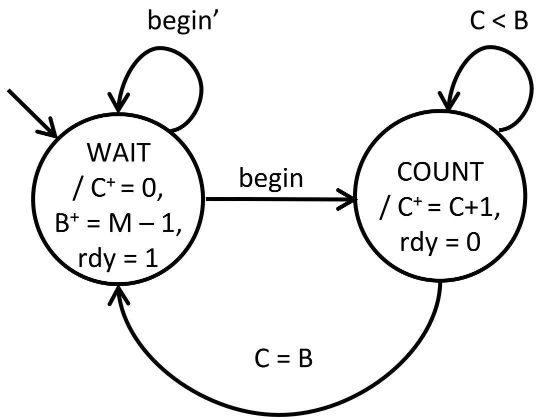
Extended Finite-State-Machines (EFSM)

EFSM design and implementation

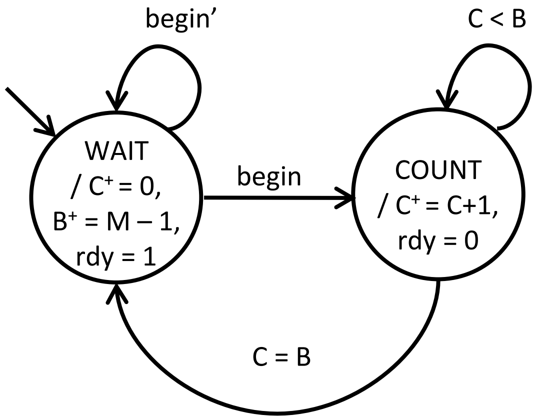
Extended Finite-State Machines

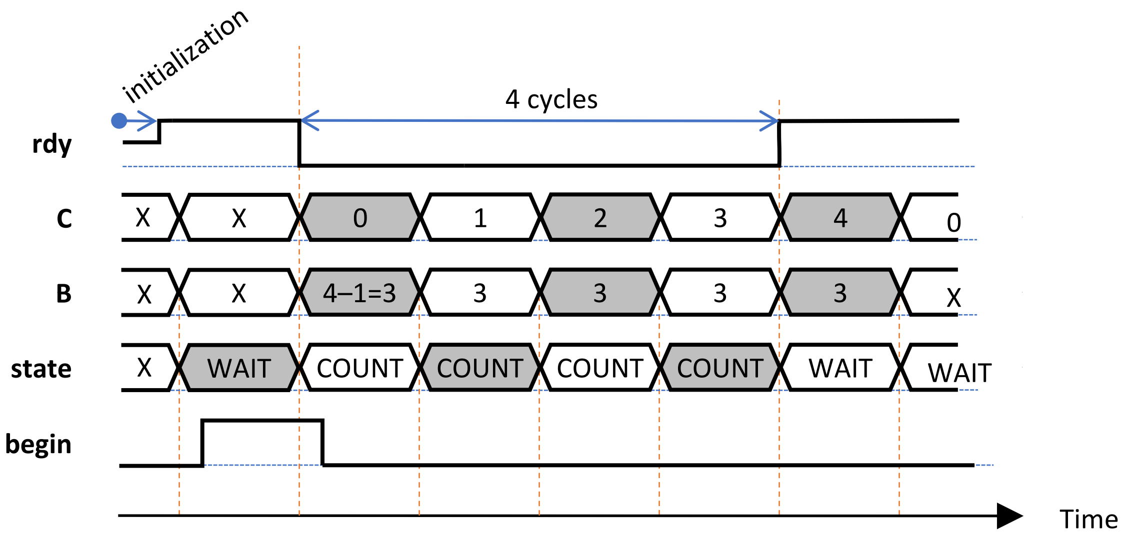
- Finite-state machines (FSM)
  - Inputs: symbols, binary encoded (i.e. $\{0, 1\}^m$)
  - Outputs: symbols, $\{0, 1\}^n$
- FSM + non-binary data + additional *state variables*
  - Full state: main state variable (*state*) + other state variables (*variables*)
    - Variables are updated at the same instant than the state.
  - Inputs: any type values for transition logic expressions
  - Outputs: any type values for computations
  - Example: counter up to *M*: 0, 1, 2, … *M*-1



