Module-4- Modelling programs – FSM-CDFG-DFG

Introduction

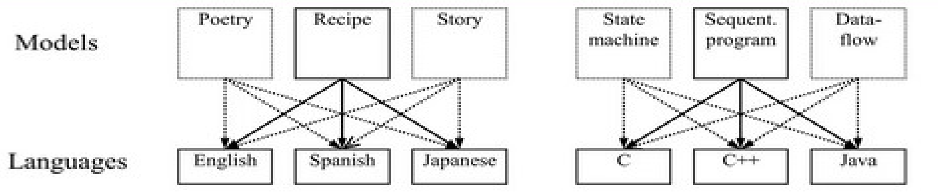
- Describing embedded system's processing behavior
  - Sequential program computation model
    - A set of instructions execute sequentially
    - One statement at a time
  - Assembly language
  - High level languages like C

- Complexity increasing with increasing IC capacity
  - Past: washing machines, small games, etc.
    - Hundreds of lines of code
  - Today: TV set-top boxes, Cell phone, etc.
    - Hundreds of thousands of lines of code
  - Describing the behavior becomes more complex
- Desired behavior often not fully understood in beginning
  - Many implementation bugs due to description mistakes /omissions

Types of models

- How can we (precisely) capture behavior?
  - We may think of languages (C, C++), but *computation model* is the key
- Common computation models:
  1. Sequential program model
     - Statements, rules for composing statements, semantics for executing one statement at a time
  2. Communicating process model
     - Multiple sequential programs running concurrently
  3. State machine model
     - For control dominated systems
     - A control dominated system is one whose behavior consists mostly of monitoring control inputs and reacting by setting control outputs
  4. Dataflow model
     - For data dominated systems
     - A data dominated system's behavior consists mostly of transforming streams of input data into streams output data
  5. Object-oriented model
     - For breaking complex software into simpler, well-defined pieces

Models vs. languages



Recipes vs. English **Sequential programs vs. C**

- Variety of languages can capture one model
  - E.g., sequential program model → C,C++, Java
- One language can capture variety of models
  - E.g., C++ → sequential program model, object-oriented model, state machine model
- Certain languages better at capturing certain computation models

- Variety of languages can capture one model
  - E.g., sequential program model → C,C++, Java
- One language can capture variety of models
  - E.g., C++ → sequential program model, object-oriented model, state machine model
- Certain languages better at capturing certain computation models

Text versus Graphics

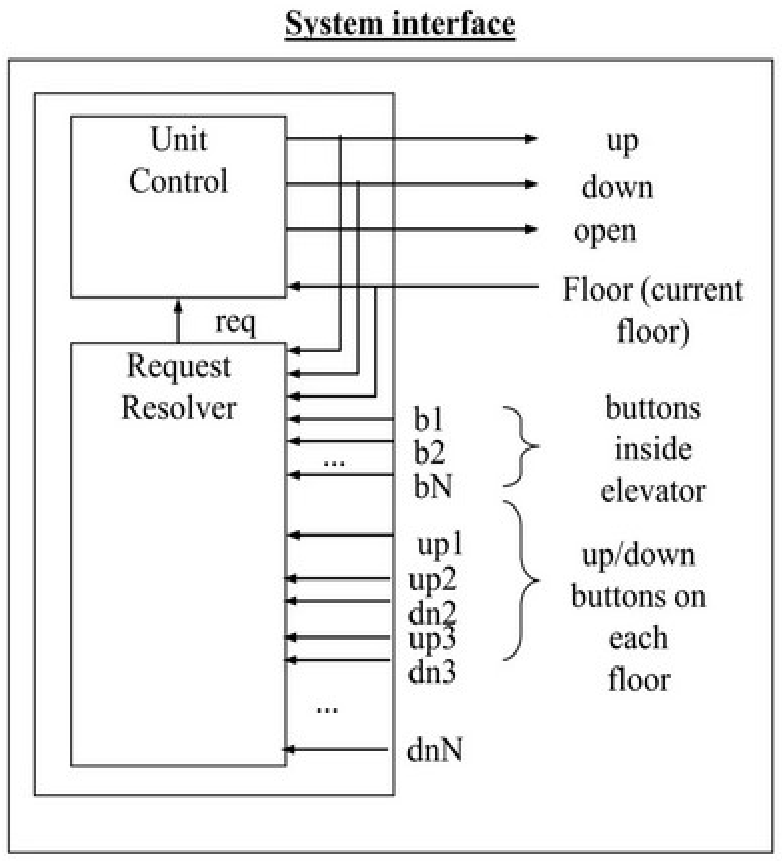
- Models versus languages not to be confused with text versus graphics
  - Text and graphics are just two types of languages
    - Text: letters, numbers
    - Graphics: circles, arrows (plus some letters, numbers)

```
  X = 1;
Y = X + 1;
```

$$\begin{tikzcd}
\boxed{X = 1} \arrow[d] \\
\boxed{Y = X + 1}
\end{tikzcd}$$

Introductory example: An elevator controller

- Simple elevator controller
- Two major blocks:
  - *Request Resolver* resolves various floor requests into single requested floor
  - *Unit Control* moves elevator to this requested floor
- Simple English description….

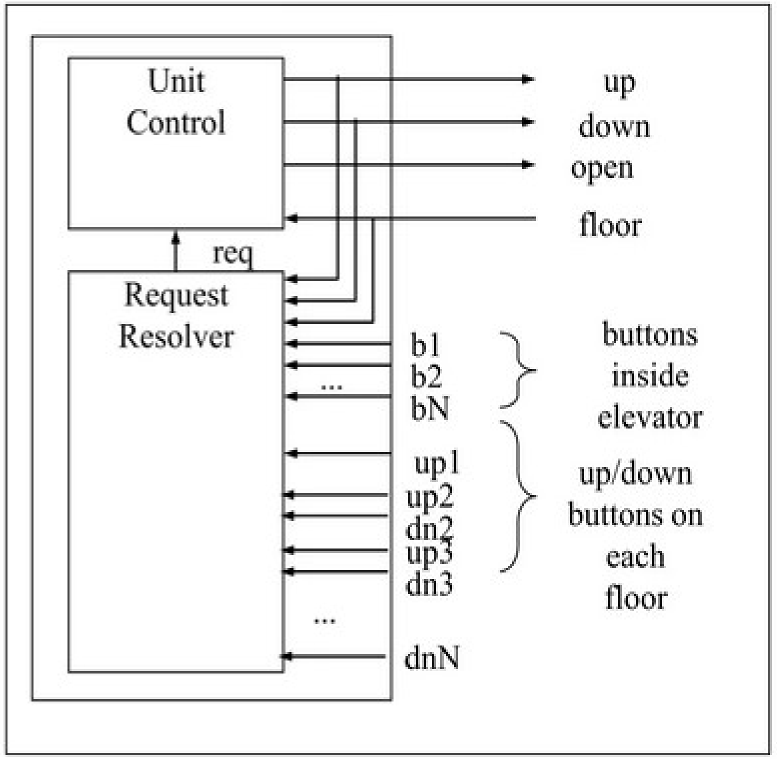


Introductory example: An elevator controller

Partial English description

"Move the elevator either up or down to reach the requested floor. Once at the requested floor, open the door for at least 10 seconds, and keep it open until the requested floor changes. Ensure the door is never open while moving. Don't change directions unless there are no higher requests when moving up or no lower requests when moving down…"

