Control of Time-Varying Behavior

Can often express a “mission” in terms of a sequence of sub-tasks (or a plan)

• But: we also want to handle contingencies when they arrive

Finite state machines are a simple way of expressing such plans and contingencies
Finite State Machines (FSMs)

Pure FSM is composed of:

- A set of states
- A set of possible inputs (or events)
- A set of possible outputs (or actions)
- A transition function:
  - Given the current state and an input: defines the output and the next state
Finite State Machines (FSMs)

States:

• Represent all possible “situations” that must be distinguished
• At any given time, the system is in exactly one of the states
• There is a finite number of these states
Finite State Machines (FSMs)

An example: a 3-bit counter that increments when “count” input is received

- States: ?
Finite State Machines (FSMs)

An example: a counter

• States: the different combinations of the digits: 000, 001, 010, … 111

• Inputs: ?
Finite State Machines (FSMs)

An example: a counter

- Inputs (events):
  - Only one: “count”
  - We will call this “C”

- Outputs: ?
Finite State Machines (FSMs)

An example: a counter

- Outputs: same as the set of states
- Transition function: ?
Finite State Machines (FSMs)

An example: a counter

• Transition function:
  – On the count event, transition to the next highest value
FSM Example: Synchronous Counter

A Graphical Representation:

A set of states
FSM Example: Synchronous Counter

A transition

C/001

001

000

010

011

100

111

110

101
FSM Example: Synchronous Counter

A transition

The event
FSM Example: Synchronous Counter

A transition

The output
FSM Example: Synchronous Counter

A transition

The output: The Zyante book calls these “Mealy Actions”
FSM Example: Synchronous Counter

The next transition
FSM Example: Synchronous Counter

The next transition

FSM Example: Synchronous Counter

The next transition
FSM Example: Synchronous Counter

The full transition set

```plaintext
000 -> 001: C/001
001 -> 010: C/010
010 -> 011: C/011
011 -> 100: C/100
100 -> 101: C/101
101 -> 110: C/110
110 -> 111: C/111
111 -> 000: C/000
```

Diagram:

- Each state is represented by a circle.
- Arrows indicate the transitions between states.
- The labels on the arrows denote the input that triggers the transition.
FSM Example: Synchronous Counter

Initial condition

```
x/000
```

```
000 001 010 011 100 101 110 111
```

```
C/000 C/001 C/010 C/011 C/100 C/101 C/110 C/111
```

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
Example II: An Up/Down Counter

Suppose we have two events (instead of one): Count up and count down

• How does this change our state transition diagram?
Example II: An Up/Down Counter

From state 000, there are now two possible transitions

- From state 000, on the up transition (U/001), you can move to state 001.
- From state 000, on the down transition (D/111), you can move to state 111.

Possible states are:
- 000
- 001
- 010
- 011
- 100
- 101
- 110
- 111

100
Example II: An Up/Down Counter

Likewise for state 001…

```
000
     U/001

001
     U/010

010
     D/000

011
     D/111

100

101

110

111
```
Example II: An Up/Down Counter

The full transition set

- U/000 to U/001
- D/000 to D/001
- U/010 to U/011
- D/010 to D/011
- U/100 to U/101
- D/100 to D/101
- U/110 to U/111
- D/110 to D/111
- U/111 to U/000
- D/111 to D/000
FSMs and Control

How do we relate FSMs to Control?

• States are?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are some processed representation of what the sensors are observing

• Outputs are?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are some processed representation of what the sensors are observing

• Outputs are the control actions
  – These are typically “high level” actions: e.g., set the goal orientation to 125 degrees
FSMs: A Control Example

Suppose we have a vending machine:

- Accepts dimes and nickels
- Will dispense one of two things once $.20 has been entered: Jolt or Buzz Water
  - The “user” requests one of these by pressing a button
- Ignores select if < $.20 has been entered
- Immediately returns any coins above $.20
Vending Machine FSM

What are the states?
Vending Machine FSM

What are the states?

- $0
- $.05
- $.10
- $.15
- $.20
Vending Machine FSM

What are the inputs/events?
Vending Machine FSM

What are the inputs/events?

- Input nickel (N)
- Input dime (D)
- Select Jolt (J)
- Select Buzz Water (BW)
Vending Machine FSM

What are the outputs?
Vending Machine FSM

What are the outputs?

- Return nickel (RN)
- Return dime (RD)
- Dispense Jolt (DJ)
- Dispense Buzz Water (DBW)
- Nothing (Z)
Vending Machine Design

What is the initial state?
Vending Machine Design

What is the initial state?

