A Mad Scientist's Guide to Finite State Machines

Wolfgang Faust

FSM? (Flying Spaghetti Monster?)

Finite — of which there are a finite number.

State — The information it's manipulating is represented by states it can be in...

Machine — System to manipulate information (e.g. Turing Machine)

States?

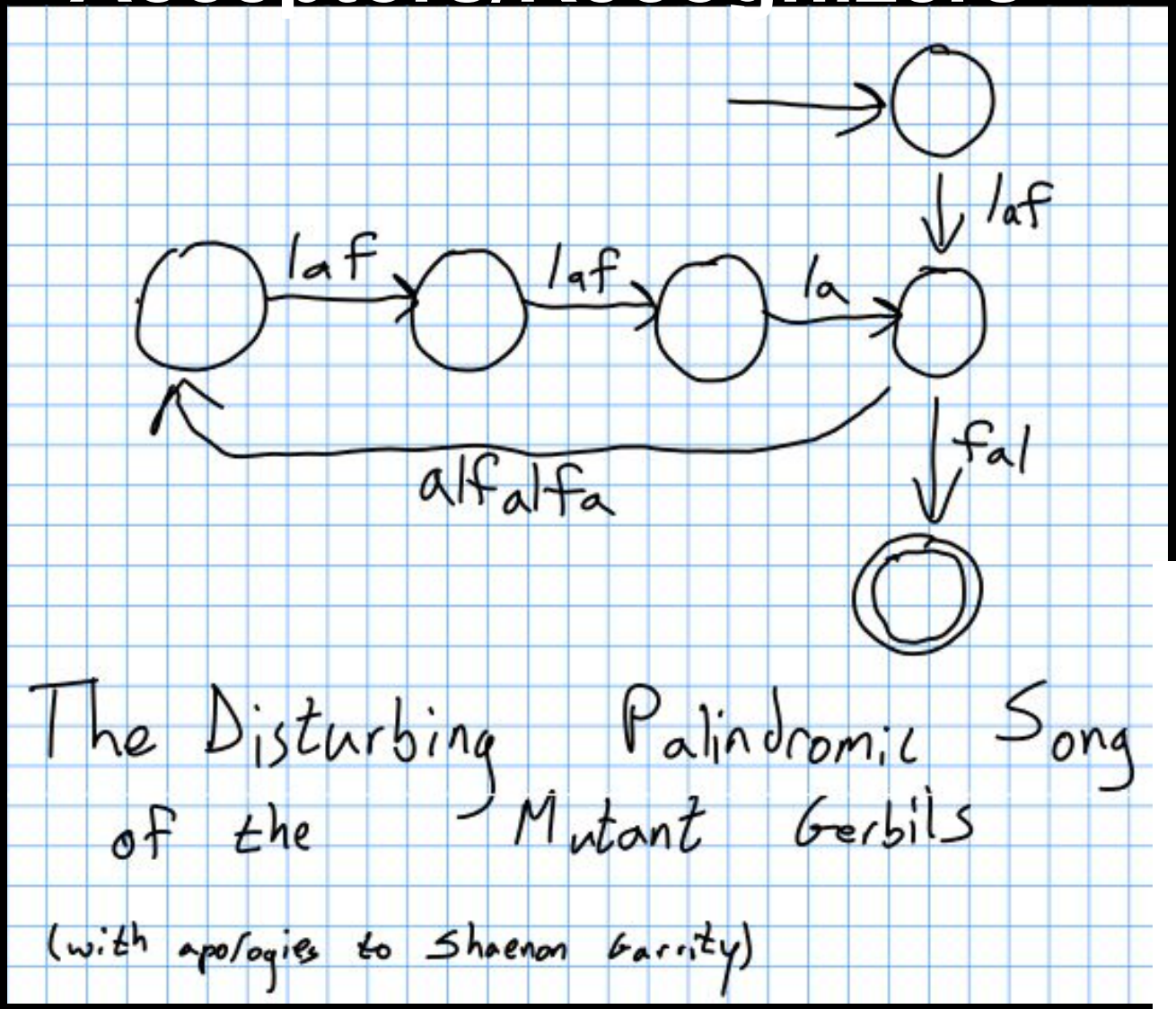
- opening, opened, closing, closed
- walk, continue_crossing, dont_walk
- humming_ominously, counting_down, stopped_at_one
- idle, attacking, fleeing, victorious, dead

