STATE MACHINES

LECTURE VI
TDDI11 Embedded Software

DEPT. COMPUTER AND INFORMATION SCIENCE (IDA)
LINKÖPINGS UNIVERSITET

INTRODUCTORY EXAMPLE: AN ELEVATOR CONTROLLER

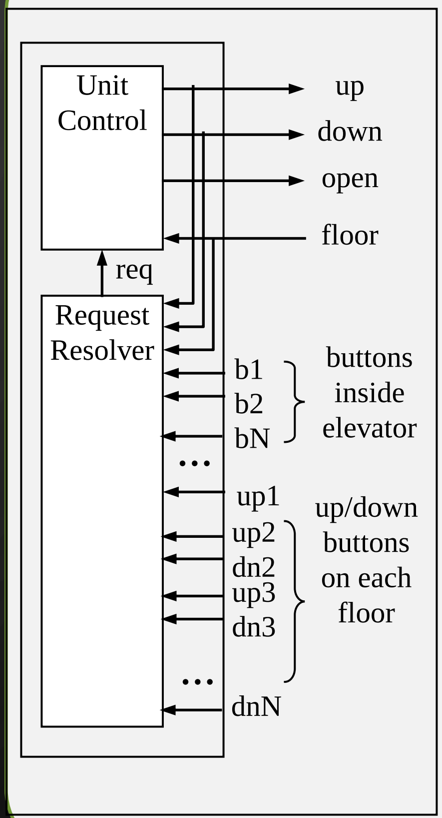
Simple elevator controller

- *Request Resolver* resolves various floor requests into single requested floor
- *Unit Control* moves elevator to this requested floor

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: A SEQUENTIAL PROGRAM MODEL

Sequential program model

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

```
void UnitControl()
{
  up = down = 0; open = 1;
  while (1) {
    while (req == floor);
    open = 0;
    if (req > floor) { up = 1;}
    else {down = 1;}
    while (req != floor);
    up = down = 0;
    open = 1;
    delay(10);
  }
}
```

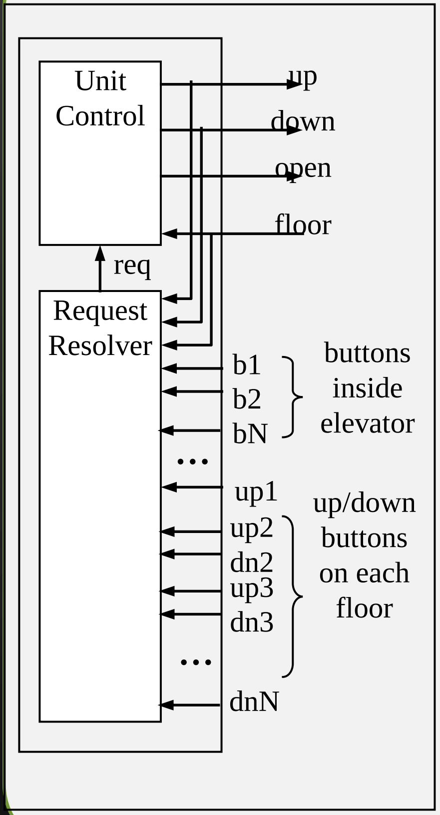
```
void
RequestResolver()
{
  while (1)
  ...
    req = ...
  ...
}
```

```
void main()
{
  Call concurrently:
    UnitControl() and
    RequestResolver()
}
```

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



Finite-State Machine (FSM) Model

- 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)

STATE MACHINE

- The system is described as a set of states
- Each state is a representation of what the system looks like now and how it got there
- Each state reacts in a specific way to *every possible* input event, leading to a new state via a transition

MEALY AND MOORE

- There are two kinds of state machines
  - Mealy and Moore
  - Both are finite state machines
  - Both can capture regular expressions

- Note: Finite state machines and finite automata are used interchangeably

MOORE MACHINES

A Moore machine is a tuple $(Q, \Sigma, \Gamma, \Delta, H, q_0)$ consisting in:

1. a finite set Q of states where q_0 is the start state
2. an alphabet Σ of input letters
3. an alphabet Γ of output characters
4. a mapping Δ associating a state in Q to each pair in $(Q \times \Sigma)$
5. a mapping H associating an output in Γ to each state in Q

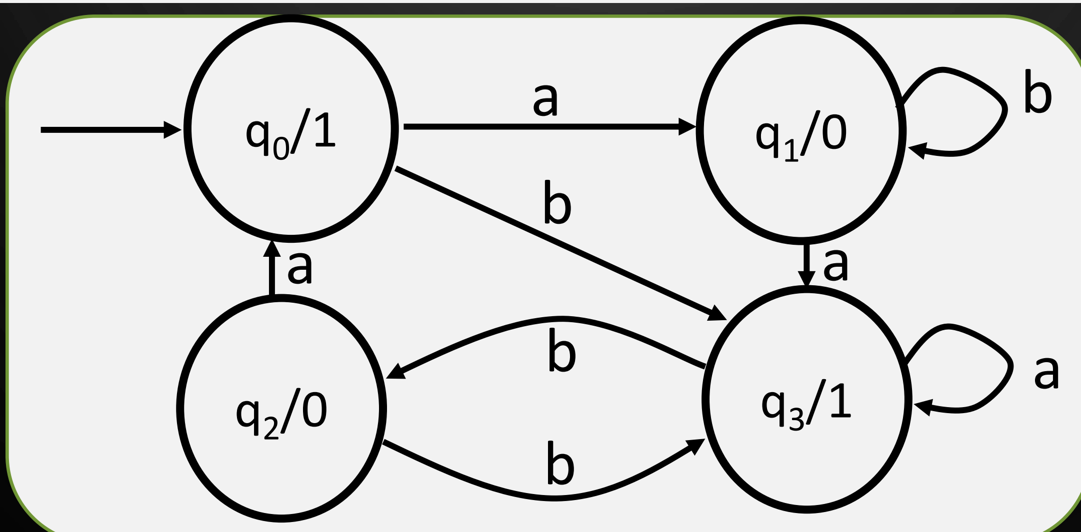
$$(q, \sigma) \xrightarrow{\Delta\text{: transition}} q' \quad \text{where } q, q' \in Q \text{ and } \sigma \in \Sigma$$

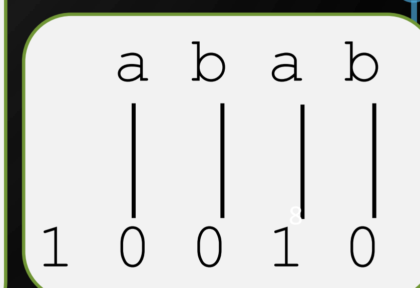
$$q \xrightarrow{H\text{: output}} o \quad \text{where } q \in Q \text{ and o} \in \Gamma$$