- $S = 0$
Vending Machine Design

What can happen from $S = 0$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = $0$? 

What does this part of the diagram look like?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.05</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.10</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$0</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$0</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:
Vending Machine Design

What can happen from $S = $0.05?  

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from S = $0.05?

What does the modified diagram look like?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.10</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.15</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$.05</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$.05</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:

\[\text{x/Z} \quad \text{N/Z} \quad \text{J/Z} \quad \text{BW/Z} \quad \text{D/Z} \quad \text{N/Z} \quad \text{D/Z} \quad \text{J/Z} \quad \text{BW/Z} \]

\[\text{$.05} \quad \text{$.10} \quad \text{$.15} \]

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
Vending Machine Design

What can happen from $S = $0.10?  

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Vending Machine Design

What can happen from $S = 0.10$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.15</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$.10</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$.10</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:
Vending Machine Design

What can happen from $S = $0.15?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = \$0.15$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.20</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>RN</td>
</tr>
<tr>
<td>J</td>
<td>$.15</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$.15</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:
Finally: what can happen from S = $0.20?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

Finally, what can happen from $S = \$.20$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.20</td>
<td>RN</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>RD</td>
</tr>
<tr>
<td>J</td>
<td>$0</td>
<td>DJ</td>
</tr>
<tr>
<td>BW</td>
<td>$0</td>
<td>DBW</td>
</tr>
</tbody>
</table>
Vending Machine Design

The complete state diagram:
Finite State Machines

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
FSM Design Pattern

- The system is always in exactly one state
- Think of transitions as happening instantaneously
FSM Design Pattern

Think of transitions as happening instantaneously

• FSM actions are also instantaneous

• For an activity that must take a finite amount of time:
  – The FSM action is to initiate the activity
  – The next state is one in which the system is waiting for activity completion
  – The next event signals completion
Vending Machine FSM

$0$

$.05$

$.10$

$.15$

$.20$

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
A Robot Control Example

Consider the following task:

• The robot is to move toward the first beacon that it “sees”
• The robot searches for a beacon in the following order: right, left, front
• Once beacon is found, move toward it and stop once the beacon is reached

What is the FSM representation?
Robot Description

Mobile robot with sensor turret on top

- Mobile robot turns take time
- Turret turns are relative to the mobile base and do not take time
Events

• Robot Turn Complete (TC)
• Beacon (B)
• No Beacon (NB)
Actions

• Look left (LL): turn turret to be facing left (relative to the mobile base)
• Look right (LR)
• Look forward (LF)
• Turn left (TL): initiate a turn of the robot base by 90 degrees to the left
• Turn right (TR): initiate right turn
• Move forward (F): initiate forward movement
• Stop (S)
Robot Control Example II

Consider the following task:
- The robot must lift off to some altitude
- Translate to some location
- Take pictures
- Return to base
- Land
- At any time: a detected failure should cause the craft to land

What is the FSM representation?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs/events are some processed representation of what the sensors are observing

• Outputs are the control actions
FSMs in C
Implementation in the Arduino environment

```c
void loop()
{
    static PeriodicAction fsm_task(50, fsm_step);

    // Check to see if 50 ms has gone by
    // If so, then the function
    // fsm_step() is called
    fsm_task.step();
}
```
FSMs in C

```c
void fsm_step() {
    static State state = STATE_0;  // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_0:
            <handle state 0>
            break;
        case STATE_1:
            <handle state 1>
            break;
        case STATE_2: ...
    }
}
```
Creating an Enumerated Variable Type

• Definition:

typedef enum {
    STATE_0, STATE_1, STATE_2
} State;

• Use:

State s = STATE_1;

s can only take on these 3 values
Locally Defined Variables

- Local variables defined inside of a function are allocated to memory only when the function is called
  - Memory region called *the stack*
- When the function returns, the memory is reclaimed for use by other functions
Static Variables

Declaring a variable inside a function as static:

```c
static State state = STATE_0;  // Initial state
```

- The variable acts like a global variable:
  - The memory continues to exist after a return from the function
  - This means that the value from the last call to the function can be used in the next call
  - But: the variable can only be “seen” by this function
Static Variables

Declaring a variable inside a function as static:

```c
static State state = STATE_0;  // Initial state
```