Timing diagrams

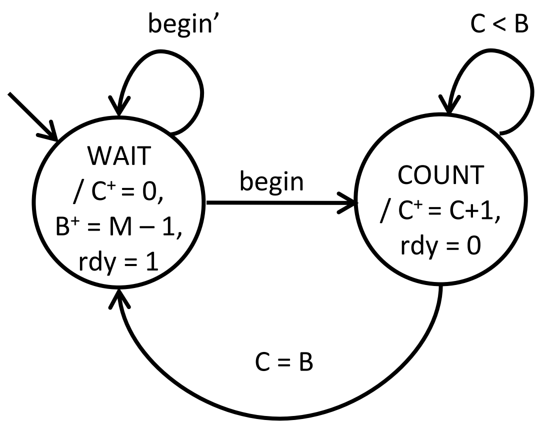
- A *timing diagram* shows how extended state and outputs vary over time
- Example:
  - Assume input M=4





Timing diagrams, tabular form

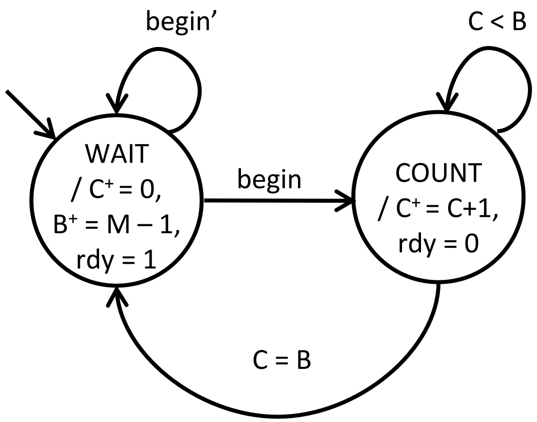
- Table filling method
  - Compute immediate outputs from extended state (and inputs)
    - Put the corresponding values into the same column, i.e., at the current state
  - Use current extended state and inputs to determine next values of state and variables
    - Put values into the next column at the right



Cycle	i+0	i+1	i+2	i+3	i+4	i+5	i+6
M	5	5	5	4	4	4	4
begin	false	false	false	true	true	false	false
state	COUNT	COUNT	WAIT	WAIT	COUNT	COUNT	COUNT
B	4	4	4	4	3	3	3
C	3	4	5	0	0	1	2
rdy	0	0	1	1	0	0	0

Implementation of EFSM

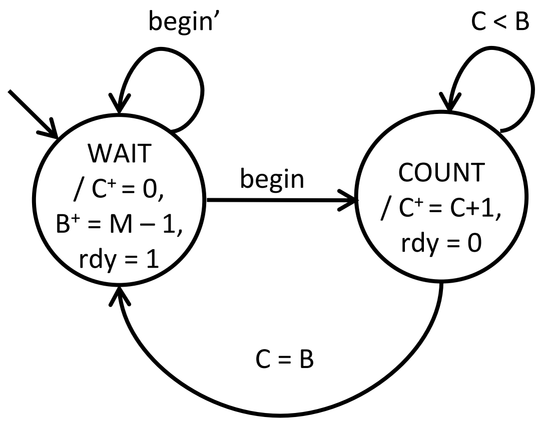
- State-based programming
  - Programs that simulate the behavior of a state machine
    - Initialize state (setup)
    - Loop
      - Write outputs
      - Read inputs
      - Compute next state
      - Set next state as current
  - Example in Lua




```lua
-- AUXILIARY FUNCTIONS

-- SETUP
init()
forward()

-- LOOP
cycle = 0
while state.curr~="STOP" do
  io.write(string.format("Cycle = %04i:\n", cycle))
  write_outputs()
  read_inputs()
  step()
  forward()
  cycle = cycle + 1
end -- while
io.write( "Program exited!\n" )
```

Implementation of EFSM, setup

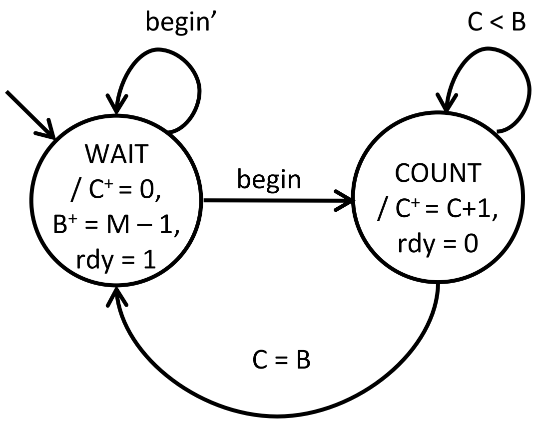
- **`init()`**
  - Sets the initial values of the extended state
- **`forward()`**
  - Sets the next state as the current, and the values of immediate outputs




```
-- AUXILIARY FUNCTIONS

init = function()
  state = {}; state.next = "WAIT"
  B = {}; B.next = nil
  C = {}; C.next = nil
end -- init()

forward = function()
  state.curr = state.next
  B.curr = B.next; C.curr = C.next
  if state.curr=="COUNT" then rdy = false
  else rdy = true
  end -- if
end -- forward()

-- SETUP
init()
forward()
```