System interface



Elevator controller using a sequential program model

Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

```
void Unit_Control ( )
 {
  up = down = 0; open = 1; //Initial condition
  while (1)    {        // Infinite / super loop
    while (req != floor)
    open = 0; //close the door
    if (req > floor) { up = 1 ; }
    else {   down = 1;   }
    while (req = = floor); // Same floor
    up = down = 0;
    open = 1; // Keep the door open
    delay(10) ; // Wait for 10 seconds
  }              }
```

```
void Request_Resolver ( )
{
  while (1)
  ...
    req = ...
  ...
}
```

```
void main ( )
{
  Call concurrently:
    Unit_Control( ) and
    Request_Resolver( )
}
```

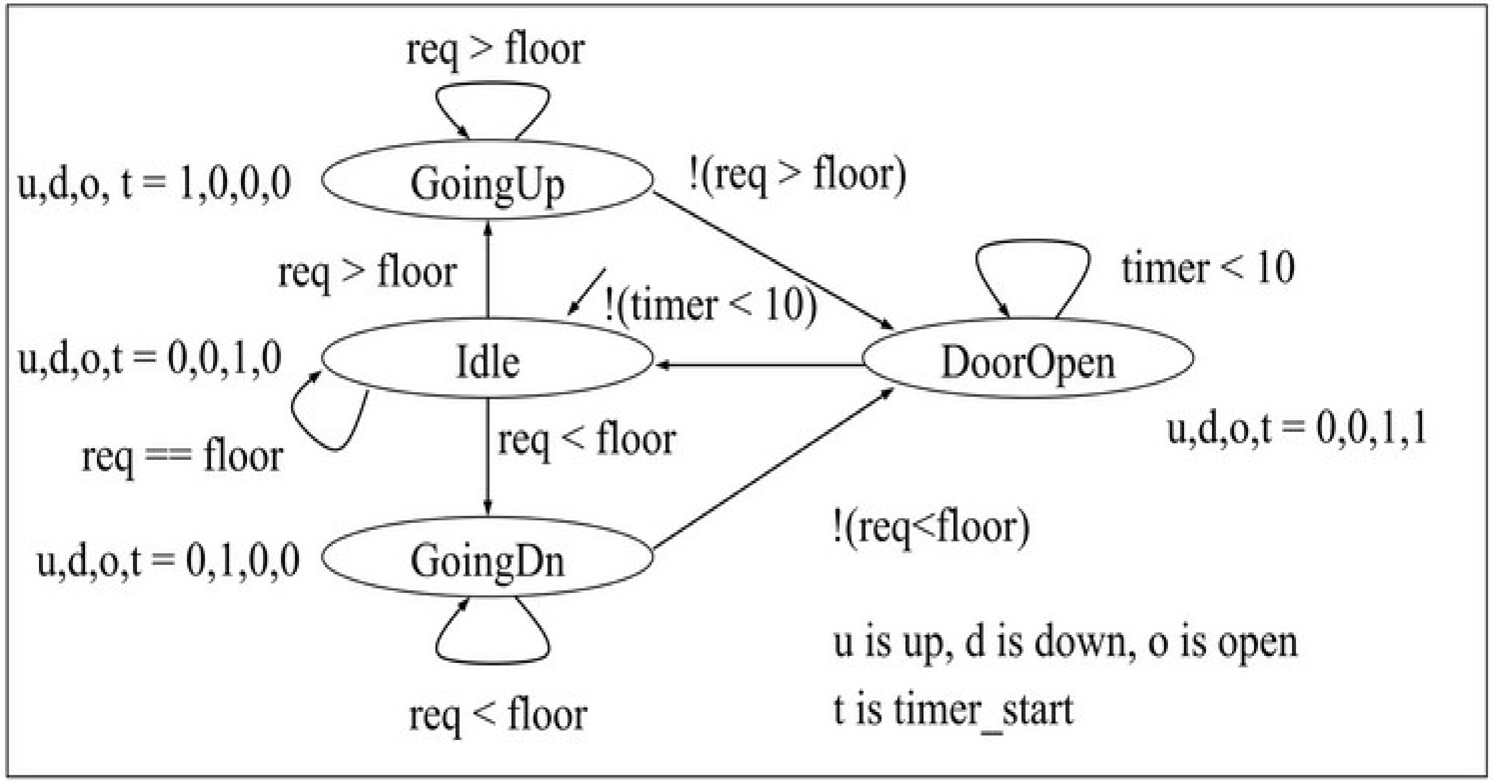
Finite-state machine (FSM) model

- In a finite state machine (FSM) model we describe the system behaviour as a set of possible states
- The system can only be in one of these states at a given time
- It describes the possible state transitions from one state to another depending on input values
- It also describes the actions that occur when in a state or when transitioning between states

- Trying to capture this behavior as sequential program is a bit awkward
- Instead, we might consider an FSM model, describing the system as:
  - Possible states
    - E.g., *Idle*, *GoingUp*, *GoingDn*, *DoorOpen*
  - Possible transitions from one state to another based on input
    - E.g., req > floor
  - Actions that occur in each state
    - E.g., In the *GoingUp* state, u,d,o,t = 1,0,0,0 (up = 1, down, open, and timer_start = 0)

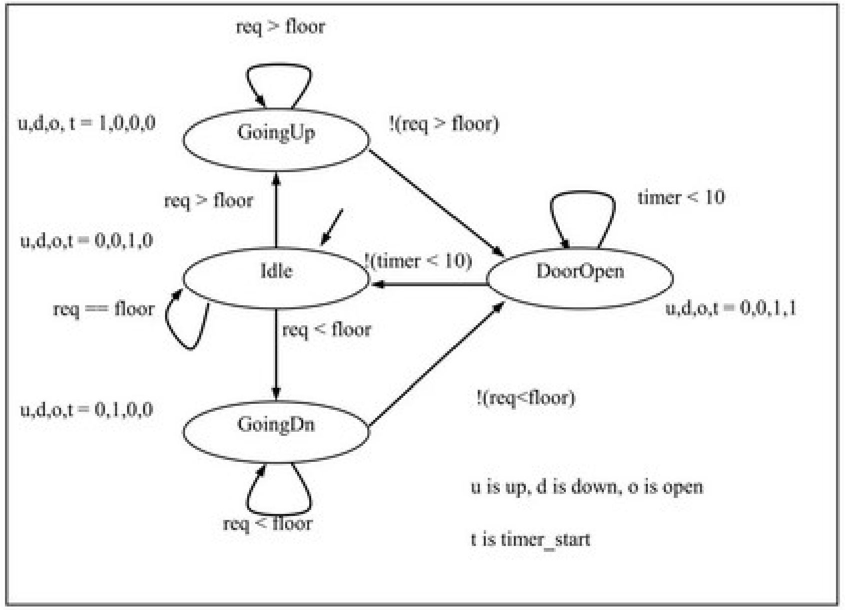
Finite-state machine (FSM) model

***Unit Control* process using a state machine**



Describing a system as a state machine

1. List all possible states
2. Declare all variables (none in this example)
3. For each state, list possible transitions, with conditions, to other states
4. For each state and/or transition, list associated actions
5. For each state, ensure exclusive and complete exiting transition conditions
   - No two exiting conditions can be true at same time
     - Otherwise nondeterministic state machine
   - One condition must be true at any given time
     - Reducing explicit transitions should be avoided when first learning