- Other key thing to remember: the assignment is executed exactly once (before the main() function is executed)
- We can use this to set the initial value of the static variable
FSMs in C

```c
fsm_step() { 
    static State state = STATE_0;  // Initial state

    <do some processing of the sensory inputs>
    switch(state) { 
        case STATE_0: 
            <handle state 0>
            break;
        case STATE_1: 
            <handle state 1>
            break;
        case STATE_2: ... 
    }
}
```
FSMs in C
(integrating with other code)

```c
fsm_step() {
    static State state = STATE_0;  // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_0:
            <handle state 0>
            break;
        case STATE_1:
            <handle state 1>
            break;
        case STATE_2: ...
    }

    <do some low-level control>
}
```
Handling Each State

• You will need to provide code that handles the event processing for each state

• Specifically:
  – You need to handle each event that can occur
  – For each event, you must specify:
    • What action is to be taken
    • What the next state is
Handling Each State

In our vending machine example:

- Events are easy to describe (only a few things can happen)
- We will write an “if()” for each of these cases (or combination of cases)
Vending Machine Definitions

typedef enum {
    STATE_0cents, STATE_5cents,
    STATE_10cents, STATE_15cents,
    STATE_20cents
} State;

typedef enum {
    EVENT_NICKEL, EVENT_DIME,
    EVENT_JOLT, EVENT_BUZZ, EVENT_NONE
} Event;
FSMs in C

```c
fsm_step() {
    static State state = STATE_0cents;  // Initial

    // Translate sensors into event
    Event event = read_sensors();

    // Execute code for the current state
    switch(state) {
        case STATE_0cents:
            <handle state>
            break;
        case STATE_5cents:
            <handle state>
            break;
        case STATE_10cents: ...
    }
}
```
FSMs in C: Processing for a Single State

```c
: case STATE_10cents:
    // $.10 has already been deposited
    if(event == EVENT_NICKEL) {
        state = STATE_15cents; // Transition to $.15
    }
    else if(event == EVENT_DIME) {
        state = STATE_20cents; // Transition to $.2
    }
    else if(event == EVENT_JOLT || event == EVENT_BUZZ) {
        display_NOT_ENOUGH();
    }
    else {
        // Do nothing
    }
    break;
```
Some events do not fall neatly into one of several categories

• So: the “if()” becomes more interesting
• Also: time can be an aspect of an event
FSMs in C

```c
fsm_step() {  
    static State state = STATE_0;  // Initial state
    static int counter = 0;
    ++counter;

    <do some processing of the sensory inputs>
    switch(state) {  
    case STATE_MISSION_PHASE_3:
        <handle phase 3>
        break;
    case STATE_MISSION_PHASE_4 :
        <handle phase 4>
        break;
    case STATE_MISSION_PHASE_5 :
        :
    }
}
```
FSMs in C: Processing for Individual States

```c
: case STATE MISSION_PHASE_3:
    if(heading_error < 10.0 &&
        heading_error > -10.0)
    {
        // Move forward!
        desired_velocity = .2; // Action
        // Transition
        state = STATE MISSION_PHASE_4;
    }
    break;
:
```
FSMs in C: Processing for Individual States

```c
: case STATE_MISSION_PHASE_4:
    if (distance_left < 20.0 ||
        distance_right < 20.0)
    {
        // Brake!
        desired_velocity = 0;
        counter = 0;    // Reset the clock
        // Transition
        state = STATE_MISSION_PHASE_5;
    }
    break;
```
FSMs in C

fsm_step() {
    static State state = STATE_0;  // Initial state
    static int counter = 0;
    counter++;

    switch(state) {
        case STATE_MISSION_PHASE_3:
            //handle phase 3
            break;
        case STATE_MISSION_PHASE_4:
            //handle phase 4
            break;
        case STATE_MISSION_PHASE_5:
            :
    }
}
FSMs in C: Processing for Individual States

```c
case STATE_MISSION_PHASE_5:
    if(counter > 20)
    {
        // A fixed amount of time has gone by
        heading_goal = heading_goal - 90.0;
        if(heading_goal <= -180.0)
            heading_goal += 360;

        // Transition
        state = STATE_MISSION_PHASE_6;
    }
    break;
```

How much time has gone by?
FSMs in C: Processing for Individual States

```c
case STATE MISSION_PHASE_5:
    if (counter > 20)
    {
        // A fixed amount of time has gone by
        heading_goal = heading_goal - 90.0;
        if (heading_goal <= -180.0)
            heading_goal += 360;

        // Transition
        state = STATE MISSION PHASE_6;
    }
break;
```

How much time has gone by? 1 sec
FSM Implementation Notes

• FSM code should not contain delays or waits
  – No delay_ms() or while(…){}
  – Remember that your FSM code will be called once per control cycle: use “if” to check for an event during that control cycle

• Use LEDs and/or Serial.print() to indicate current state
  – Do not print too much!

• Implement and test incrementally