EXAMPLE OF A MOORE MACHINE

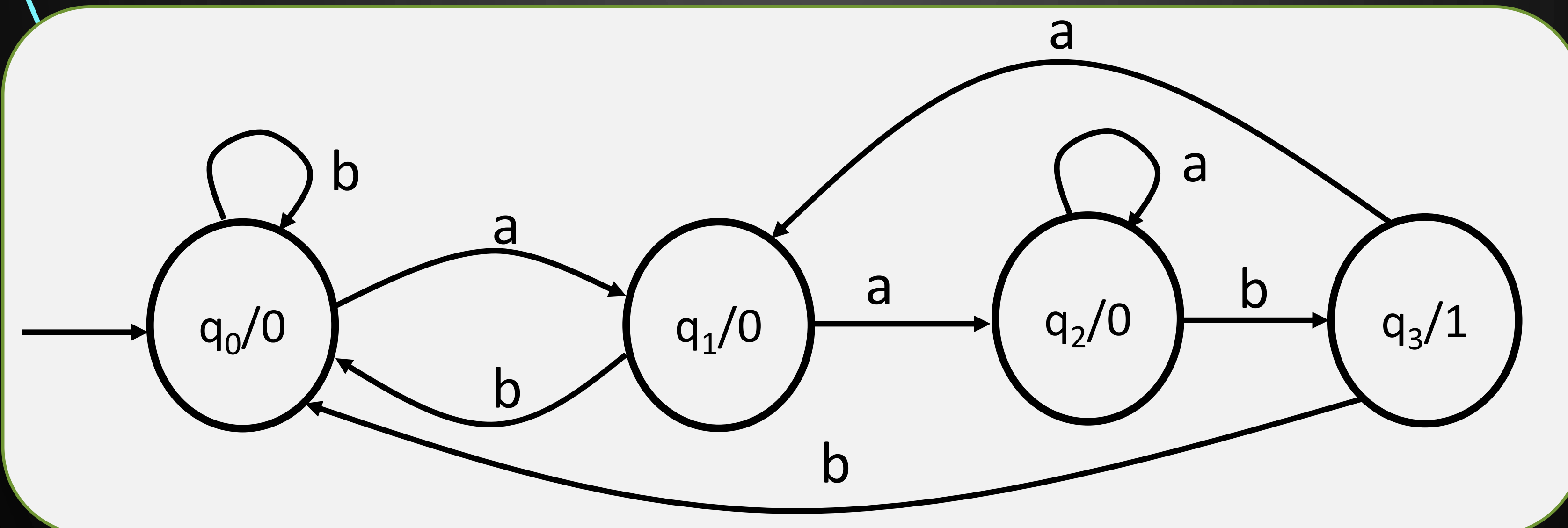
Example: states = $\{q_0,q_1,q_2,q_3\}$ $\Sigma=\{a,b\}$ $\Gamma=\{0,1\}$

Old state	Output by the old state	New state: After input a	New state: After input b
$\rightarrow q_0$	1	q_1	q_3
q_1	0	q_3	q_1
q_2	0	q_0	q_3
q_3	1	q_3	q_2





ANOTHER MOORE MACHINE



Input		a	a	a	b	a	b	b	a	a	b	b
State	q_0	q_1	q_2	q_2	q_3	q_1	q_0	q_0	q_1	q_2	q_3	q_0
Output	0	0	0	0	1	0	0	0	0	0	1	0

When does the machine output 1?

MEALY MACHINE

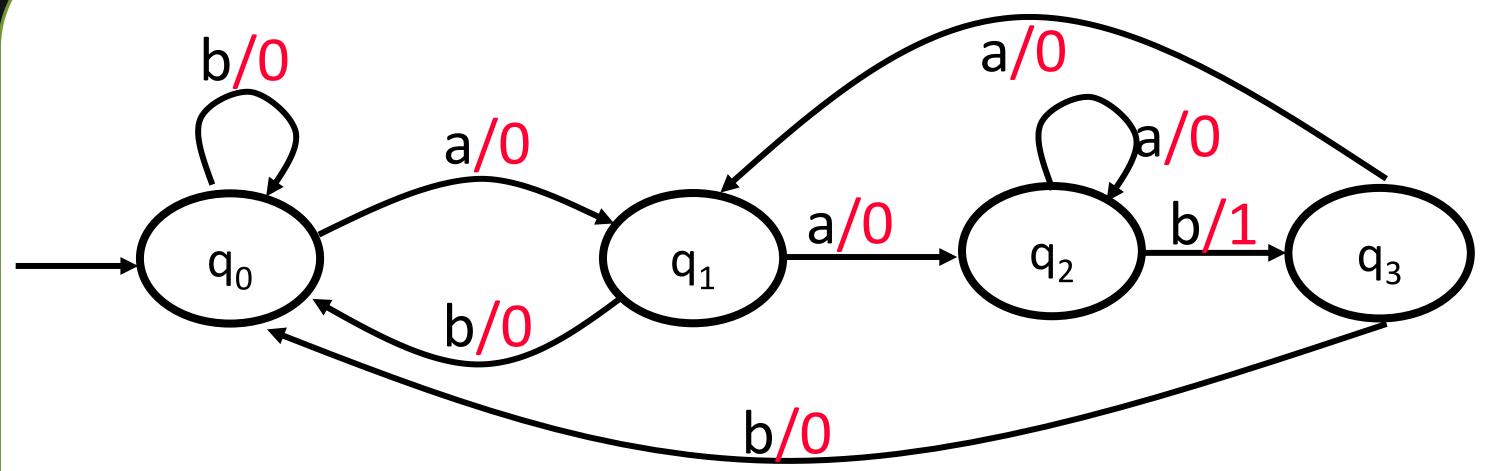
A Mealy machine is a tuple $(Q, \Sigma, \Gamma, \Delta, q_0)$ consisting in:

- a finite set of Q of states where q_0 is the start state
- an alphabet Σ of input letters
- an alphabet Γ of output characters
- a finite set of transitions Δ that indicate, for each state and letter of the input alphabet, the state to go to next and the associated output.

$(q, \sigma) \longrightarrow (q', o)$ where $q, q' \in Q, \sigma \in \Sigma$ and o $\in \Gamma$

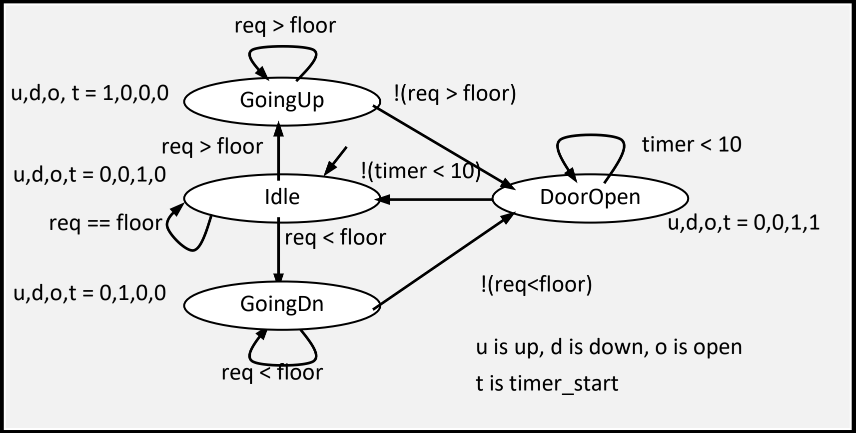
$q \xrightarrow{\sigma/o} q'$

MEALY MACHINE EXAMPLE



AN EXTENDED FINITE STATE MACHINE

FSMD extends FSM with Data: complex data types and variables for storing data



An FSMD is a tuple $(Q, \Sigma, \Gamma, V, \Delta, H, q_0)$:

1. *Q is a finite set of states*
2. Σ *is the set of possible* inputs
3. Γ is the set of possible outputs
4. V is a set of variables
5. Δ associates a state to any triple in $S \times \Sigma \times V$
6. H represents actions: it associates an output and values to V for each state

System state completely described by the current state and values of all variables

