


**DA-IICT**



## IT 314: Software Engineering

*State machine/State diagram*

Saurabh Tiwari

1

## State Machine Diagrams

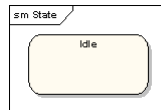
- A state machine diagram models the behavior of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events. As an example, the following state machine diagram shows the states that a door goes through during its lifetime.
- The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition `doorWay->isEmpty` is fulfilled. The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.

```

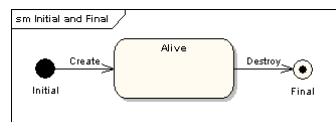
stateDiagram-v2
    [*] --> Opened : Create/
    Opened --> Closed : Close/ [doorWay->isEmpty]
    Closed --> Opened : Open/
    Closed --> Locked : Lock/
    Locked --> Closed : Unlock/
  
```

## State Machine Diagrams

- **States** - A state is denoted by a round-cornered rectangle with the name of the state written inside it.

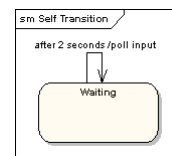
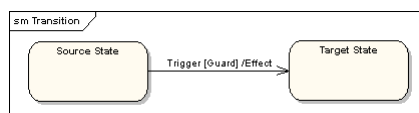


- **Initial and Final States** - The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.



## State Machine Diagrams

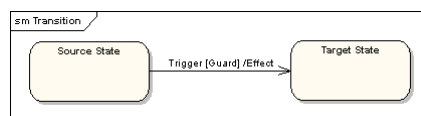
- **Transitions** - Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.
- **Self-Transitions** - A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.



- **"Trigger"** is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. **"Guard"** is a condition which must be true in order for the trigger to cause the transition. **"Effect"** is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

## State Machine Diagrams

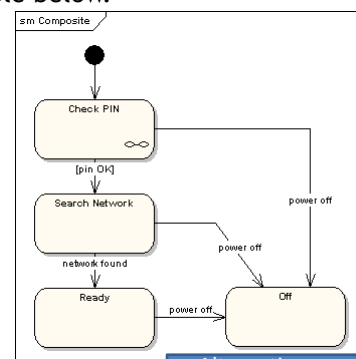
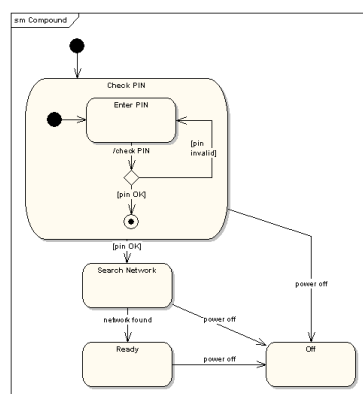
- **State Actions** - In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.



- It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.

## State Machine Diagrams

- **Compound States** - A state machine diagram may include sub-machine diagrams, as in the example below.

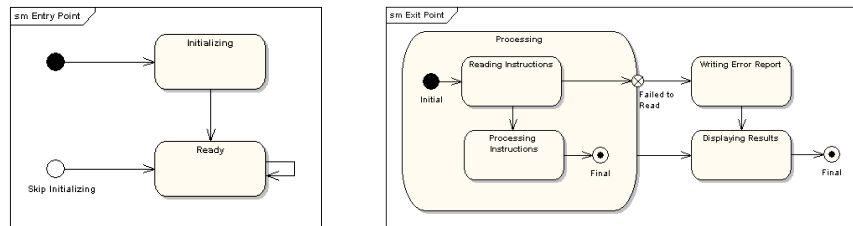


Alternative way to show the same information

• The ∞ symbol indicates that details of the Check PIN sub-machine are shown in a separate diagram.

## State Machine Diagrams

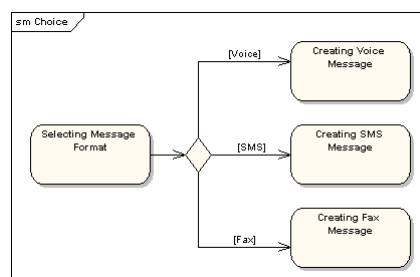
- **Entry Point** - Sometimes you won't want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.



- **Exit Point** - In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.

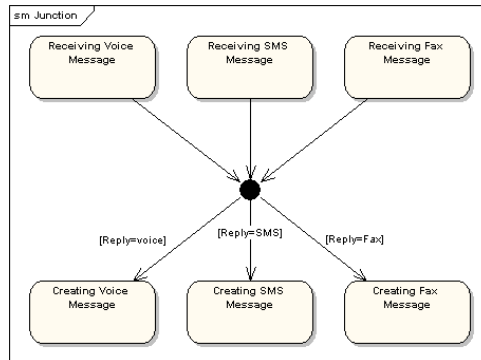
## State Machine Diagrams

- **Choice Pseudo-State** - A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



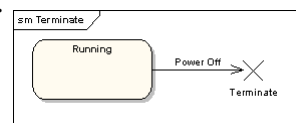
## State Machine Diagrams

- **Junction Pseudo-State** - Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition.
- A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.