Implementation of EFSM, next state

- **`step()`**
  - Computes next state




```lua
-- AUXILIARY FUNCTIONS

step = function()
  if begin==nil then
    state.next = "STOP"
    state.curr = "STOP"
  end -- if (simulation only)
  if state.curr=="WAIT" then
    C.next = 0; B.next = M - 1
    if begin then state.next = "COUNT" end
  elseif state.curr=="COUNT" then
    C.next = C.curr + 1
    if C.curr==B.curr then
      state.next = "WAIT"
    end -- if
  else -- STOP state or error
    state.next = "STOP"
  end -- if..elseif
end -- step()
```

Implementation of EFSM, input/output

```
-- AUXILIARY FUNCTIONS
write_outputs = function()
  io.write("< Counter: rdy = ")
  if rdy then io.write("1\n") else io.write("0\n") end
end -- write_outputs()
read_inputs = function()
  io.write( "> Counter: begin(0/1), M(integer) [whitespace=keep] = ")
  local line = io.read()
  local c = string.find(line, ",")
  if c~=nil then
    local new_begin = tonumber(string.sub(line, 1, (c-1)))
    if new_begin~=nil then
      if new_begin==0 then
        begin = false
      elseif new_begin==1 then
        begin = true
      else -- user wants to stop simulation
        begin = nil
      end -- if
    end -- if
    local new_M = tonumber(string.sub(line, (c + 1)))
    if new_M~=nil then M = new_M end
  else
    begin = nil
  end -- if
end -- read_inputs()
```

Example: Characterization of a mobile robot

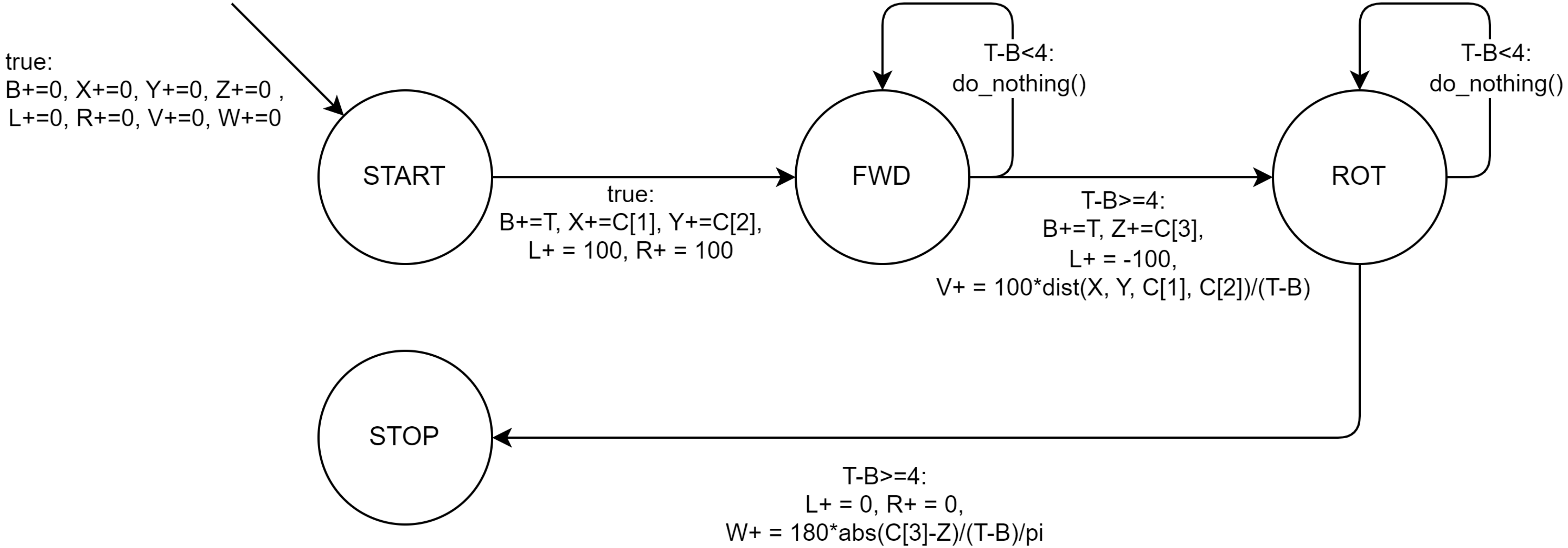
- Characterization refers to parameter identification
- For a simple mobile robot, like the UABotet, that is limited to either moving forward or turning in place, but not both at the same time
  - Linear speed in terms of control input, $v(u_{linear})$
  - Rotation speed, $w(u_{angular})$
  - Notice that there are other factors (battery charge, floor and wheel conditions, …) that affect v and w
- In an on/off control mode, only two values are required v_{max} and w_{max}
- Characterization of UABotet implies identifying the values of v_{max} and w_{max}