DESCRIBING A SYSTEM AS A STATE MACHINE

Sequential program model

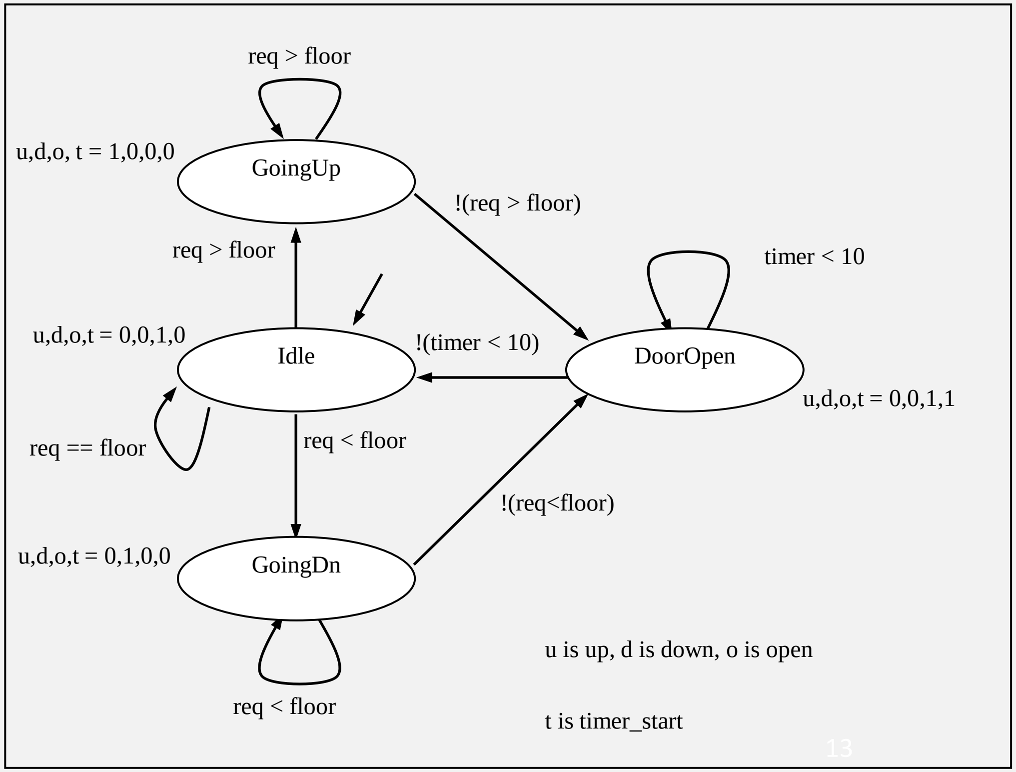
Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

```
void UnitControl()
{
  up = down = 0; open = 1;
  while (1) {
    while (req == floor);
    open = 0;
    if (req > floor) { up = 1;}
    else {down = 1;}
    while (req != floor);
    up = down = 0;
    open = 1;
    delay(10);
  }
}
```

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

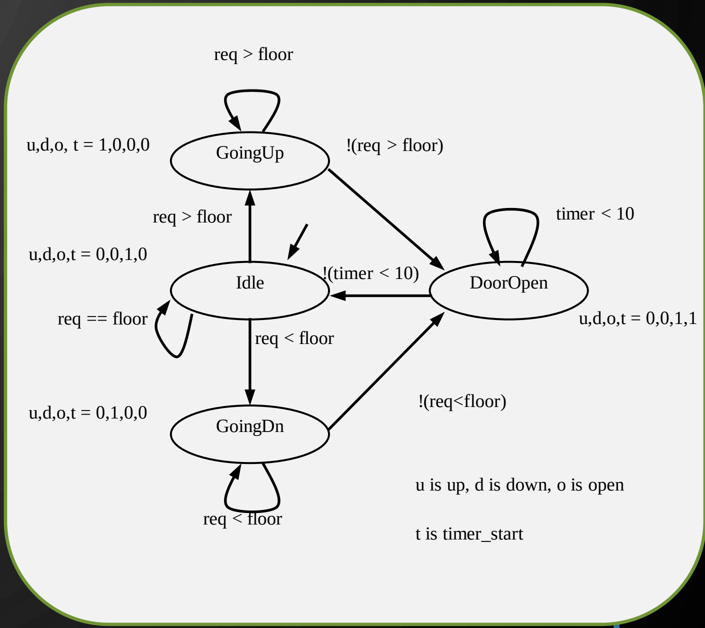
```
void main()
{
  Call concurrently:
    UnitControl() and
    RequestResolver()
}
```

State machine model



DESCRIBING A SYSTEM AS A STATE MACHINE

1. List all possible states
2. Declare all variables
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



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

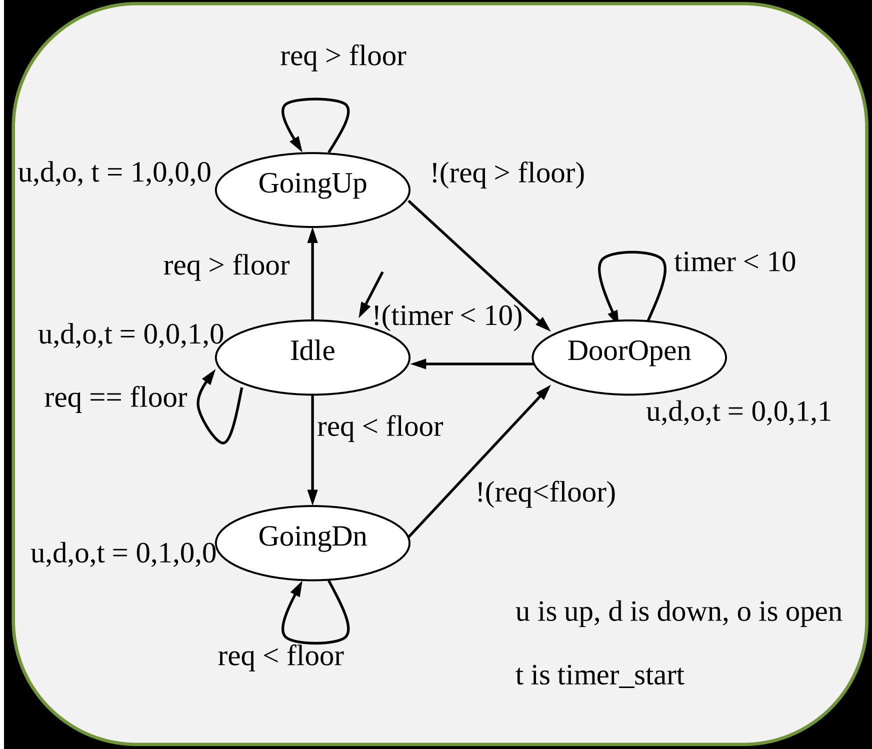
CAPTURING STATE MACHINES IN SEQUENTIAL PROGRAMMING LANGUAGE

- Despite benefits of state machine model, most popular development tools use sequential programming language
  - C, C++, Java, Ada, VHDL, Verilog, etc.
  - Development tools are complex and expensive, therefore not easy to adapt or replace

- Two approaches to capturing state machine model with sequential programming language
  - Front-end tool approach
  - Language subset approach

LANGUAGE SUBSET APPROACH

```c
#define IDLE 0
#define GOINGUP 1
#define GOINGDN 2
#define DOOROPEN 3
void UnitControl() {
  int state = IDLE;
  while (1) {
   switch (state) {
    IDLE: up=0; down=0; open=1; timer_start=0;
     if (req==floor) {state = IDLE;}
     if (req > floor) {state = GOINGUP;}
     if (req < floor) {state = GOINGDN;}
     break;
    GOINGUP: up=1; down=0; open=0; timer_start=0;
     if (req > floor) {state = GOINGUP;}
     if (!(req>floor)) {state = DOOROPEN;}
     break;
    GOINGDN: up=1; down=0; open=0; timer_start=0;
     if (req < floor) {state = GOINGDN;}
     if (!(req<floor)) {state = DOOROPEN;}
     break;
    DOOROPEN: up=0; down=0; open=1; timer_start=1;
     if (timer < 10) {state = DOOROPEN;}
     if (!(timer<10)){state = IDLE;}
     break;
```



