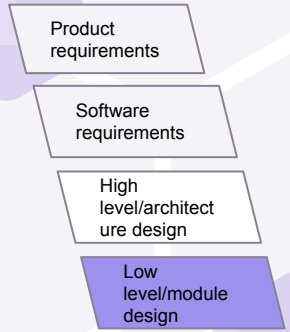
15 & 16: Finite state machines



Finite state machines

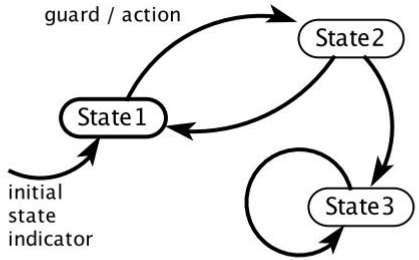
Low-level design for a module

Shows the change in state of a module

- Contrast with **flowchart**, which just shows flow of computation

At basic level, composed of:

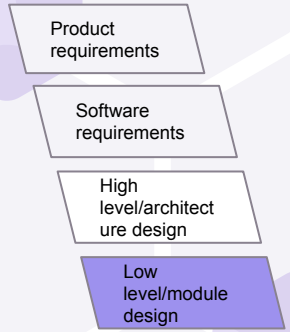
- States (one state is initial state)
- Guards (predicates on inputs)
- Actions (setting outputs)



Lee/Seshia chapter 3

Figure 3.3: Visual notation for a finite state machine.

Variants



Multiple ways to define statecharts/FSMs

- Mealy vs. Moore, deterministic vs non-deterministic, etc
- Extended FSMs: state variables (variables that are **not** inputs or outputs) can appear in guards and actions

We will use **deterministic, extended** FSMs as defined by Lee/Seshia

- Will be useful when we talk about modeling
- Translate well to coding

Adjustable-height desk FSM

Whiteboard

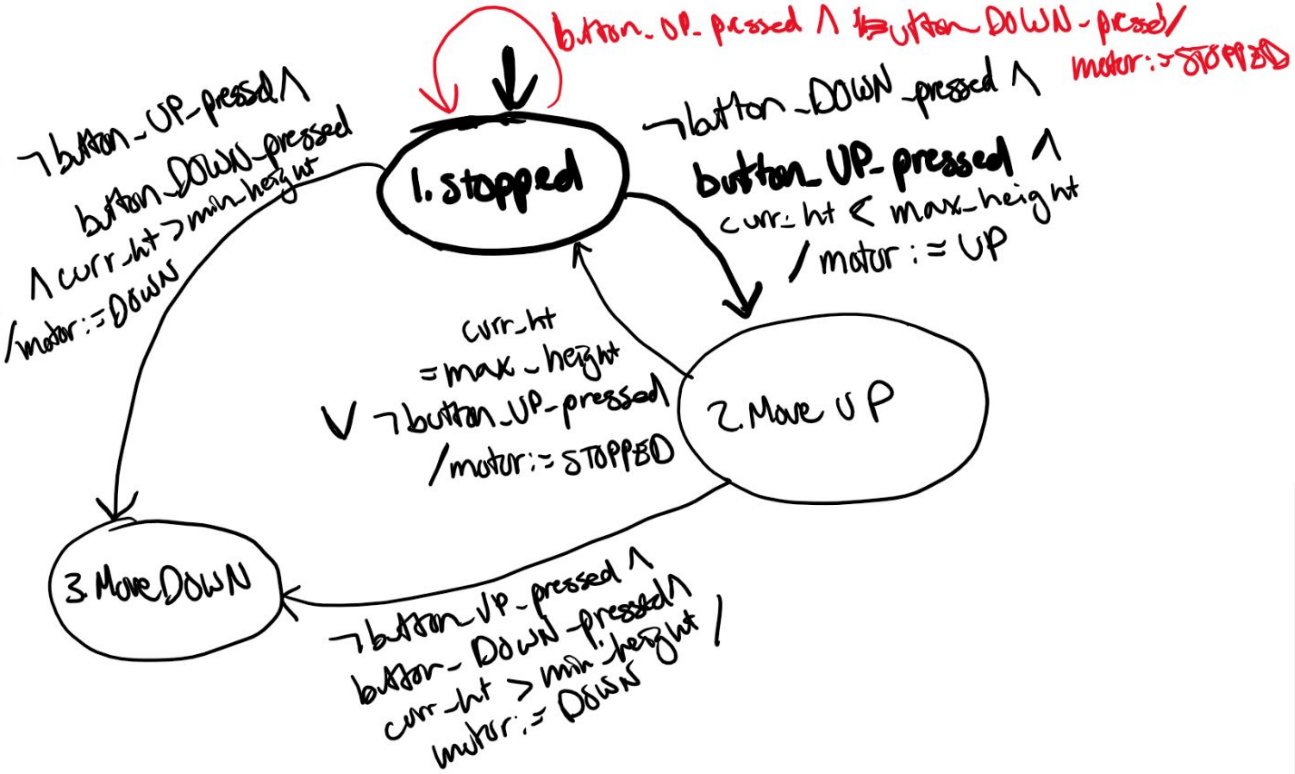
Inputs:
Button (1, 2, 3, UP, DOWN, M) pressed: boolean
Current desk height: fixed-point number