Transitions

From one state to another

- `counting_down → stopped_at_one`
- `attacking → dead`
- `fleeing → victorious`

Acceptors/Recognizers

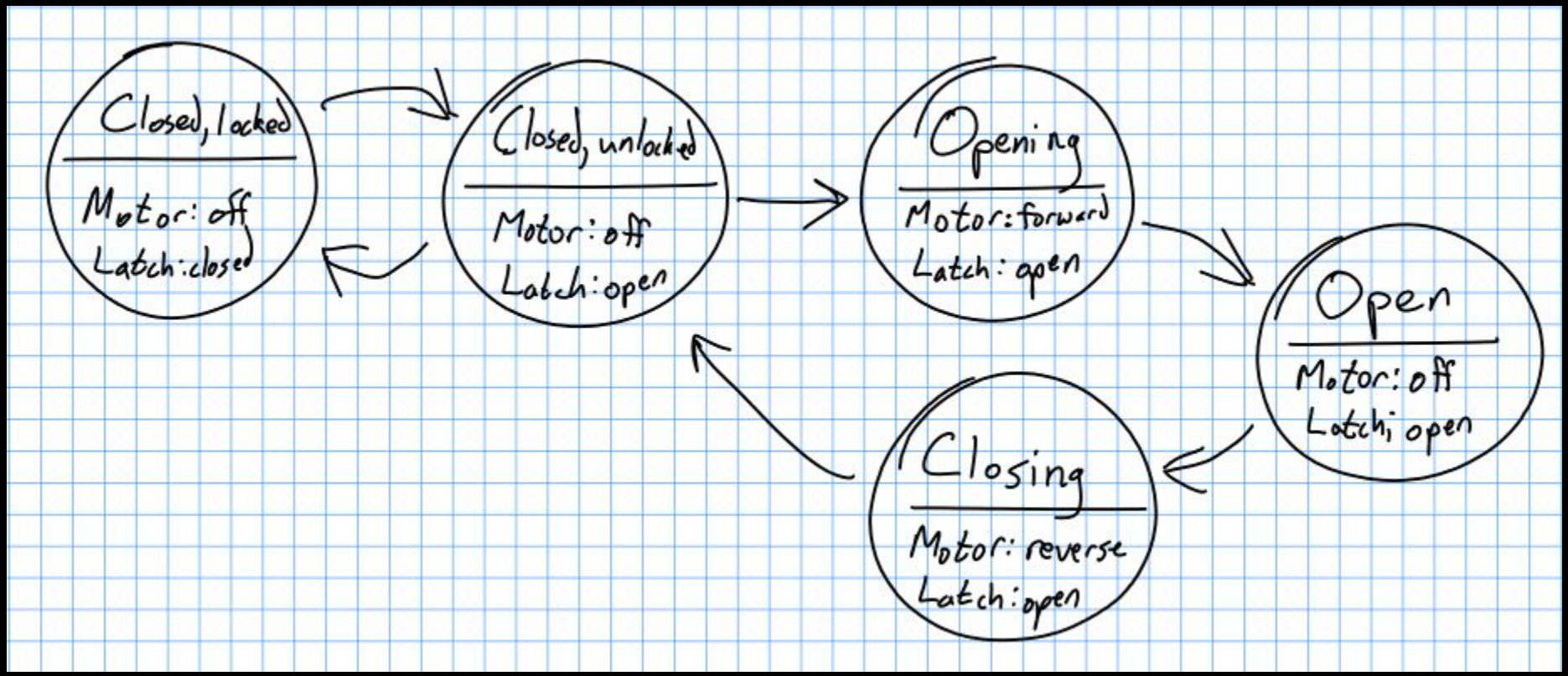


Inputs

Must be finite (digital); for analog values, translate:

```
LOW, HIGH
RISING, FALLING
TOO_COLD, TOO_HOT, JUST_RIGHT
```

Transducers



State transitions

```
switch (state):
   case S_CLOSED:
      if (pressed(BTN_LOCK)) state=S_LOCKED;
      if (pressed(BTN_MOVE)) state=S_OPENING;
   case S_OPEN:
      if (pressed(BTN_MOVE)) state=S_CLOSING;
   case S_LOCKED:
      if (pressed(BTN_LOCK)) state=S_UNLOCKED;
   case S_OPENING:
      if (bridge_is_open()) state=S_OPEN;
   case S_CLOSING:
      if (bridge_is_closed() state=S_CLOSED;
```

Transition tables

Current state	Event	Next state
Closed	Move button	Opening
Closed	Lock button	Locked
Locked	Lock button	Unlocked
Opening	Finished opening	Open
Open	Move button	Closing
Closing	Finished closing	Closed

Output signals

```
# State:         Cl Op Cg Og Lk
TABLE_LOCK  = {N, N, N,  N, Y};
TABLE_MOTOR = {0, 0, 1, -1, 0};

motor.set(TABLE_MOTOR[state]);
lock.set(TABLE_LOCK[state]);
```