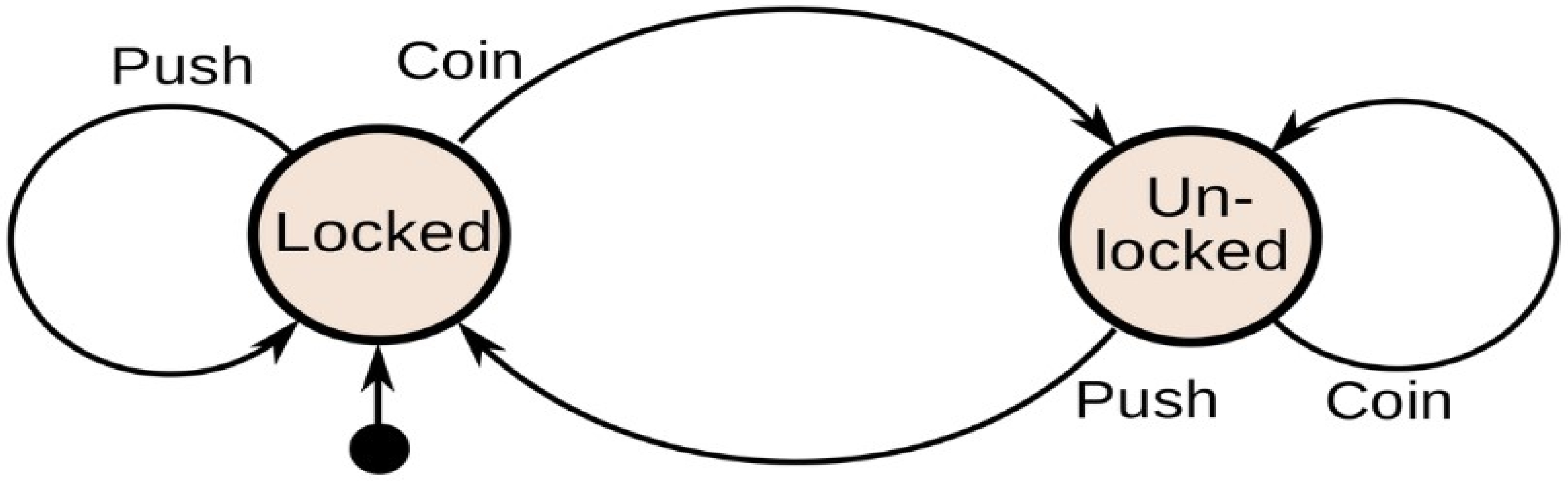
State machine vs. sequential program model

- Different thought process used with each model
- State machine:
  - Encourages designer to think of all possible states and transitions among states based on all possible input conditions
- Sequential program model:
  - Designed to transform data through series of instructions that may be iterated and conditionally executed
- Advantages of State machine description
  - More natural means of computing in those cases
  - *Not* due to graphical representation (state diagram)
    - Would still have same benefits if textual language used (i.e., state table)
    - Besides, sequential program model could use graphical representation (i.e., flowchart)

Role of appropriate model and language

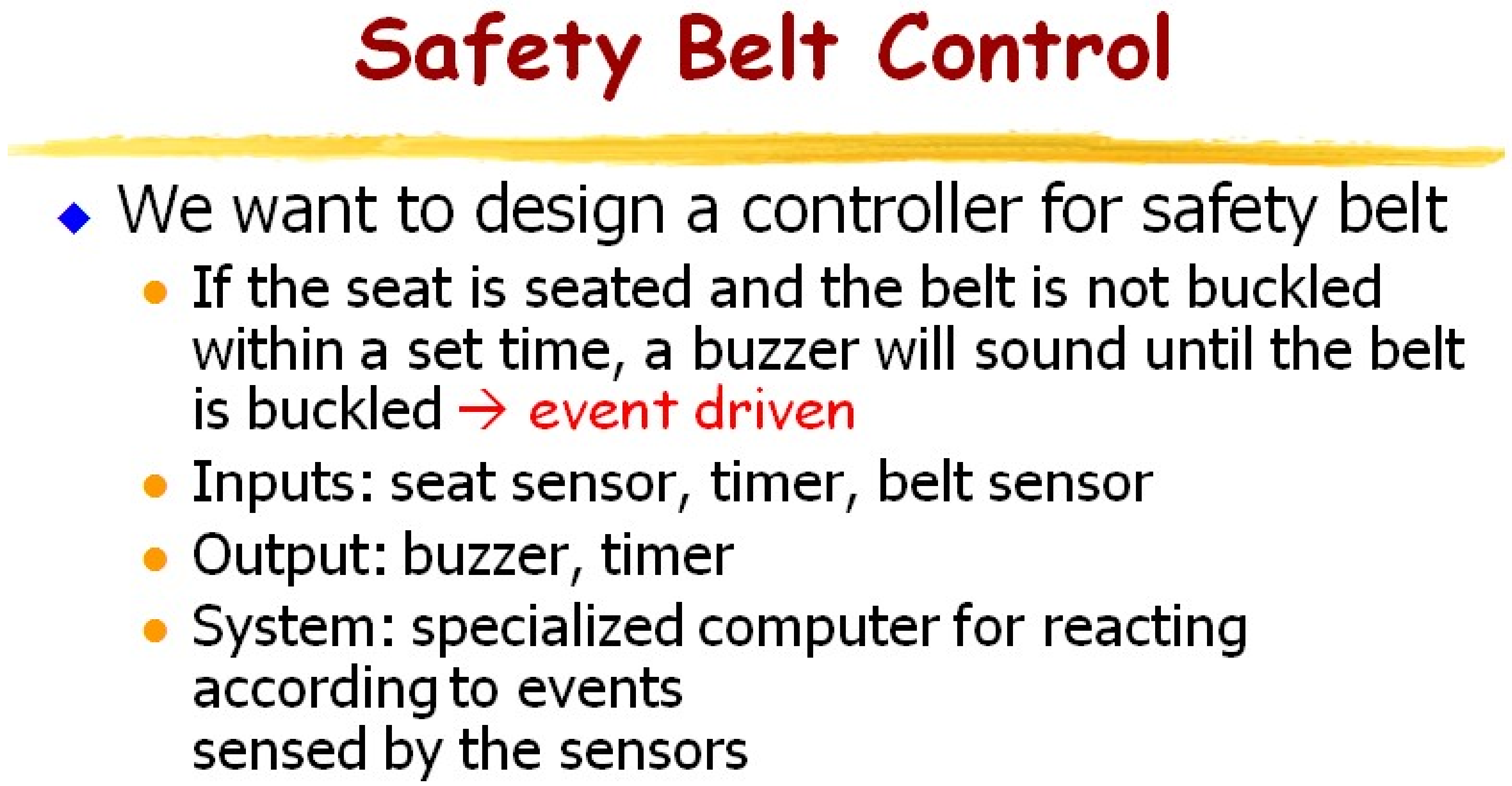
- Finding appropriate model to capture embedded system is an important step
  - Model shapes the way we think of the system
    - Originally thought of sequence of actions, wrote sequential program
      - First wait for requested floor to differ from target floor
      - Then, we close the door
      - Then, we move up or down to the desired floor
      - Then, we open the door
      - Then, we repeat this sequence
    - To create state machine, we thought in terms of states and transitions among states
      - When system must react to changing inputs, state machine might be best model
- Language should capture model easily
  - Ideally should have features that directly capture constructs of model
  - *FireMode* would be very complex in sequential program
    - Checks inserted throughout code
  - Other factors may force choice of different model
    - Structured techniques can be used instead
      - E.g., Template for state machine capture in sequential program language

Example: coin-operated turnstile



Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlocks the turnstile so that the customer can push through.
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When the customer has pushed through, locks the turnstile.