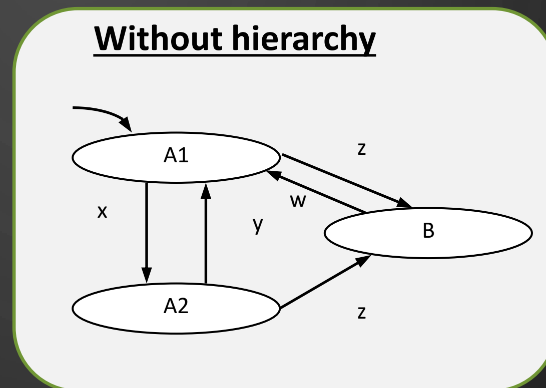
***UnitControl* state machine in sequential programming language**

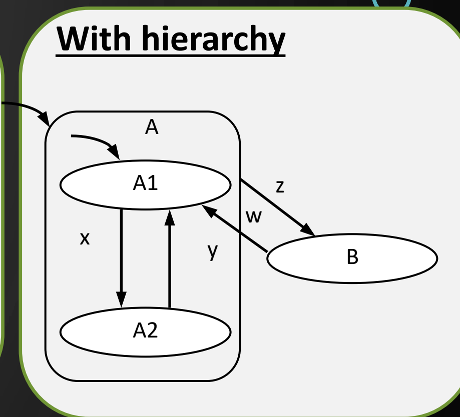
HIERARCHICAL/CONCURRENT STATE MACHINE MODEL

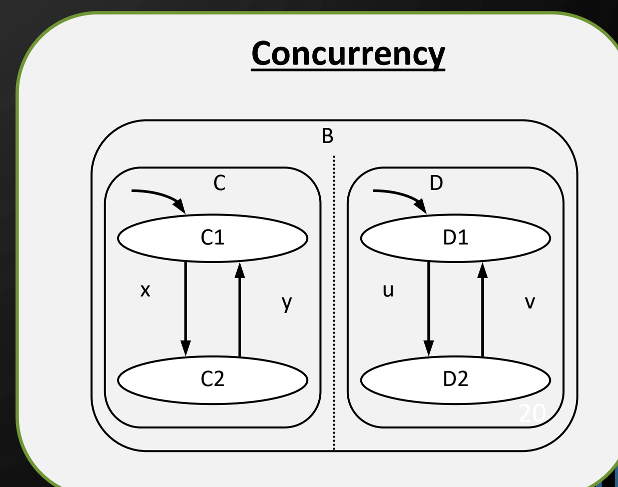
Another Model for state-based specification

- Uses Hierarchies and Concurrency to eliminate clutter and clarify the structure