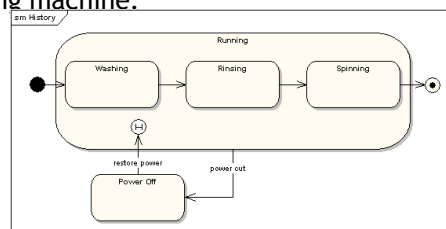


## State Machine Diagrams

- **Terminate Pseudo-State** - Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.

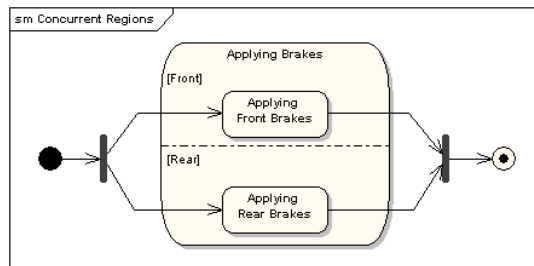


- **History States** - A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.



## State Machine Diagrams

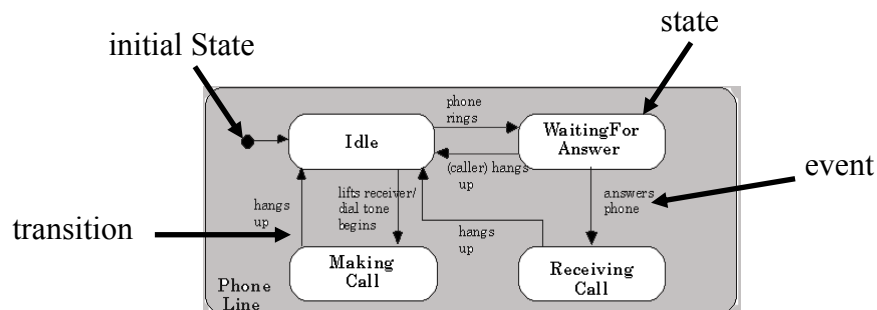
- **Concurrent Regions** - A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



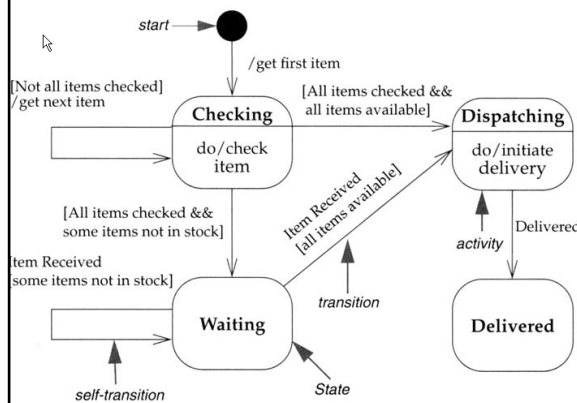
## State Machine Diagrams

A State chart diagram shows the lifecycle of an object

- A *state* is a condition of an object for a particular time
- An *event* causes a *transition* from one state to another state
- Here is a State chart for a Phone Line object:



## State Machine Diagrams



- Transitions labels have three optional parts:

**Event [Guard] / Action**

- Find one of each
- **Item Received** is an event, **/get first item** is an action, **[Not all items checked]** is a guard

- State may also label activities, e.g., **do/check item**

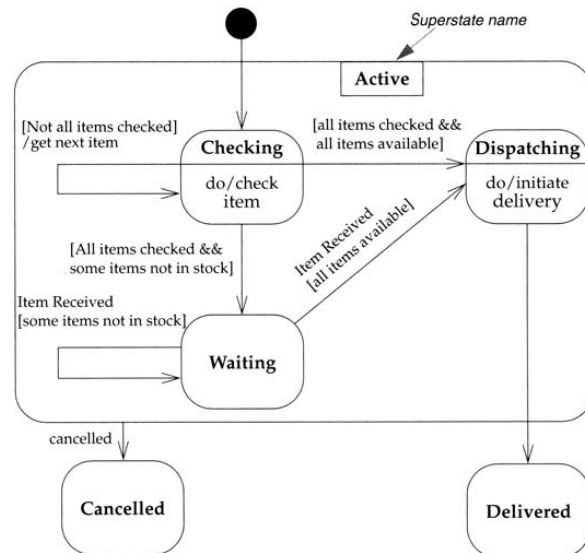
- **Actions**, associated with transitions, occur quickly and aren't interruptible
- **Activities**, associated with states, can take longer and are interruptible
- Definition of "quickly" depends on the kind of system, e.g., real-time vs. info system