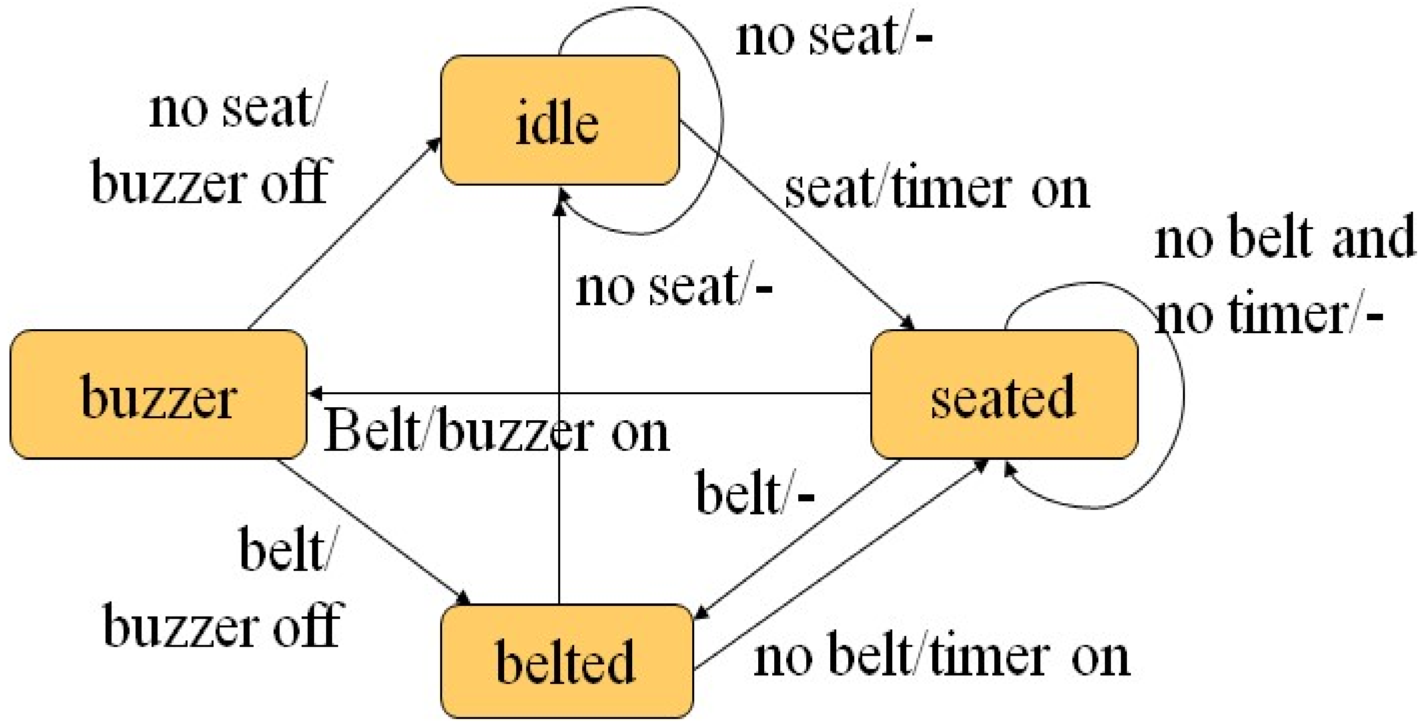
More examples



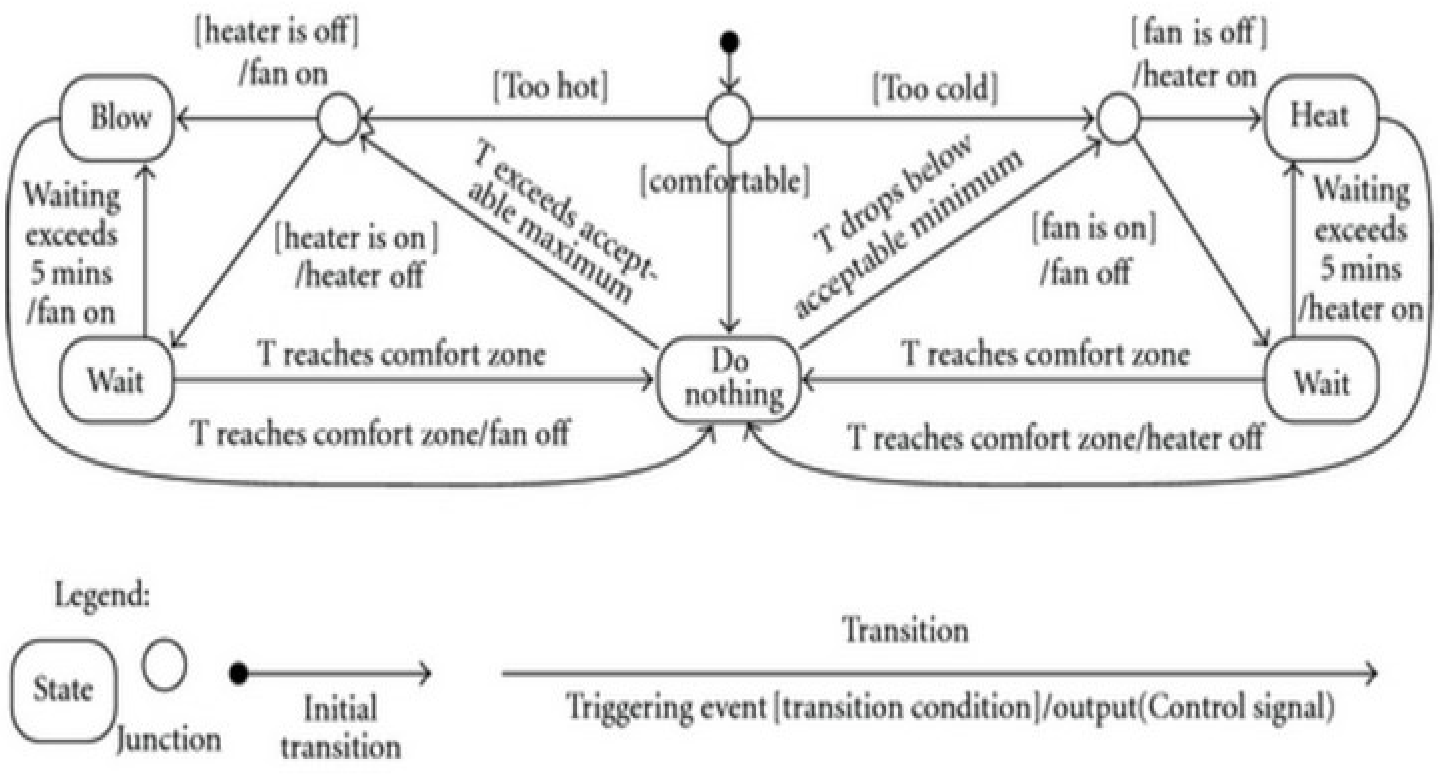
Safety Belt Control

- We want to design a controller for safety belt
  - If the seat is seated and the belt is not buckled within a set time, a buzzer will sound until the belt is buckled → event driven
  - Inputs: seat sensor, timer, belt sensor
  - Output: buzzer, timer
  - System: specialized computer for reacting according to events
    sensed by the sensors