Outputs:
Motor command: (UP, DOWN, STOPPED): enum
LED display: fixed-point number or NONE

Constants:
max_height, min_height (fixed point numbers)

Variables (for extended FSM):
Presets 1, 2, 3 (fixed-point numbers)

Initial values:
Motor command: STOPPED
LED display: NONE
Presets 1/2/3: min_height



Note: this is not the complete FSM, it is what we had from lecture after talking through some considerations.

Reminders

Fill out the parts ordering form!

Fill out the capstone form (if capstoning)

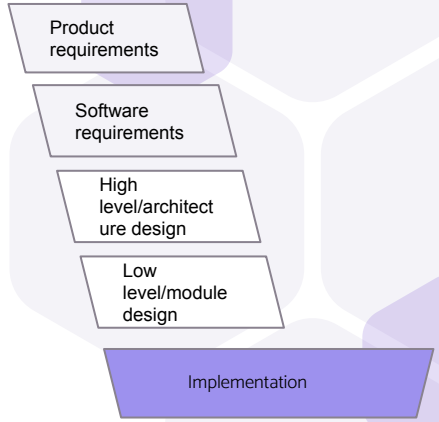
FSM conventions/rules

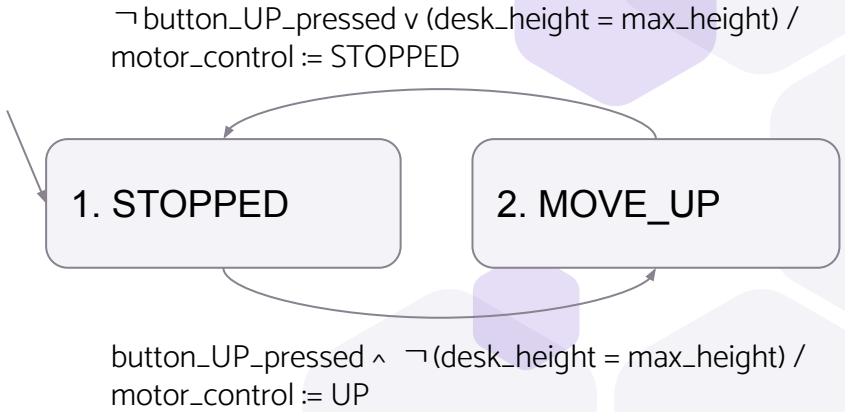
- Define inputs, outputs, and variables
  - Define initial values for outputs/variables
- Label each state with a number and a short, descriptive name
  - Label the start state
- Guards for transitions out of a state:
  - should be mutually exclusive
  - should only be predicated on inputs and variables
- Outputs on transitions should only set outputs and variables

Implementation of FSM

```
typedef enum { STOPPED = 1, MOVE_UP = 2 } state;

state update_fsm(state current_state, bool button_UP_pressed, ...) {
  state next_state;
  switch(current_state) {
    case STOPPED:
    // transition 1-2
    if (button_UP_pressed && desk_height != max_height) {
        next_state = MOVE_UP;
        set_motor_control(MOTOR_UP)
    } else {
        next_state = current_state
    }
    break; // important, or this goes to the next case
    case MOVE_UP:
    ...
    default:
      error("wrong state!");
  }
  return next_state;
}
```





When should update_fsm be called?

```
state update_fsm(state current_state,
    bool button_UP_pressed, …)
```

Time-triggered vs. event-triggered design

Time-triggered: computation to (potentially) change state happens every x ms, regardless if inputs have changed

Event-triggered: computation to (potentially) change state happens when an input changes

Pros/cons of time- vs. event-triggered design?

