**State Machines**

* State Machines: model systems for which output depends on history of previous inputs
* Idea: define a set of possible system states and the ways to transition between them
* Transduce: stream of inputs >> stream of outputs (not state IDs)
* State Transition Diagram
  + should consider all possible inputs
  + each arc has ‘input/output’
  + indicate starting state
  + invalid inputs >> just remain in current state

|  |  |
| --- | --- |
| Torchlight | Language recognizer: abcabcabcabc … |

* Types of SM
  + Finite: draw state transition diagram
  + Infinite: draw time step table
  + Determine output function and next state function
* Requirements of State Machines: s = SM() #instantiate a state machine
  + s.start(): create an attribute for the instance, called state (has value of startState)
  + s.step(self, i): computes output, updates machines’ internal state, returns output value
  + getNextValues(self,state,inp) >> (next state, output)
    - is a pure function that does not change state of object
    - each state corresponds to one if statement
  + s.transduce(list): returns sequence of output values based on input list
  + define startState or start\_state
  + run(self, n = 10): for machines that do not take inputs
* Import: inside libdw it looks like this ‘class libdw.sm.SM’
  + import libdw.sm as sm  
    class CM(sm.SM):
  + from libdw import SM  
    class CM(SM): ????????????????

from libdw import sm # module sm has class SM

# from libdw import SM