Modal Systems & Statecharts

“When you come to a fork in the road – take it.”
— Yogi Berra
Anti-Patterns:
- No detailed design; just code
- Deeply nested if statements instead of switch statements for state-full code
- Mixing mode change logic with normal output sequences

Detailed design of state-intensive behaviors
- Operating modes, e.g., stop, start, run
- Inputs that drive sequences of events
- Key technique: statecharts (software finite state machine)
Example code for 3-speed fan

- Draw a flowchart -- how easy is it to understand this code?
- Are there any bugs in this code?

```
// Change: input true on cycle when speed change button depressed
// OnOff : input true one cycle when on/off switch depressed
static uint8_t speed; // 0=Off; 1=Slow; 2=Medium; 3=Fast

.....
if(speed == 0)
{ if(Change == 1 || OnOff == 1) { speed = 1; } }
else if (Change == 1)
{ if (speed == 1) { speed = 2; }
  else if (speed == 2) { speed = 3; }
  else { speed = 0; }
}
else if (OnOff == 1)
{ speed = 0; }
```
A statechart is a software Finite State Machine:
- Set of states with side effects
- Set of guards that cause transitions
  - No side effects on transitions
- Initial state

Convert example fan code to statechart
- (See following four slides)
- Define a state for each fan speed
- Define transitions
- Easier to understand? Any bugs?
Exercise: One Button 3-speed Fan

- CHANGE button
- Output: Speed

System Reset

- S1. OFF
  - Speed ↔ Stop
- S2. SLOW
  - Speed ↔ Slow
- S3. MEDIUM
  - Speed ↔ Med
- S4. FAST
  - Speed ↔ Fast
Exercise: Two Button 3-speed fan

- CHANGE button
- ONOFF button
- Output: Speed

System Reset

S1. OFF
Speed ↔ Stop

S2. SLOW
Speed ↔ Slow

S3. MEDIUM
Speed ↔ Med

S4. FAST
Speed ↔ Fast

CHANGE or ONOFF

ONOFF

CHANGE

CHANGE
static enum CurrState {OFF, SLOW, MEDIUM, FAST}; // define states
static const uint8_t SpdOff = 0; // define speed constant values
static const uint8_t SpdSlow = 10;
static const uint8_t SpdMed = 15;
static const uint8_t SpdFast = 25;
CurrState = OFF; // initialize state machine to OFF

void ProcessStates (void) // run periodically from main loop
{
    switch (CurrState)
    {
    case OFF: // State S1
        speed(SpdOff); // Take action in state
        // Test arc guards and take transitions
        if (SpdButton() == TRUE || OnOffButton() == TRUE) { CurrState = SLOW;}
        break; // go to end of switch statement
    case SLOW: // State S2
        speed(SpdSlow); // take action
        if (SpdButton() == TRUE) { CurrState = MEDIUM;}
        if (OnOffButton() == TRUE) { CurrState = OFF;}
        break;
    }
case MEDIUM: // State S3
    speed(SpdMed); // take action
    if (SpdButton() == TRUE) { CurrState = FAST;}
    if (OnOffButton() == TRUE) { CurrState = OFF;}
    break;

case FAST: // State S4
    speed(SpdFast); // take action
    if (SpdButton() == TRUE) { CurrState = SLOW;}
    if (OnOffButton() == TRUE) { CurrState = OFF;}
    break;

default: // Error: invalid state
    error(INVALID_STATE_ERROR); // should never get here
}
Controller for a multi-speed motor or other similar application

- Inputs: CHANGE, ONOFF
- Outputs: Speed = \{Stop, Slow, Med, Fast\}
- State names (arbitrary labels): \{OFF, SLOW, MEDIUM, FAST\}
- System Reset is to state S1
RDRF = “Receive Data Register Full” \(\Rightarrow\) Data byte arrived
TDRE = “Transmit Data Register Empty” \(\Rightarrow\) Done sending

SCDR = “Serial Comms. Data Reg.”
XON/XOFF \(\Rightarrow\) Flow Control

Half-Duplex Serial Port Example

[Valvano 2006]
Best Practices for Statecharts

- Use statecharts for stateful code
  - Maps to easier-to-test switch statement
  - Avoid actions on arcs to simplify code
  - Move complex behaviors to per-state subroutine helper functions to limit cyclomatic complexity

- Summary of pitfalls
  - Some code is better as flowchart if there is no state history
  - Don’t let statechart get too complex
    - Might need to decompose into nested or parallel state machines
START

HEY, WAIT, THIS FLOWCHART IS A TRAP!

YES

https://xkcd.com/1195/