## When to develop a state chart?

Model objects that have change state in interesting ways:

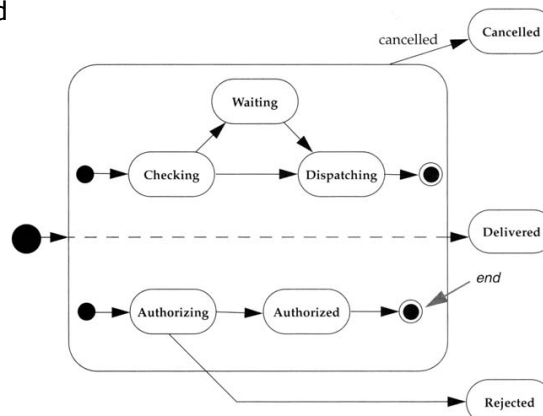
- Devices (microwave oven, Ipod)
- Complex user interfaces (e.g., menus)
- Transactions (databases, banks, etc.)
- Stateful sessions (server-side objects)
- Controllers for other objects
- Role mutators (what role is an object playing?)
- Etc.

## Superstates (nested states)



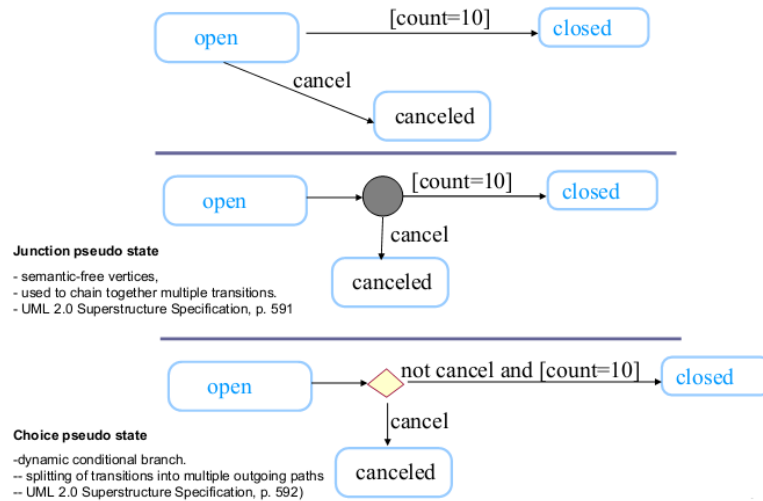
## Concurrency in state diagrams

- Dashed line indicates that an order is in two different states, e.g. Checking & Authorizing
- When order leaves concurrent states, it's in a single state: Canceled, Delivered or Rejected