Allows states to be grouped together



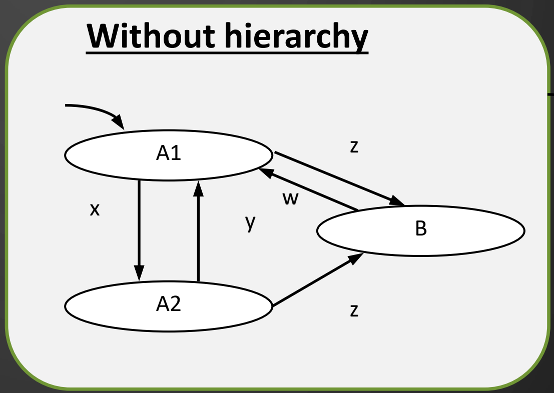


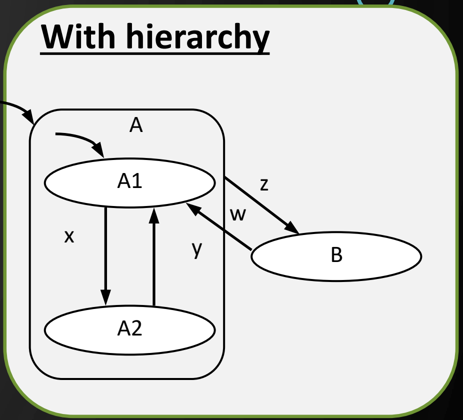


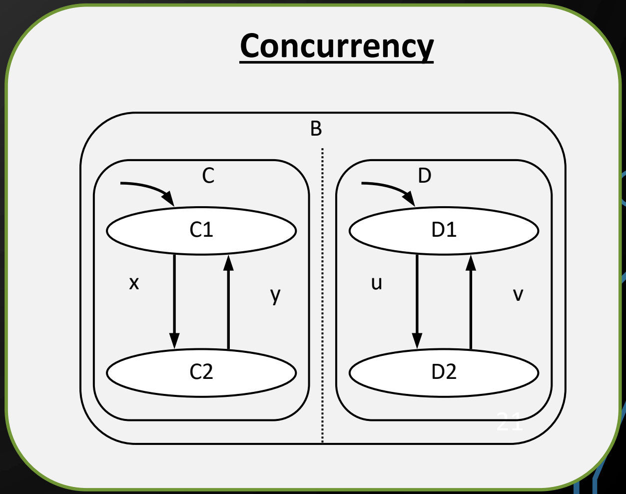
GROUPS IN STATECHART

- Statecharts define a language for HCFSMs

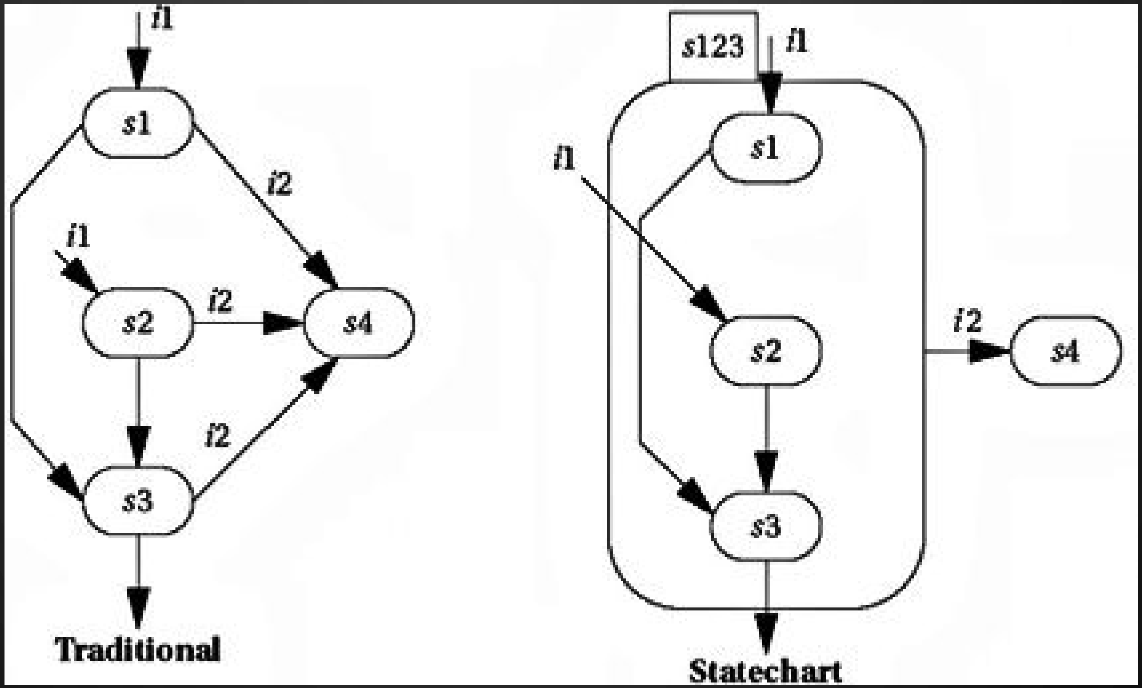
- Two groups are
  - OR
  - AND





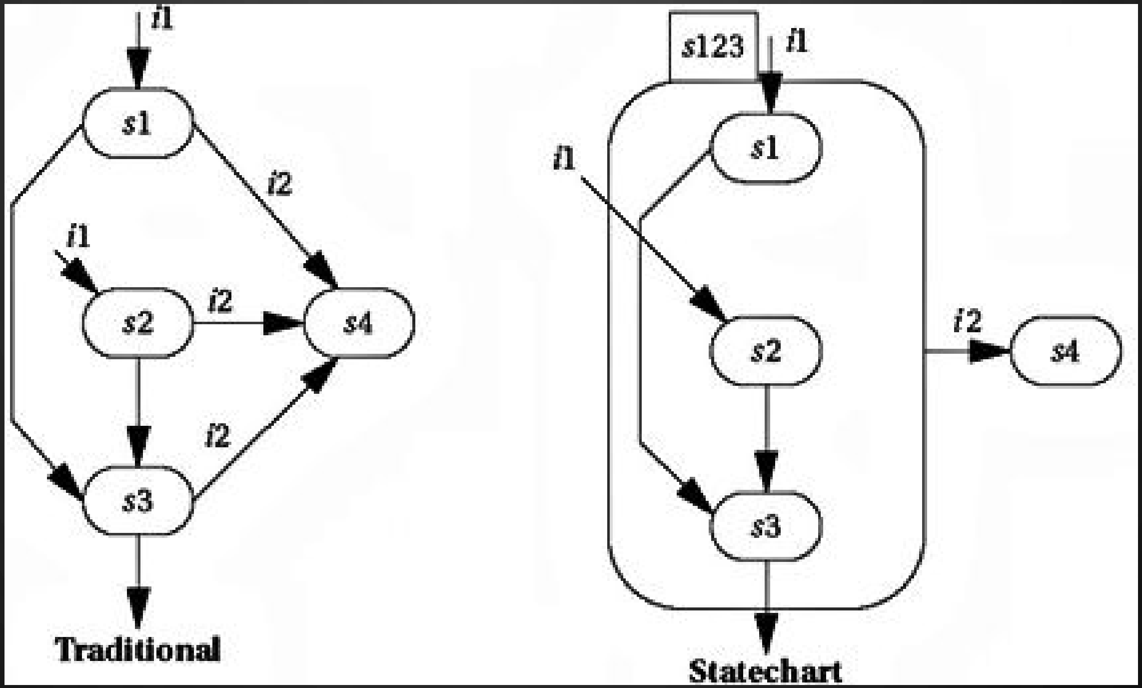


EXAMPLE OF OR



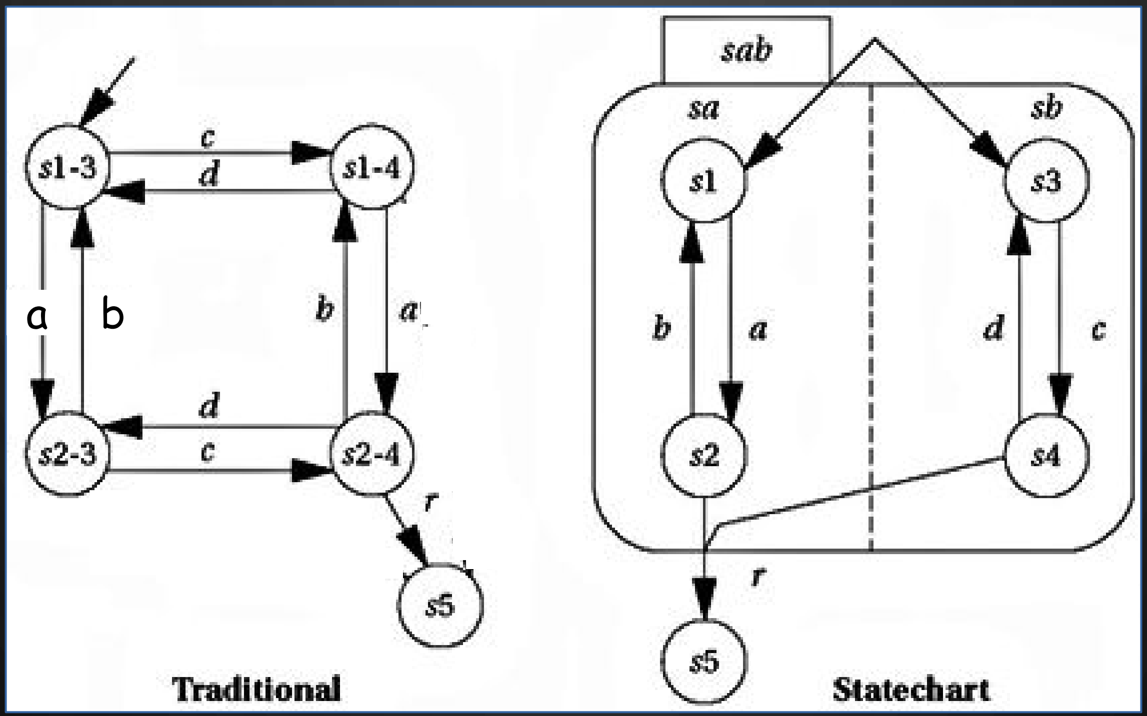
- The machine goes to s4 from s1, s2, s3
- Statechart captures this by having one state around s1, s2, and s3

EXAMPLE OF OR



- A single transition out of s123 means that on event i2, all states go to s4.
- The OR state allows transitions between its own internal states

