the system returns to the first state: State_0. The safe is unlocked, that is, the system is in the final state (State_4), only if the **correct numbers** are entered in the **correct order**.
- On the right hand side, a **possible FSM implementation** in scratch environment is given.