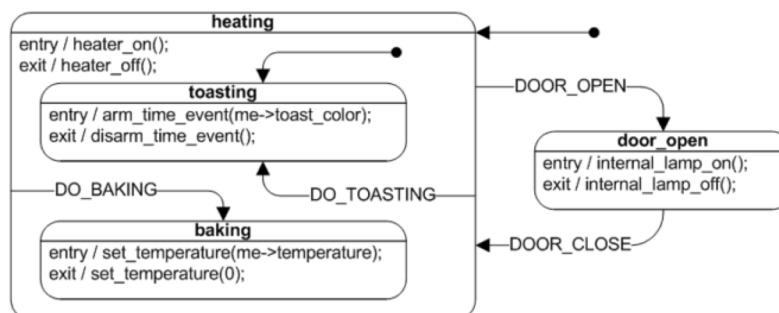


## State Transitions – notational variation



17

## State notations

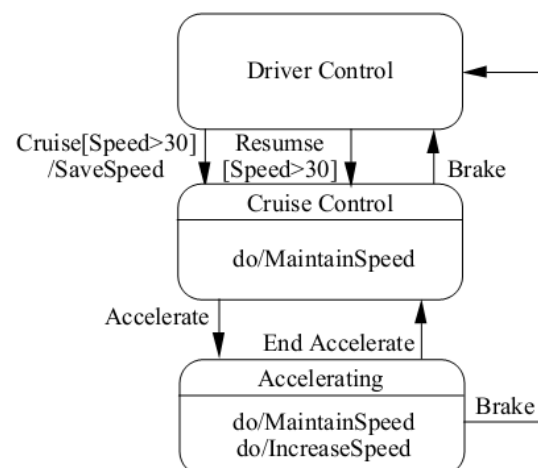


18

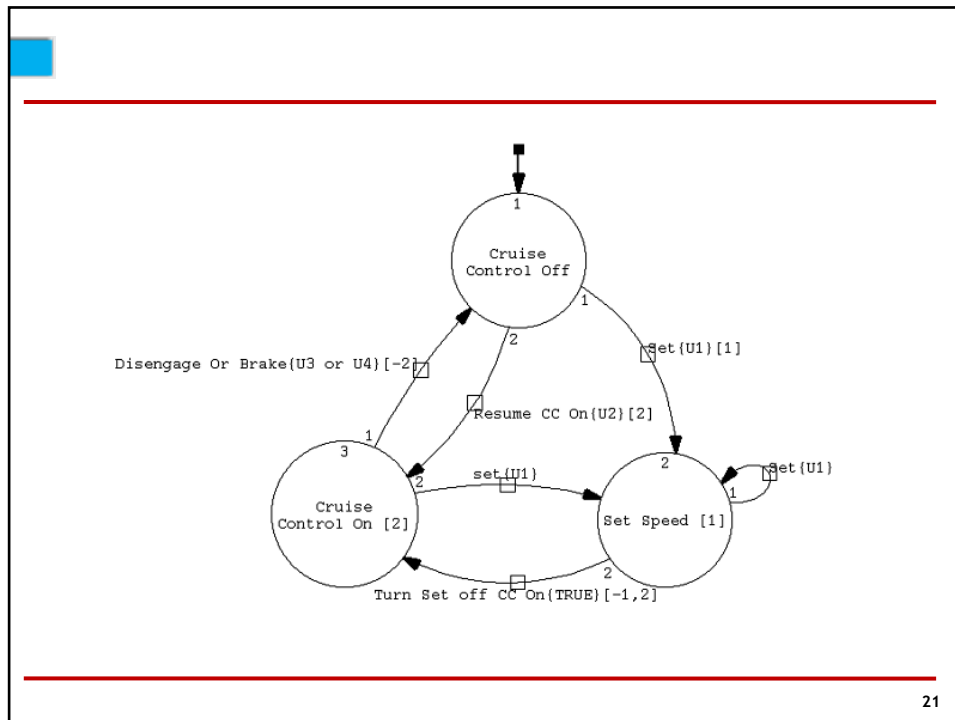
## Exercise- Cruise Control System



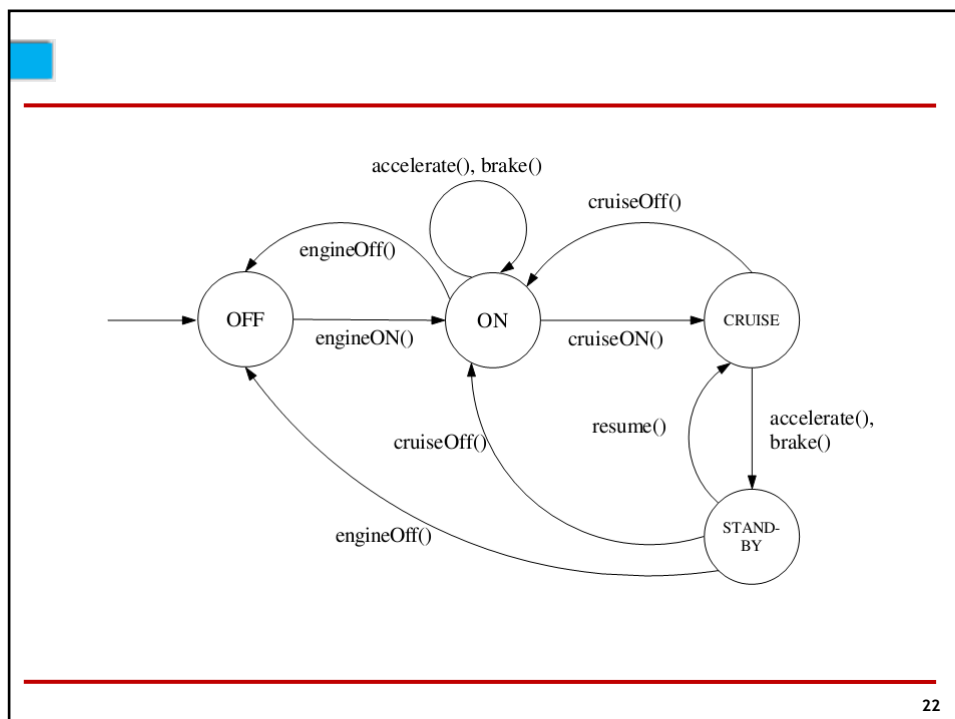
19



20



21



22

## Cruise Control System Specifications

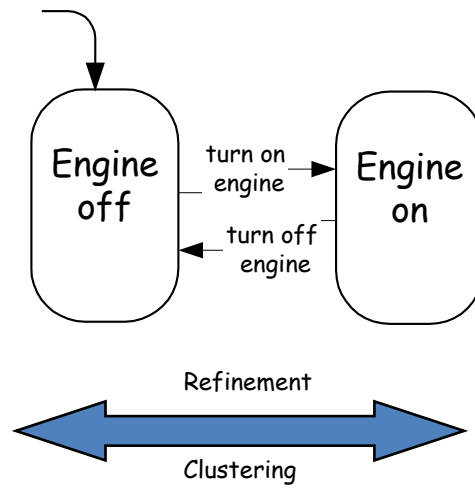
1. You can assume an automatic transmission vehicle.
2. For any of the cruise control (CC) functions to take effect, CC must be turned on first.
3. CC can be in the following states: off, enabled (i.e., on and cruising), and disabled (on, but not cruising).
4. The CC system should be automatically disabled below 30mph and above 90mph.
5. Four actions are permitted during CC: set speed, accelerate, decelerate, and resume speed.
6. When the system is under CC and the brake is pressed, CC is disabled. When the resume button is pressed, the system resumes at the last set CC speed.