FSM for Event-driven Systems



Example of a simple state machine: for embedded control in a heating/cooling system


[heater is off]
/fan on
[Too hot]
[Too cold]
[fan is off]
/heater on
Blow
Heat
[comfortable]
T exceeds acceptable maximum
T drops below acceptable minimum
Waiting exceeds 5 mins /fan on
[heater is on] /heater off
[fan is on] /fan off
Waiting exceeds 5 mins /heater on
Wait
T reaches comfort zone
Do nothing
T reaches comfort zone
Wait
T reaches comfort zone/fan off
T reaches comfort zone/heater off
Legend:
State
Junction
Initial transition
Transition
Triggering event [transition condition]/output(Control signal)

A Washing Machine Controller

A washing machine is an example of a finite state machine. It starts in a defined state (empty and cold). When it is started, it moves through a series of states, full with water, heated to the correct temperature, through water being admitted or pumped out, or heaters being turned on and off.

- Fill with hot water until tub is filled to the correct depth
- Switch on heater until temperature is correct
- Agitate for a predetermined time interval
- Pump out water for a predetermined time
- Fill with cold water till tub is full
- Agitate for a predetermined time interval
- Pump out water for a predetermined time
- Spin tub for predetermined time interval
- Stop

Washing Machine – Inputs and Outputs

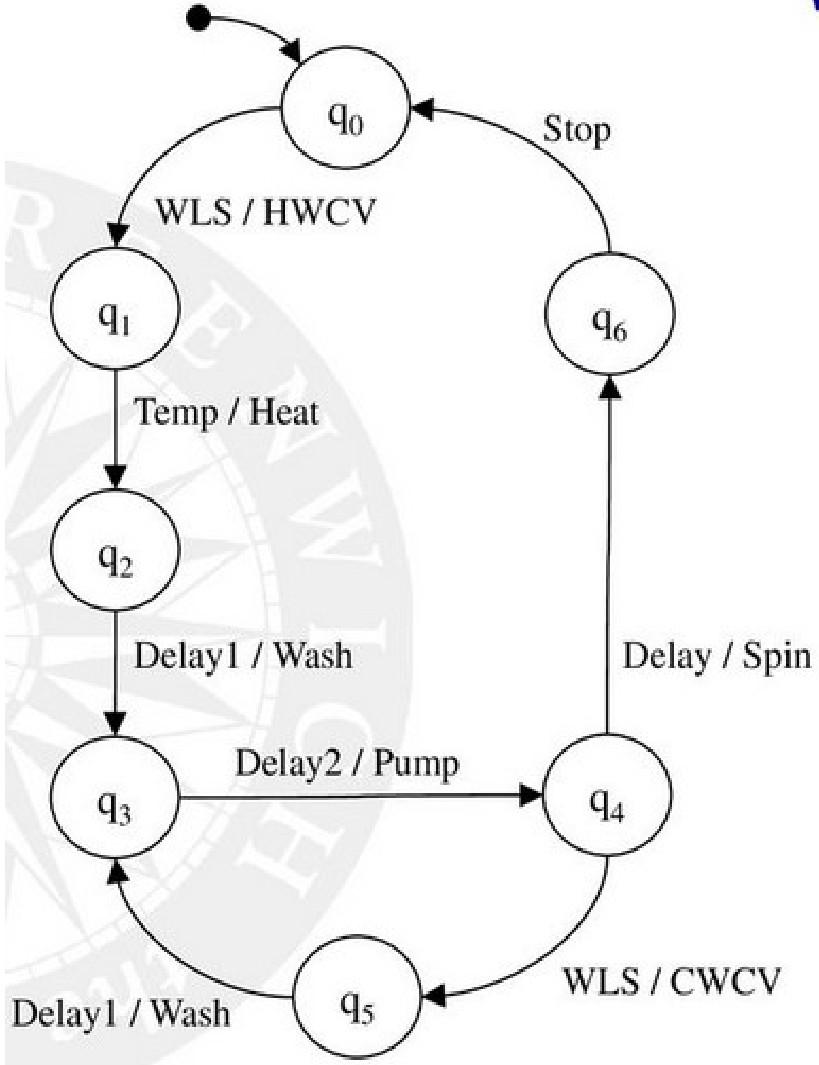
The inputs which drive the finite state machine through its states are either external inputs from the user (setting the washing machine program), or internal inputs from thermostats, level sensors or timers.

The example described here is greatly simplified it assumes that there is a single sequence of states. Real life washing machines have a greater complexity, pre-wash, filling to different heights; different temperatures; different rinse and spin cycles.

There are six devices to be controlled by output signals, and two input signals. The output signals will be assigned to the upper six bits of a suitably configured output port, the two inputs will be connected to the least significant bits of an input port of the same device. Since there are heaters and motors to be driven, some kind of level translation devices are required, but not discussed further.

Port Assignment	Device
Port A Bit 7 (Output)	Hot Water Control Valve (HWCV)
Port A Bit 6 (Output)	Cold Water Control Valve (CWCV)
Port A Bit 5 (Output)	Water Heater (Heat)
Port A Bit 4 (Output)	Washing Drum Motor (Wash/Rinse) (Wash)
Port A Bit 3 (Output)	Washing Drum Motor (Spin Speed) (Spin)
Port A Bit 2 (Output)	Pump Motor (Pump)
Port A Bit 1 (Input)	Water Level Sensor (WLS)
Port A Bit 0 (Input)	Thermostat (Temp)