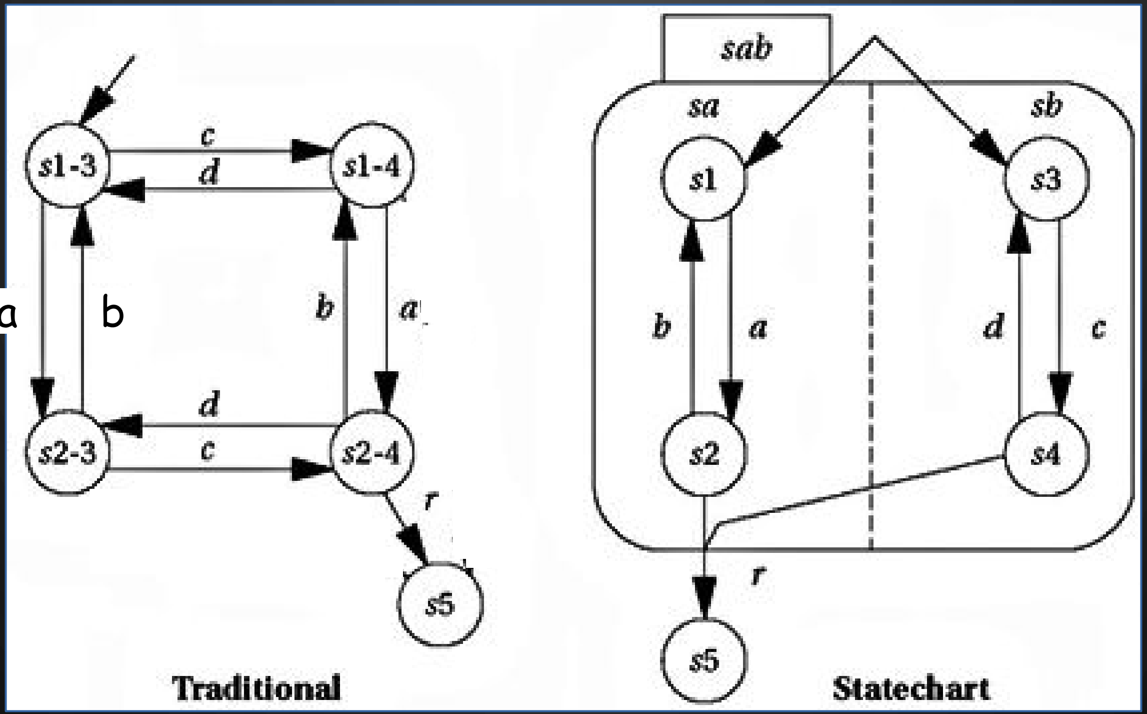
EXAMPLE OF AND GROUP



The traditional model has many transitions

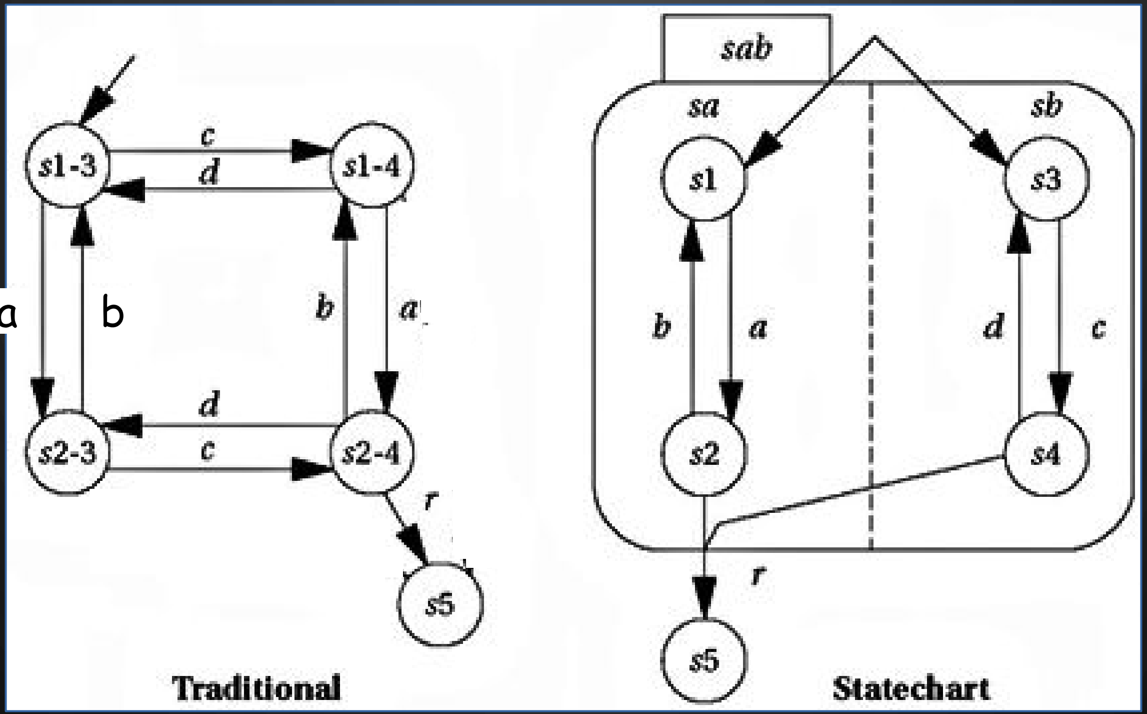
- Between states
- Going out of all states
- One initial transition

EXAMPLE OF AND



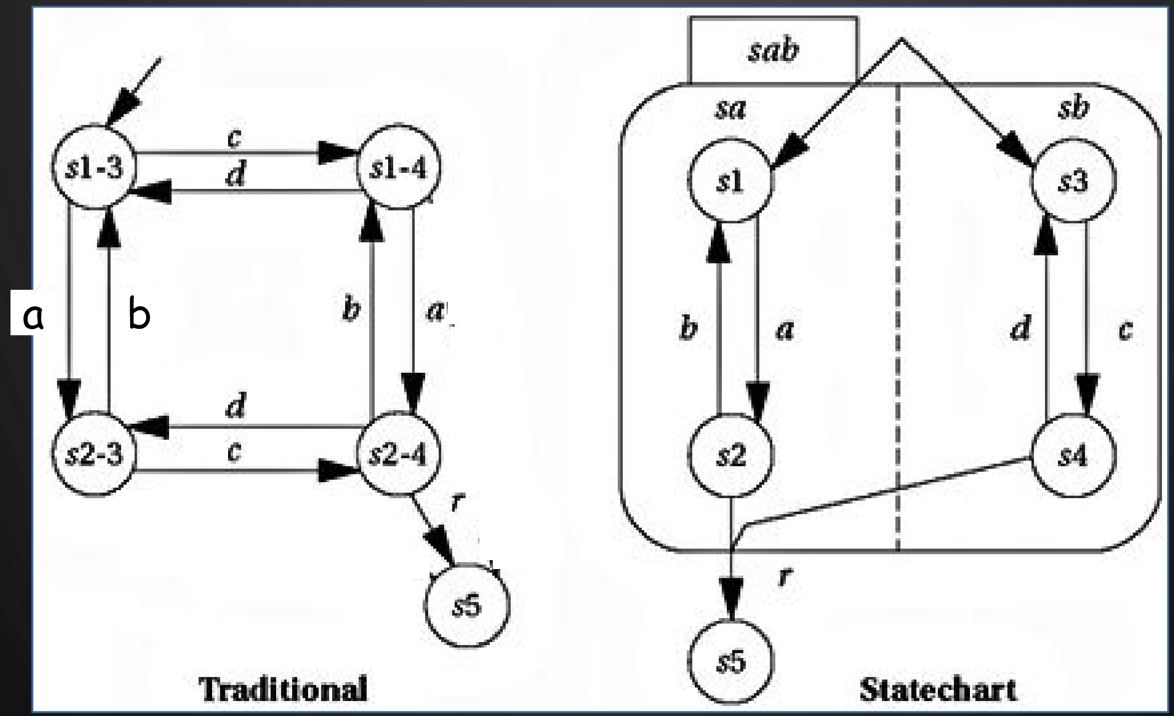
- The statechart has an AND state called sab
- sab has two components sa and sb
- When the machine enters the state sab, it is simultaneously in both sa and sb
  - We must know both sa and sb to know sab

EXAMPLE OF AND



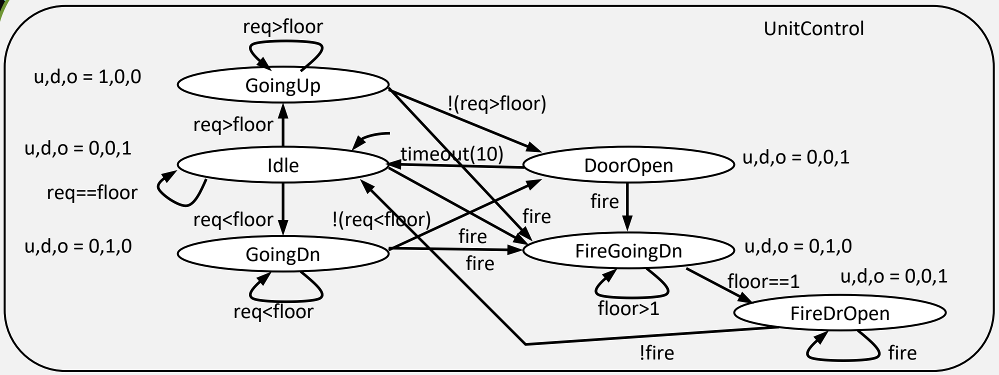
- The name of states reveal their relations
- s1-3 corresponds to sab in s1 and in s3
- When the machine enters the state sab, it is simultaneously in both sa and sb