7. When the system is under CC and the accelerator pedal is pressed, CC is disabled and the speed increases correspondingly. When the accelerator is released, the CC resumes at its last set CC speed. If at any point of time during acceleration the CC speed is set, CC replaces the old set speed with the new speed.
8. If CC is enabled and the vehicle starts going uphill or downhill, CC should automatically apply the accelerator or brake to maintain the set speed.

## Cruise Control (CC) Events

- Engine on
- Engine off
- CC off
- CC on (+ cruising+ disabled)
- Set speed (CC is on)
- Accelerate
- Decelerate
- Resume CC

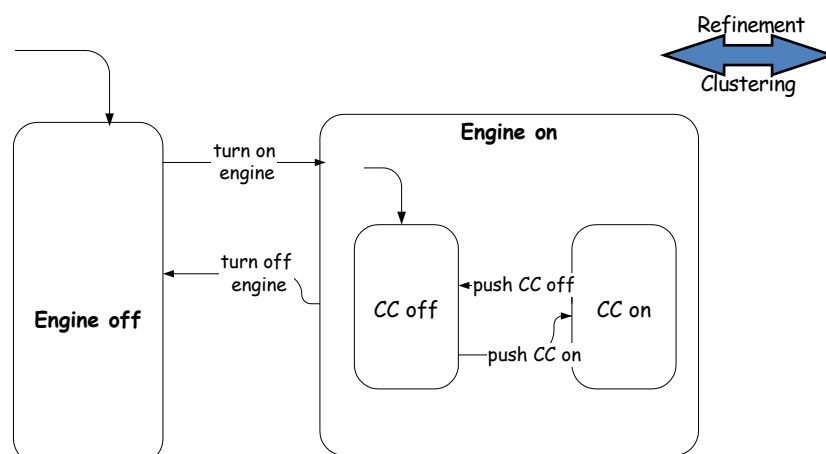
- Top level states: Engine on and engine off

- Events to change states: turn on engine, turn off engine

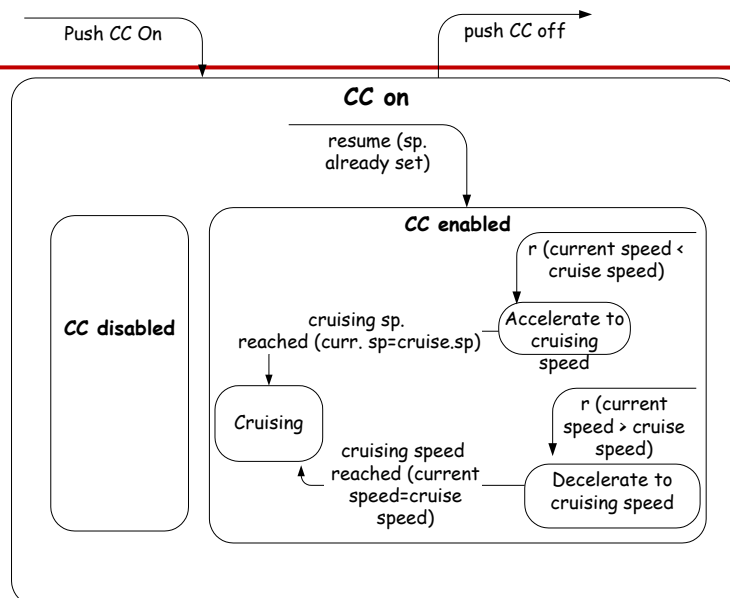
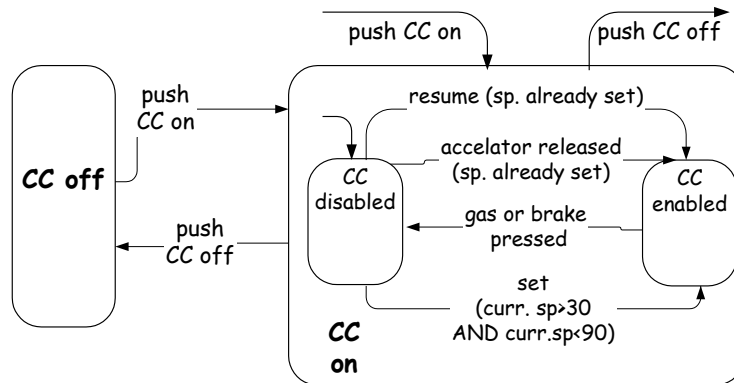
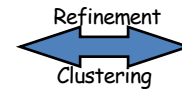