Washing Machine FSM



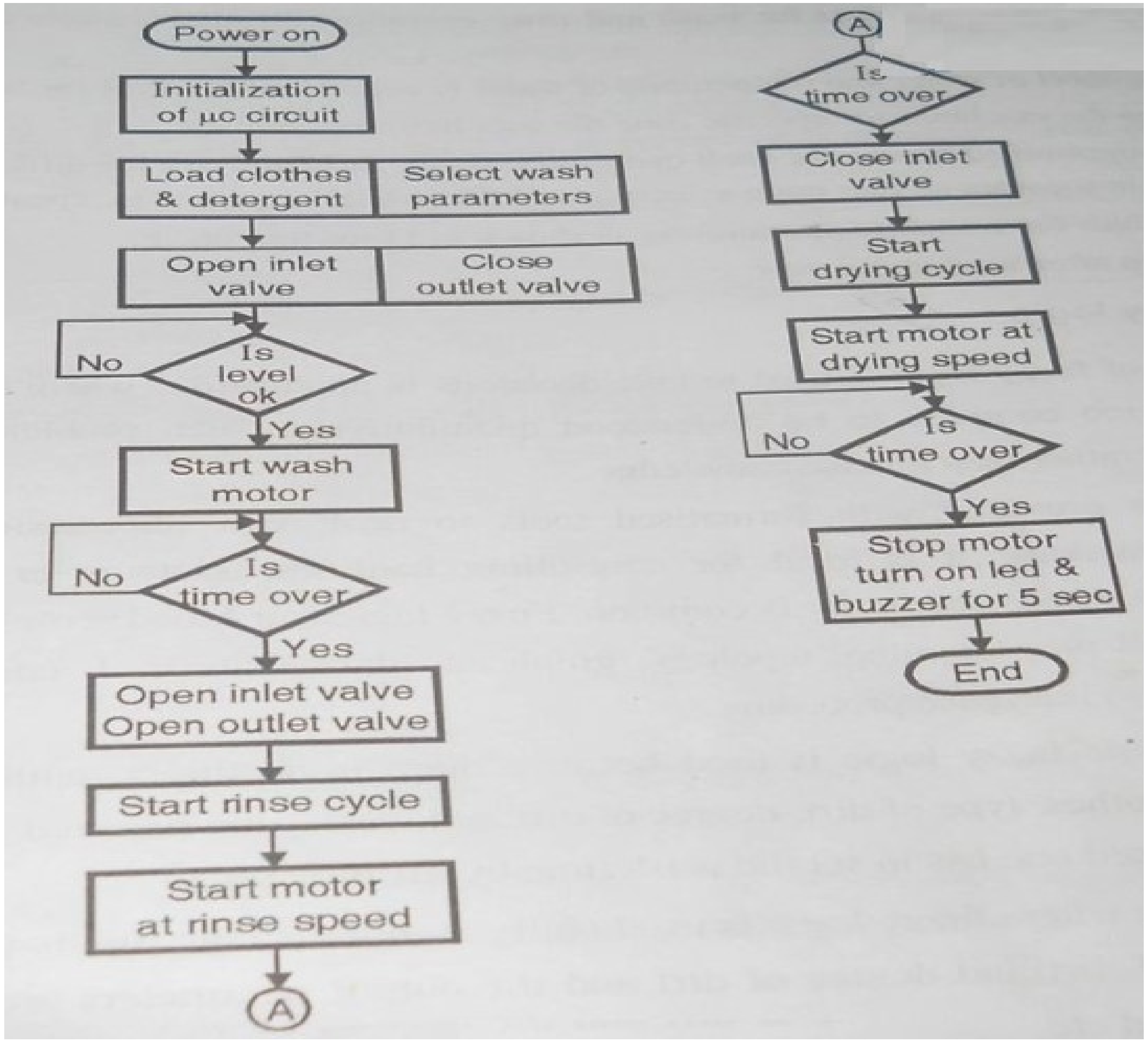
State	
q_0	Cold and Empty
q_1	Full with Hot Water
q_2	Water Heated to Correct Temperature
q_3	Drum Agitating
q_4	Pumping
q_5	Full with Cold Water
q_6	Spinning

Washing Machine State Table

The state table will be implemented as a table and a pointer. The information in the table will be of two kinds, firstly the bit commands to be sent to the heater, motors and pumps; and secondly the time delay and conditional inputs required before the next state can be entered

	Device Control						Input Conditions				
State	HWCV	CWCV	Heat	Wash	Pump	Spin	WLS	Temp	Delay 1	Delay 2	Action
Q1	1	0	0	0	0	0	1	0	0	0	Fill with Hot Water Until Full
Q2	0	0	1	0	0	0	0	1	0	0	Heat Until Required Temperature
Q3	0	0	0	1	0	0	0	0	1	0	Wash for time D1
Q4	0	0	0	0	1	0	0	0	0	1	Pump for time D2
Q5	0	1	0	0	0	0	1	0	0	0	Fill with Cold Water Until Full
Q3	0	0	0	1	0	0	0	0	1	0	Rinse for Time D1
Q4	0	0	0	0	1	0	0	0	0	1	Pump for time D2
Q6	0	0	0	0	0	1	0	0	1	0	Spin for Time D1
Q0	0	0	0	0	0	0	0	0	0	0	Stop

CDFG – Example: Washing machine

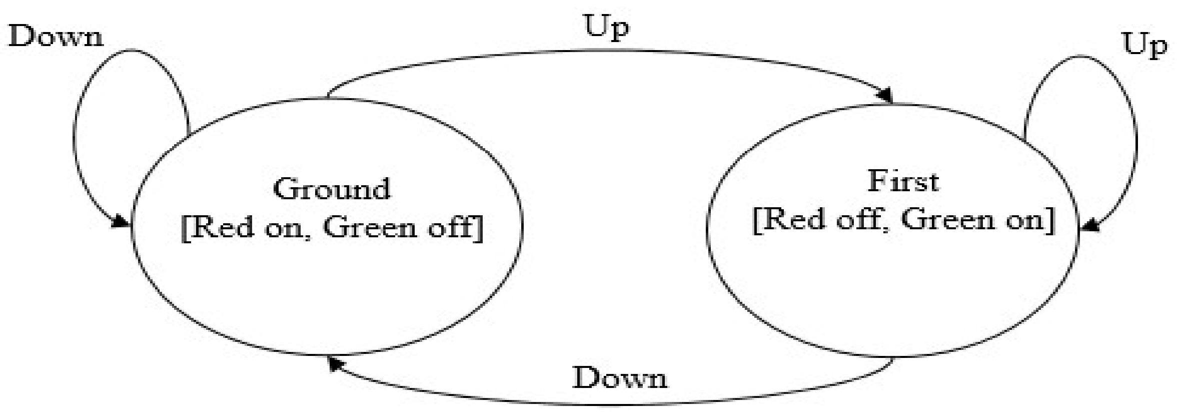


FSM – Example: Elevator

Step 1: Describe the machine in words.