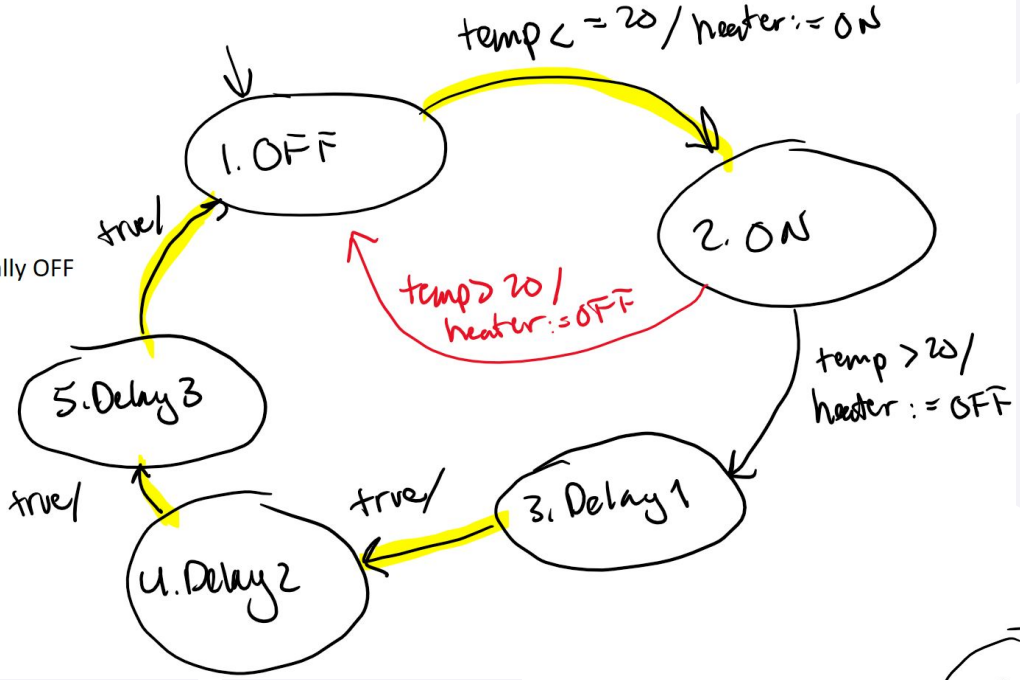
Another FSM example: HW problem

Consider a variant of the thermostat of example 3.5. In this variant, there is only one temperature threshold, and to avoid chattering the thermostat simply leaves the heat on or off for at least a fixed amount of time. In the initial state, if the temperature is less than or equal to 20 degrees Celsius, it turns the heater on, and leaves it on for at least 30 seconds. After that, if the temperature is greater than 20 degrees, it turns the heater off and leaves it off for at least 2 minutes. It turns it on again only if the temperature is less than or equal to 20 degrees

Design an FSM that behaves as described, assuming it reacts exactly once every 30 seconds.

nputs:
Current temp

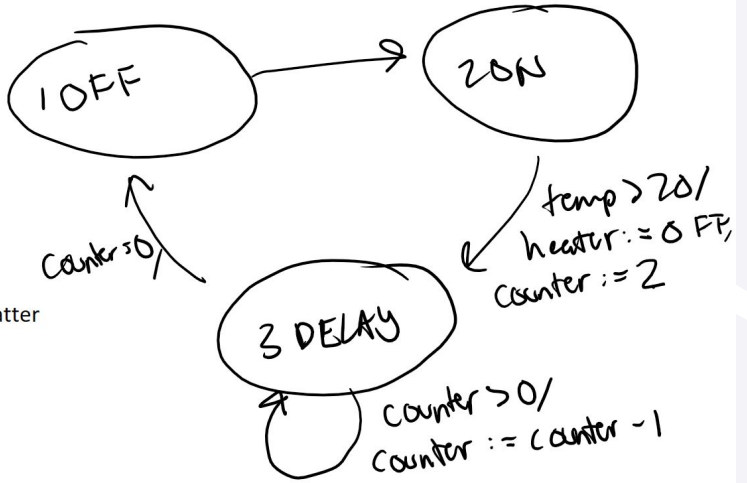
Outputs:
Heater (ON/OFF); initially OFF



Extended FSM (alternative design)
Inputs:
Current temp

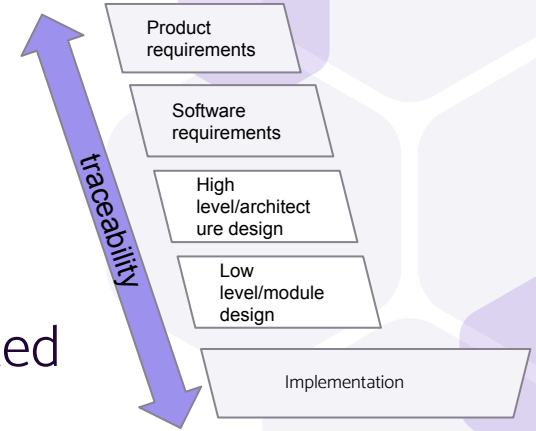
Outputs:
Heater (ON/OFF); initially OFF

Variables: counter; initial value doesn't matter



How do we know that our design has met our requirements?

Traceability



Ensures that all requirements have been implemented and tested

Often done using a **traceability matrix**

- Example: each column is a requirement; each row is a transition
- “x” in a cell if the transition helps meet the requirement
- If a column has no x’s, means requirement isn’t being met
- If a row has no x’s, means transition is unnecessary (or requirement is missing/wasn’t stated!)