EXAMPLE OF AND



- Both models describe the same behavior but the statechart is much simpler, cleaner and easier to understand

EXAMPLE OF HCFSM MODELING

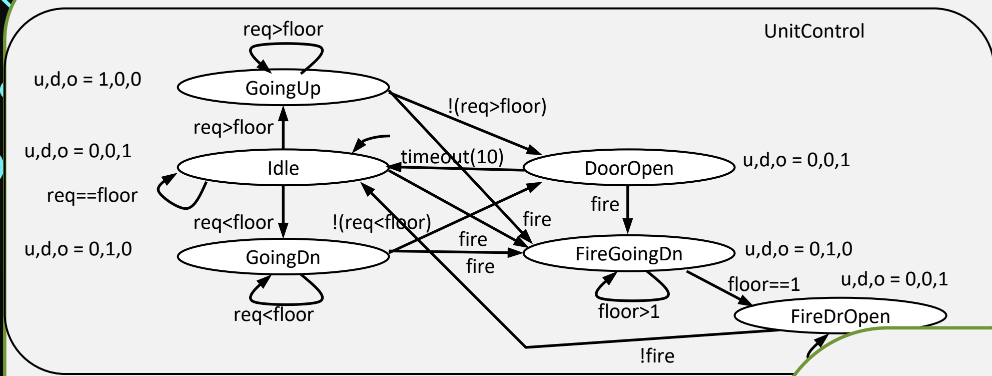


Without hierarchy

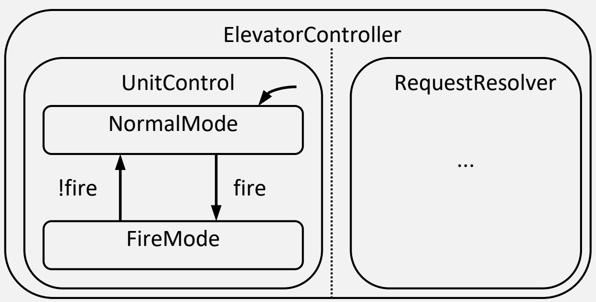
- FireMode
  - When *fire* is true, move elevator to 1st floor and open door
    - w/o hierarchy: Getting messy!
    - w/ hierarchy: Simple!

EXAMPLE OF HCFSM MODELING

- FireMode
  - When *fire* is true, move elevator to 1st floor and open door
    - w/o hierarchy: Getting messy!
    - w/ hierarchy: Simple!