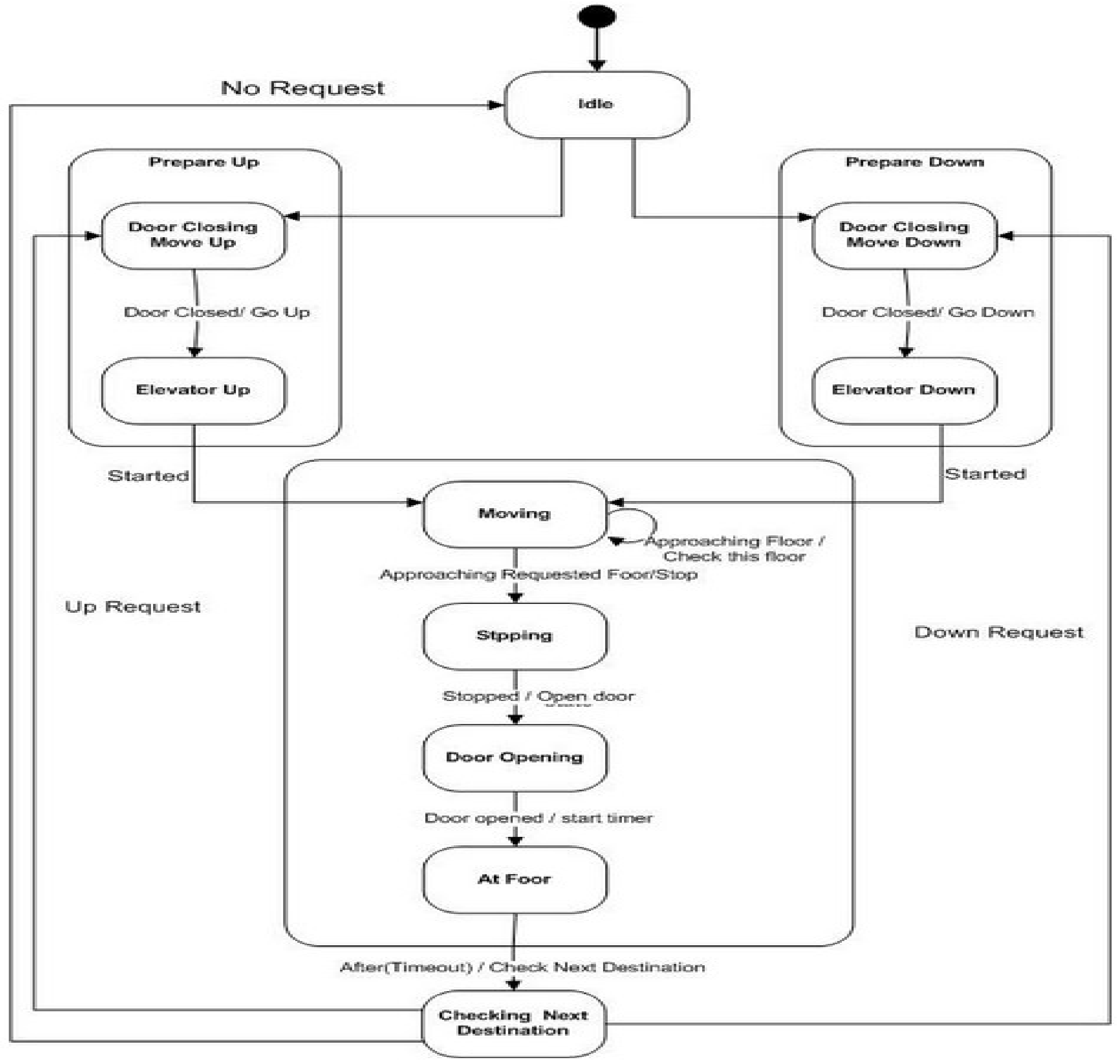
In this example, we'll be designing a controller for an elevator. The elevator can be at one of two floors: Ground or First. There is one button that controls the elevator, and it has two values: Up or Down. Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First. At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way.

Step 2: Draw the FSM diagram

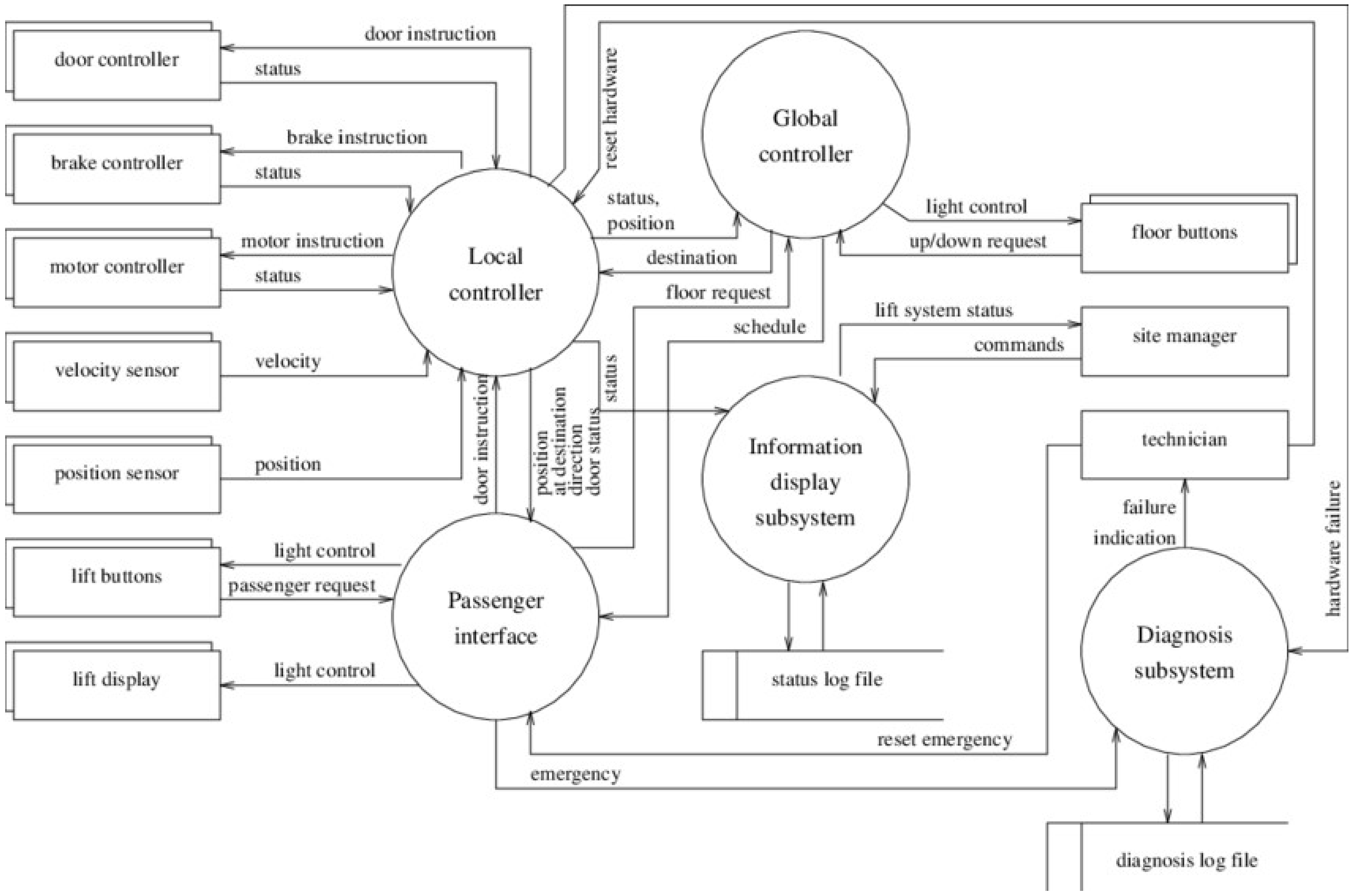


In this diagram, the bubbles represent the states, and the arrows represent state transitions. The arrow labels indicate the input value corresponding to the transition. For instance, when the elevator is in the Ground state, and the input is Up, the next state is First. The information in the brackets indicates the output values for the lights in each state.