Example: Characterization of UABotet, 1

- Procedure
  - Make it move for a while and compute v_{max} = distance increment / elapsed time
  - Make it rotate for a while and compute w_{max} = angle variation / elapsed time
- Controller's inputs and outputs
  - Inputs
    - C : Coordinates/pose table
      - C[1] : X position [m]
      - C[2] : Y position [m]
      - C[3] : Orientation w.r.t. Z [rad]
    - T : Time [s]
  - Outputs
    - L, R : Control values of left and right motors [−100, 100]
    - V, W: Values of linear [cm/s] and rotational [deg/s] speeds

Example: Characterization of UABotet, 2

- EFSM diagram



Notice that it is a Mealy machine and that all output signals come from variables

Example: Characterization of UABotet, 3

```lua
init = function()
  robot = nil -- Object handle
  if sim then robot = sim.getObject("..") end
  C = {}       -- Coordinates (X, Y, angle w.r.t. Z)
  T = 0        -- Time
  state = {}; state.next = "START"
  B = {}; B.next = 0 -- Begin time
  X = {}; X.next = 0 -- X position
  Y = {}; Y.next = 0 -- Y position
  Z = {}; Z.next = 0 -- orientation
  L = {}; L.next = 0 -- DC value for left motor
  R = {}; R.next = 0 -- DC value for right motor
  V = {}; V.next = 0 -- Linear speed
  W = {}; W.next = 0 -- Angular speed
end -- init()
forward = function()
  state.curr = state.next
  B.curr = B.next
  X.curr = X.next
  Y.curr = Y.next
  Z.curr = Z.next
  L.curr = L.next
  R.curr = R.next
  V.curr = V.next
  W.curr = W.next
end -- forward()
```

```lua
step = function()
  if not sim and C[1]==nil then state.curr = "STOP" end
  if state.curr=="START" then
    B.next = T; X.next = C[1]; Y.next = C[2]
    L.next = 100; R.next = 100
    state.next = "FWD"
  elseif state.curr=="FWD" then
    local delay = T-B.curr
    if delay>4 then
      local dX = C[1]-X.curr
      local dY = C[2]-Y.curr
      V.next = 100*math.sqrt(dX*dX+dY*dY)/delay
      B.next = T; Z.next = C[3]
      L.next = -100; R.next = 100
      state.next = "ROT"
    end -- if
  elseif state.curr=="ROT" then
    local delay = T-B.curr
    if delay>4 then
      local dA = math.abs(C[3]-Z.curr)
      W.next = 180*dA/delay/math.pi
      L.next = 0; R.next = 0
      state.next = "STOP"
    end -- if
  else -- STOP or error
      state.next = "STOP"
  end -- if..ifelse
end -- step()
```

Example: Characterization of UABotet, 4

```
read_inputs = function()
  if sim then
    T = sim.getSimulationTime()
    local position = sim.getObjectPosition(
      robot, sim.handle_world)
    local eulerAngles = sim.getObjectOrientation(
      robot, sim.handle_world)
    C[1] = position[1]; C[2] = position[2]
    C[3] = eulerAngles[3]
  else -- console input
    T = T + 0.05
    io.write(string.format("> T = %.4fs or ...", T))
    local newT = tonumber(io.read())
    if newT then T = newT end
    io.write(string.format("> X = "))
    C[1] = tonumber(io.read())
    if C[1] then
      io.write(string.format("> Y = "))
      C[2] = tonumber(io.read())
      if C[2] then
        io.write(string.format("> orientation = "))
        C[3] = tonumber(io.read())
      end -- if
    end -- if
    if not C[2] or not C[3] then C[1] = nil end
  end -- if
end -- read_inputs()
```

```
write_outputs = function()
  print(string.format(
      "< L= %i, R= %i, V= %.2fcm/s, W= %.2fdeg/s\n",
      L.curr, R.curr, V.curr, W.curr))
  if sim then
    sim.setInt32Signal("DC_left", L.curr)
    sim.setInt32Signal("DC_right", R.curr)
  end -- if
end -- write_outputs()
```