- Zoom in engine on: CC on and CC off

- Events to change states: push CC on, push CC off



- Zoom in CC on: CC enabled and CC disabled
- Disable to enable: push resume (speed set);  
push set ( $30 < \text{speed} < 90$ );  
accelerator released;
- Enable to Disable: gas or brake pressed;



## Generating Objects' Statechart

```

Context CoinBox {
  int curQtr, quantity, totalQtrs
  boolean allowVend

  inv : int curQtr, quantity, totalQtrs >= 0

  :: CoinBox()
    post : self.curQtr = 0
         self.allowVend = FALSE
         self.quantity = 0
         self.totalQtrs = 0

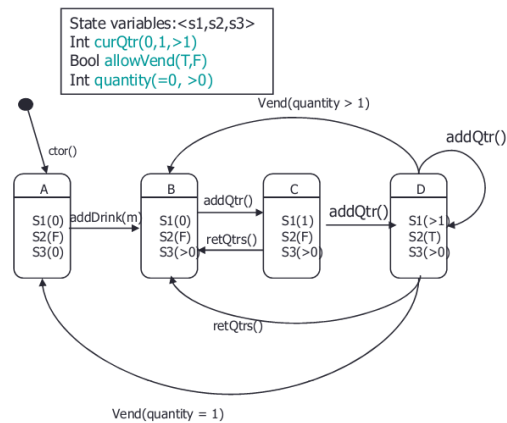
  :: addQtr():void // add a quarter in the machine
    pre : self.quantity > 0;
    post : self.curQtr = curQtr@pre + 1
          if (self.curQtr@pre = 1) then
            self.allowVend = TRUE

  :: retQtrs():void // return quarters back to the user
    pre : self.curQtr > 0;
    post : self.curQtr = 0
          self.allowVend = FALSE

  :: vend():void // deliver a drink
    pre : self.allowVend == TRUE and
          self.quantity > 0;
    post : self.curQtr = 0
          self.allowVend = FALSE
          self.quantity = quantity@pre - 1
          self.totalQtrs = totalQtrs@pre + curQtr@pre

  :: addDrink(m: int):void // add m unit of drink in
                           // the machine
    pre : self.quantity == 0 and m > 0;
    post : self.quantity = quantity@pre + m
}

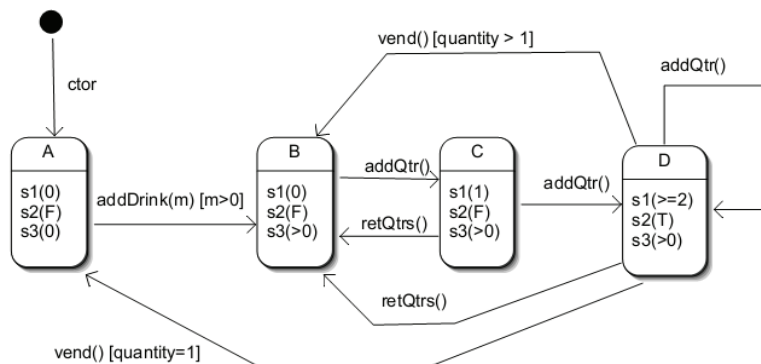
```