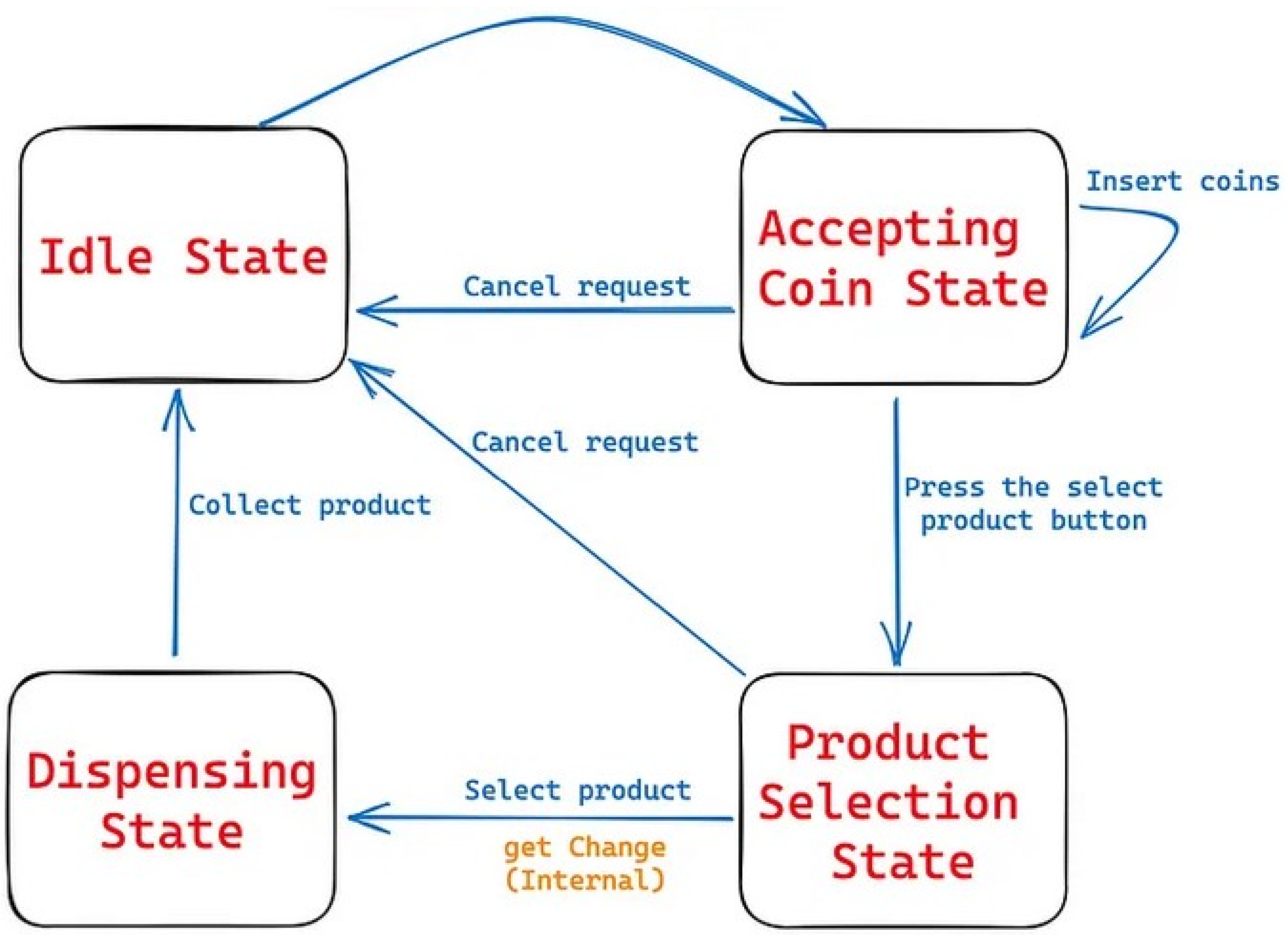
FSM – Example: Elevator in detail



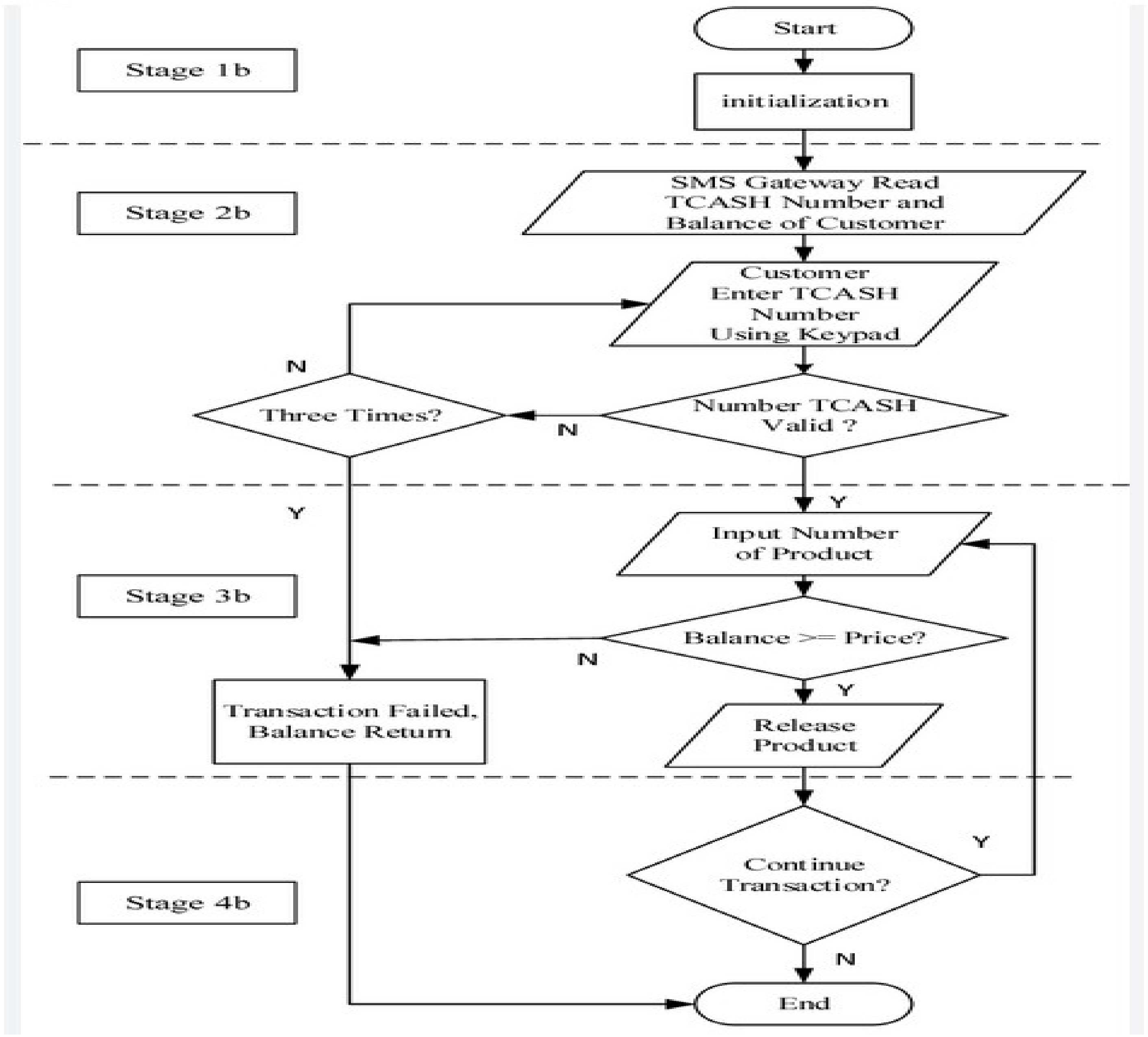
DFG – Example: Elevator in detail



FSM – Example: Vending machine



CDFG – Example: Vending machine


Stage 1b
Start
initialization
Stage 2b
SMS Gateway Read TCASH Number and Balance of Customer
Customer Enter TCASH Number Using Keypad
N
Three Times?
N
Number TCASH Valid ?
Y
Y
Input Number of Product
Stage 3b
Balance >= Price?
N
Transaction Failed, Balance Return
Y
Release Product
Y
Continue Transaction?
Stage 4b
N
End