- The program changes I/O upon being linked to a simulator
- If not associated with a sim, then simulates EFSM itself
  - Local simulation engine:

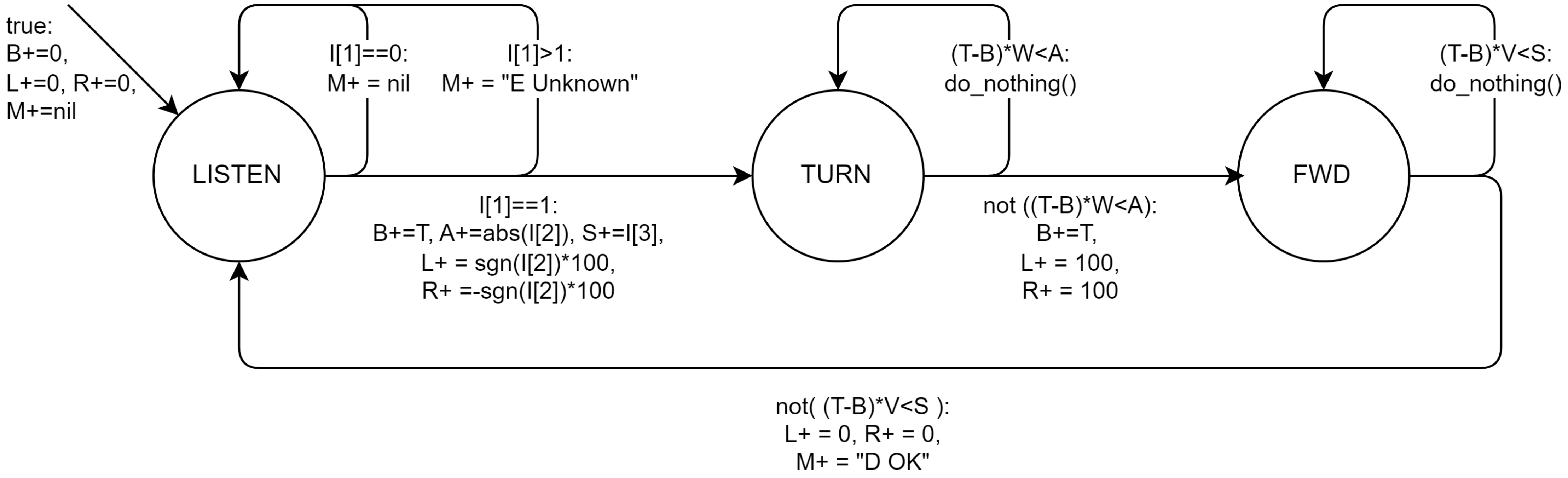
```
if not sim then
  init()
  forward()
  while state.curr~="STOP" do
    write_outputs()
    read_inputs()
    step()
    forward()
  end -- while
end -- if
```

Exercise 2: Open-loop control of mobile robot

- Specification
  - Move the robot to the specified polar coordinates (angle, distance) any time instruction command is 1 (“go”) and stop
- Controller’s inputs and outputs
  - Inputs
    - I : Instruction table
      - I[1] : Instruction command = 0 (none), 1 (go)
      - I[2] : Angle to turn [deg], from −90 to 90
      - I[3] : Distance to move [cm], from 0 to 255
    - T : Time [s]
    - V, W: Values of linear and rotational speeds (constant)
  - Outputs
    - L, R : Control values of left and right motors
    - M : Reply message, nil or string

Exercise 2: Open-loop control of mobile robot

- Basic EFSM



sgn(*a*) returns -1, 0, or +1 for *a*<0, *a*==0, and *a*>0, respectively

Exercise 2: Open-loop control of mobile robot

- Programming in Lua
  - Use the characterization program as example
  - init(), forward() and step()
- Is there any problem when instruction is GO 0 10?, and GO 15 0?