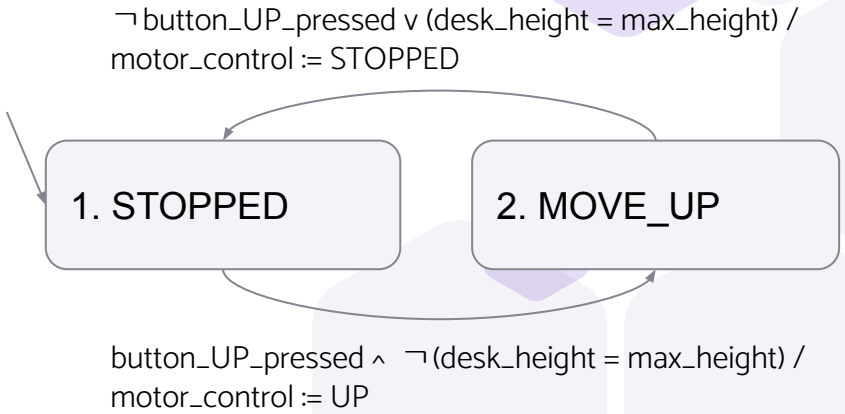
Example: requirements to FSM traceability

R1: If the desk is not at its maximum height, and the up button is held, the motor shall be commanded UP

R2: If the M button is pressed and released, and one of the numbered buttons [1, 2, 3] is pressed and released within 10 seconds, then the current height shall be stored as a preset for the corresponding numbered button

...

	R1	R2	R3
T 1-2	X		
T 2-1	X		



Summary

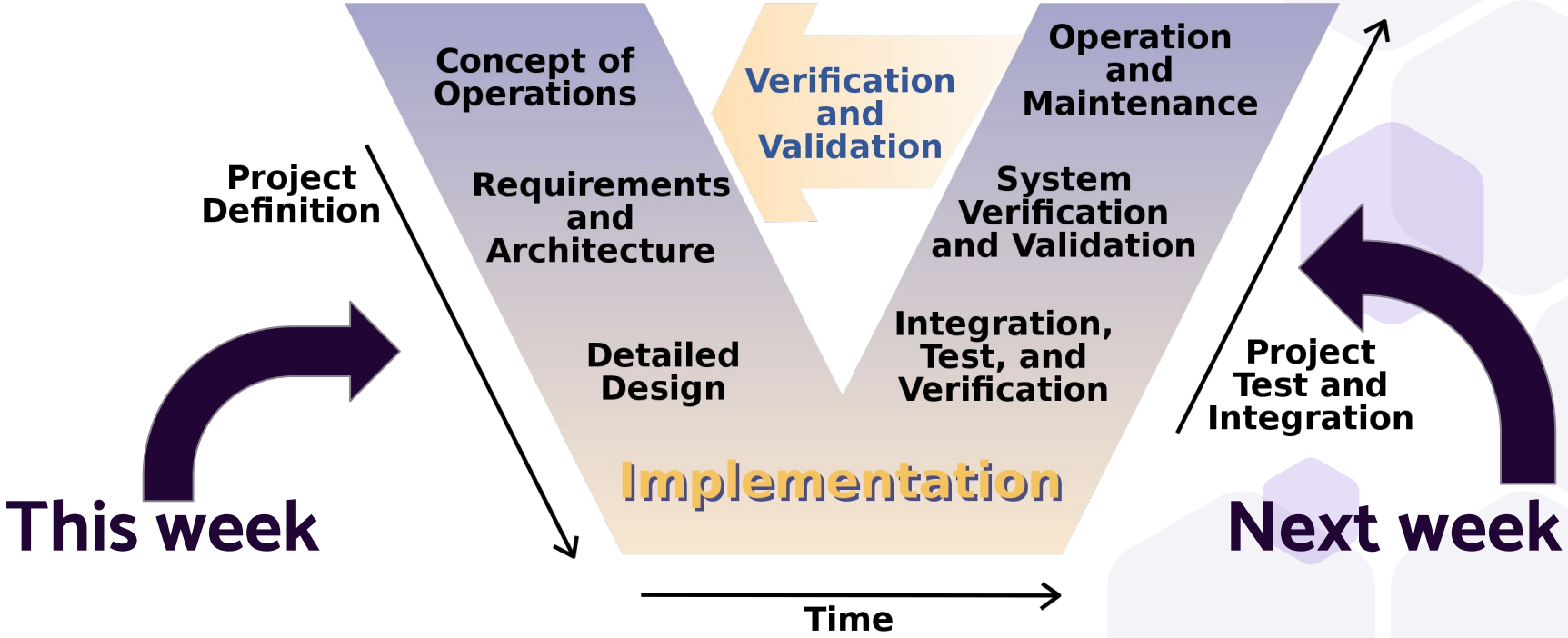


Image source