Statechart for the COINBOX class

Specification in OCL

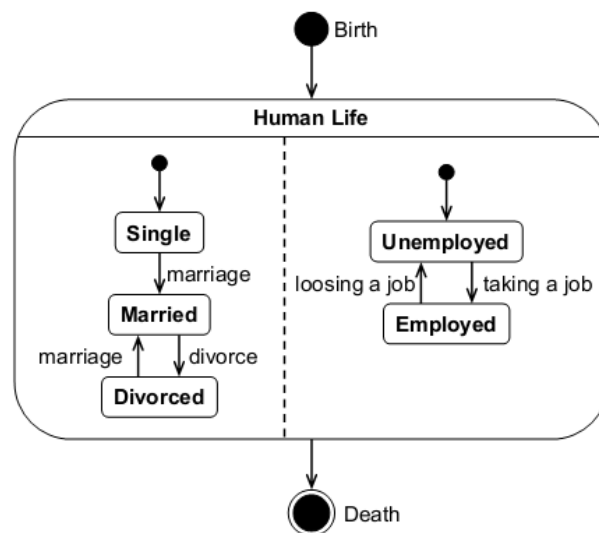
## Generating Objects' Statechart



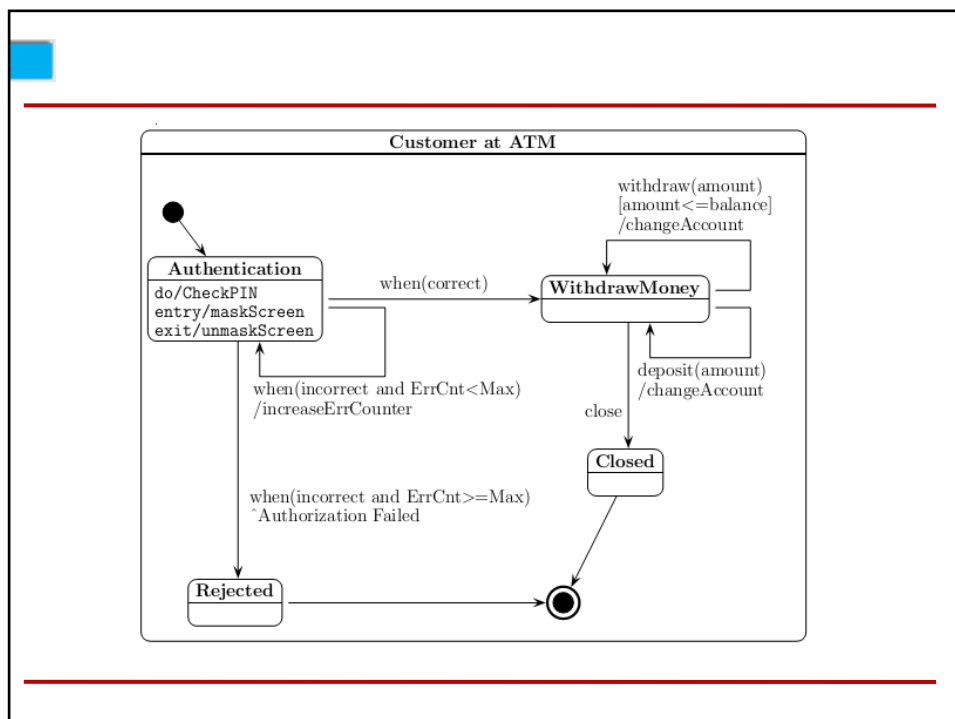
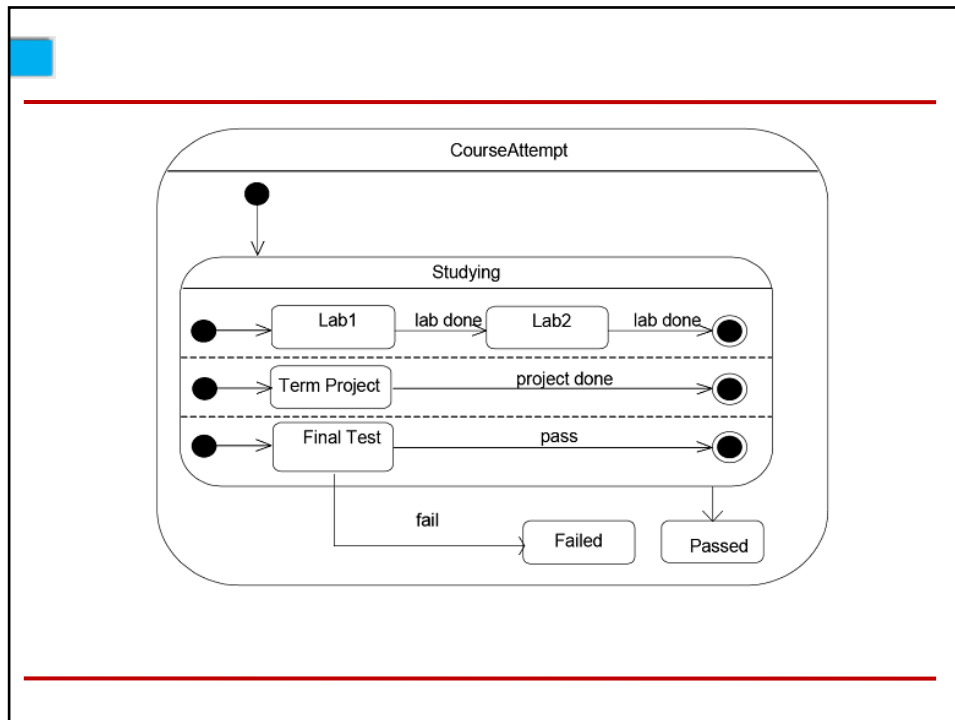
State Variables: <S1, S2, S3>  
 S1- int curQtr := {=0, =1, >=2},  
 S2- boolean allowVend := {T,F}  
 S3- int quantity := {=0, >0}