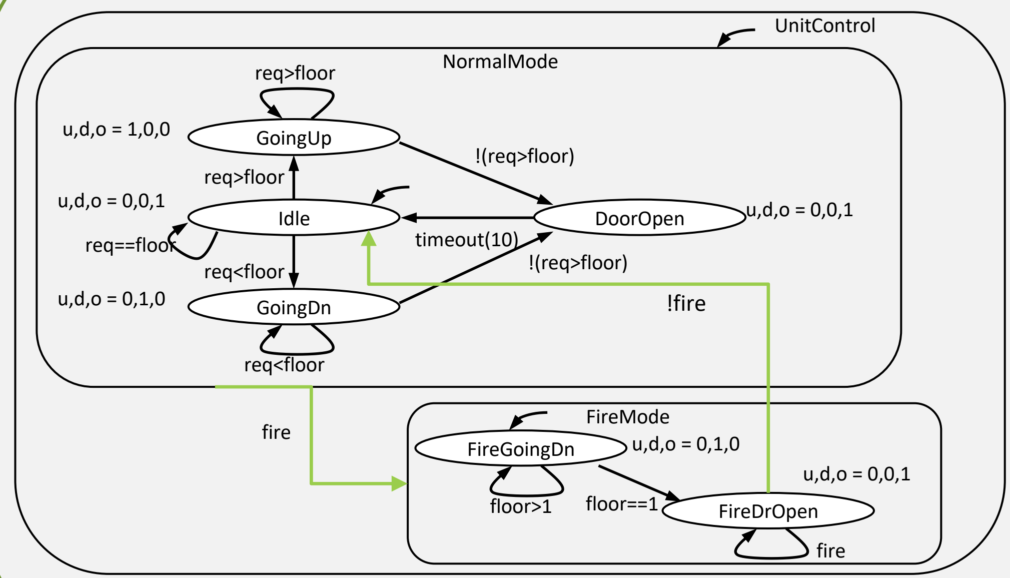


Without hierarchy

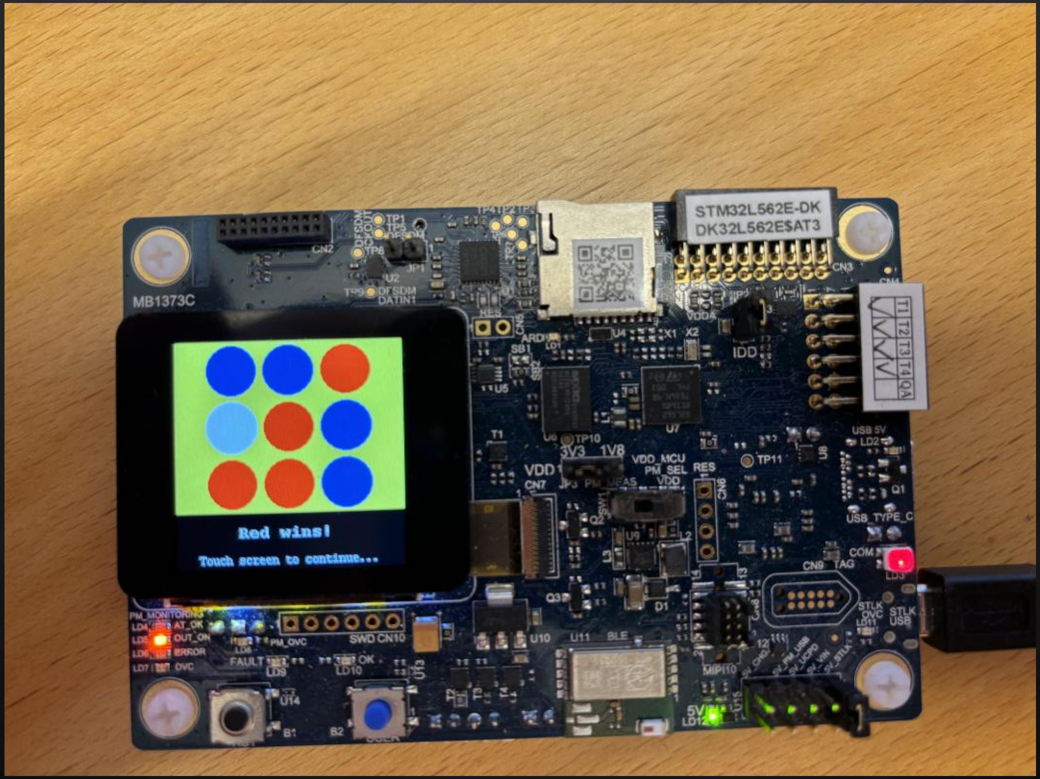


With concurrent *RequestResolver*

With hierarchy



FSM LAB ON STM32L562



STATE MACHINE FOR THE GAME

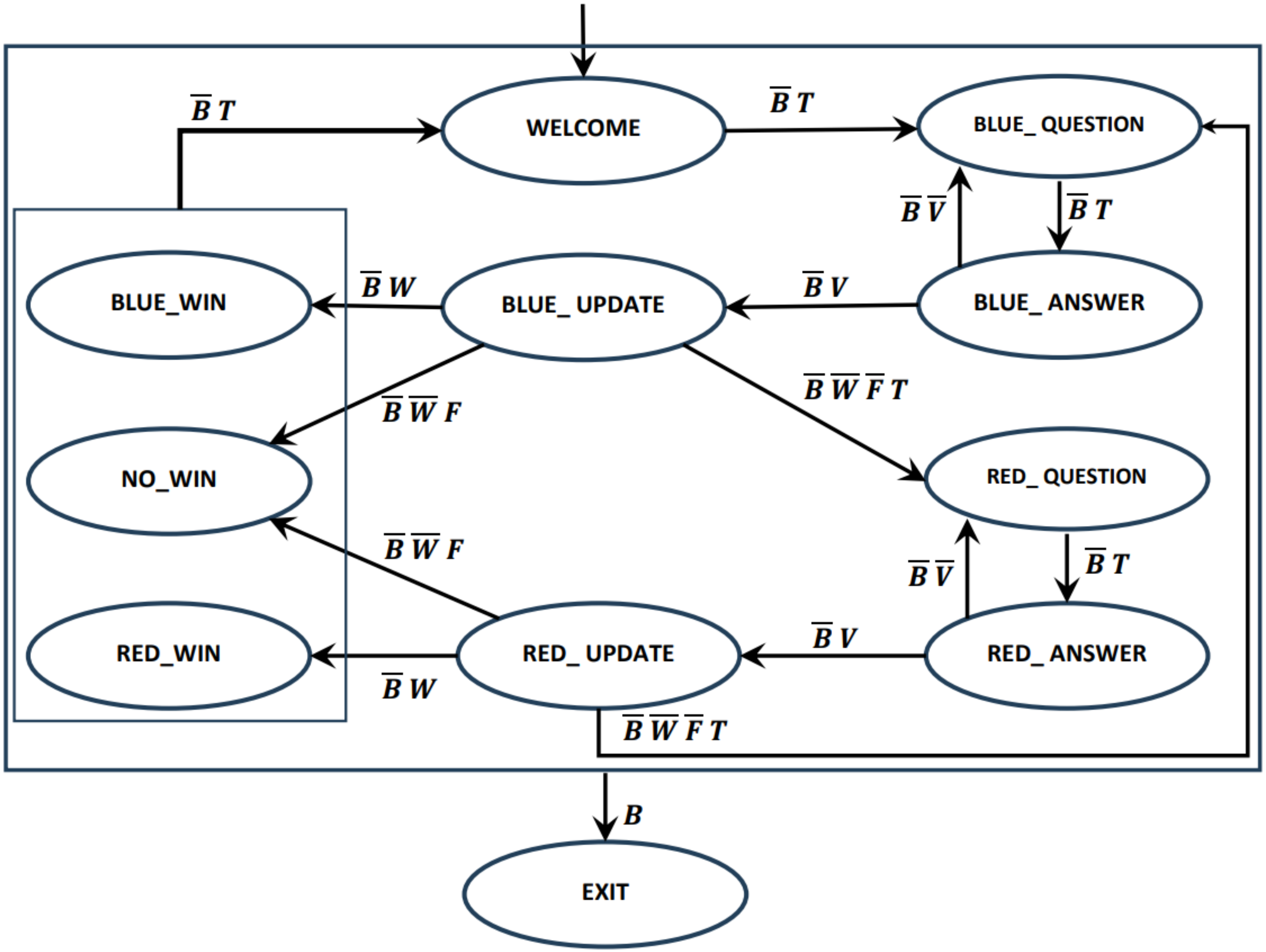


Figure 2 State Machine for TicTacToe. B stands for (pressed) Button, V for Valid (move), T for Touch (screen detected), W for Win, F for Full. Overlines correspond to negations.

STATE MACHINE FOR THE GAME

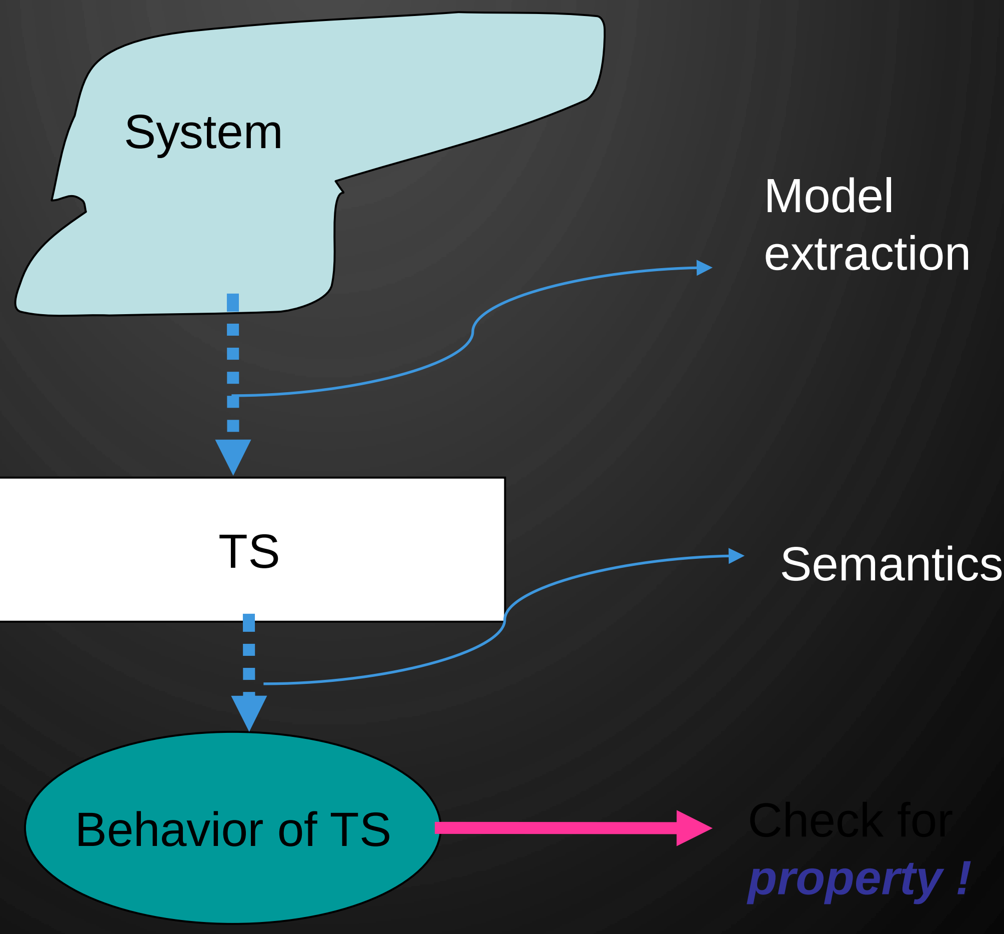
```
while (1){
     ...

    //Update State on entry
    if(State != PreviousState){
         switch(State){
              case WELCOME:
                Initialize_Board(BoardState);
                ...
                break;
              case BLUE_QUESTION:
                Touch = 0;
                ...
                break;
              case BLUE_ANSWER:
                Valid = Check_Move_Validity(…);
                break;
                ...
         } //update state
    }//on entry
```

```
    // transitions
    switch(State){
         case WELCOME:
              PreviousState = WELCOME;
              if(Button){
                   State = EXIT;
              }else if(Touch){
                   State = BLUE_QUESTION;
              }
              break;
         case BLUE_QUESTION:
              PreviousState = BLUE_QUESTION;
              if(Button){
                   State = EXIT;
              }else if(Touch){
                   State = BLUE_ANSWER;
              }
              break;
              ...
         }// transitions
```

VERIFICATION

THE VERIFICATION SETTING



TRAIN GATE EXAMPLE

- On the model checker UPPAAL
- You can download it from http://uppaal.org/
- Several demos of timed systems, the train gate controller is one of them
- You can do simulation and verification