A Useful Pattern for

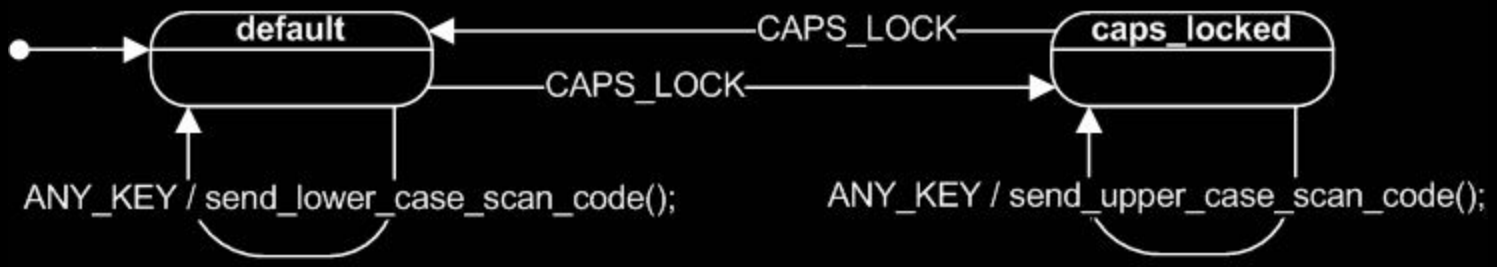
Entry/Exit Actions

```
void setState (int newState) {
    // Exit actions
    if (state == S_HUMMING_OMINOUSLY)
        stop_humming();

    // Switch state
    state = newState;

    // Entry actions
    if (state == S_COUNTING_DOWN)
        sound_klaxon(); // Sound horn when countdown begins
}
```

Ways to drive your FSM

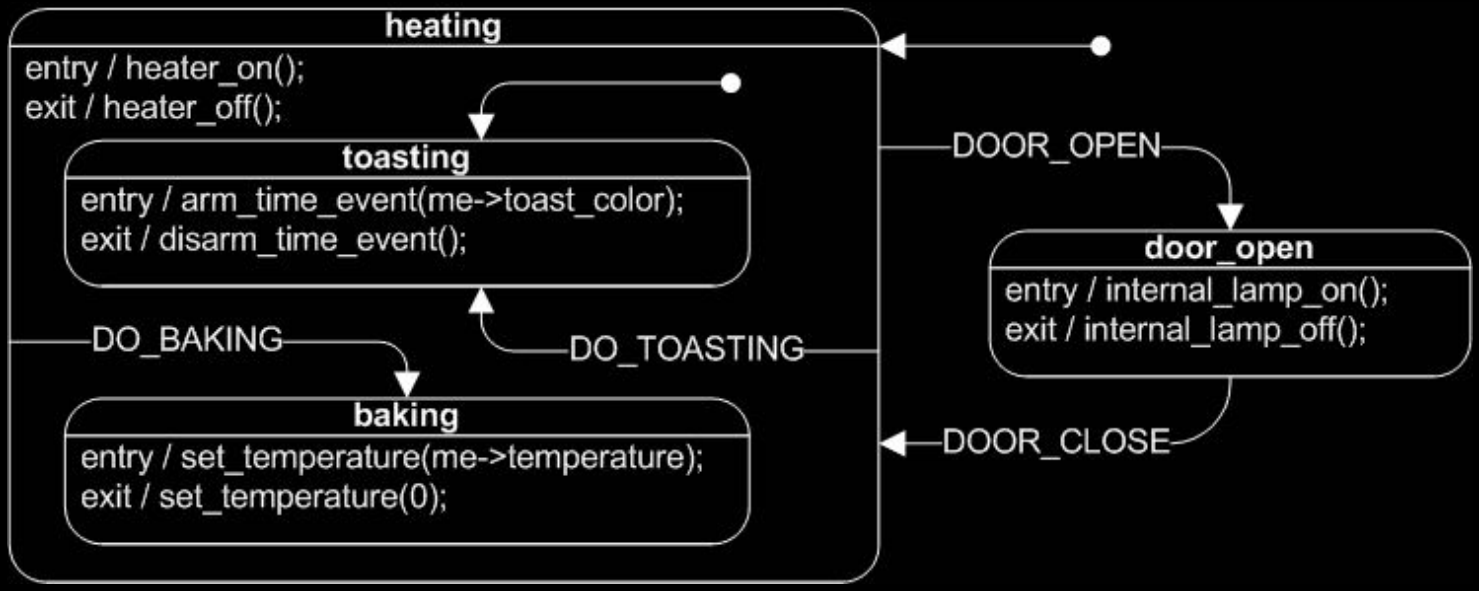
- ➔ Event loop
- ➔ Characters/tokens
- ➔ External events

Going further: UML State Machines



Going further: UML State Machines

Hierarchically Nested States



Going further: UML State Machines

Orthogonal regions