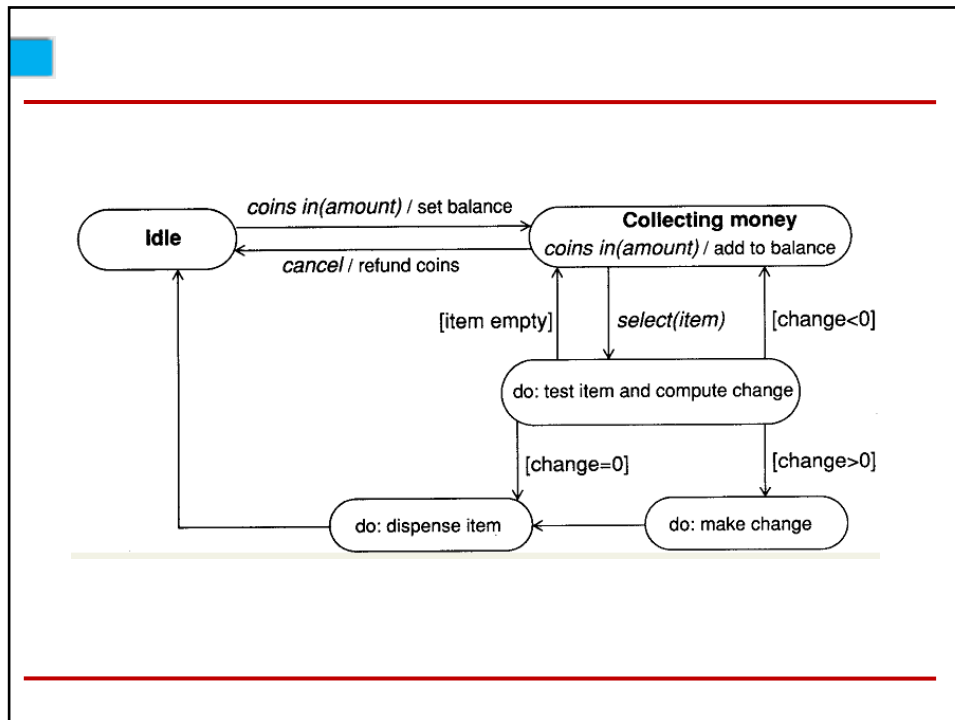
Questions??

Next...  
State and State variables...









## Process of State Machine Modeling

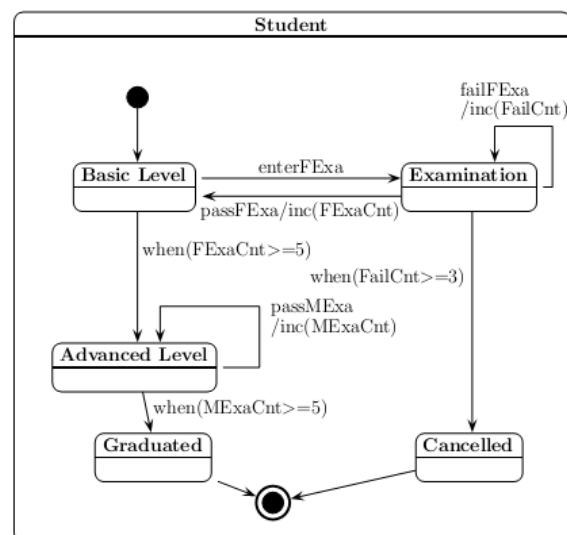
1. Identify possible states.
2. Identify external events which can cause changes of states.
3. To each state attach transitions based on relevant events.
4. Add internal actions and transitions.
5. Identify composite states and specify contents of regions.
6. Repeat until the state machine is complete.
7. Review the state machine by simulation of the **owner's** lifecycle. Based on simulated events, check the appropriate triggering of transitions and execution of behavior.

## Exercise?

A student must complete the basic level before entering the advanced level. After both levels, the student has to pass five examinations.

An examination can be retaken at most twice. After the third failed attempt the student's registration is cancelled.

## Solution?





## Exercise? We will Solve in Lab!!

---

- Give a state diagram that describes the process of passing a graduate course as a set of concurrent activities. The process has as follows: To pass, a student has to attend all but two lectures, present to the class a paper she read, and complete a course project, due on the last day of the term. To give her presentation, the student is given a date by the instructor, prepares her presentation, and gives it on the assigned day. At any time, the student can drop the course.
  - Make sure to define events, conditions, actions for transitions in your diagram, where appropriate.
- 




## Criticism?

---

Not really a good model, because . . .

- the student leaves the basic level to take an exam
  - the student can cheat by repeating a passed exam
  - the student cannot enter parallel exams
  - the student has to complete each exam once tried
  - the student cannot pass exams of the advanced level
  - while in the basic level
-

 **Sample Exercises?**

---

Seven Problem Exercises...

*Next Lecture...*  
*Relation of Class and State Models...*  
*More on State variables...*  
*Sequence to State Models...*

---