

CS6133 Fall 2011 Software Specification and Verification

Lecture 4 Finite State Machine

Finite State Machine

- ◆ Finite state machine (FSM) was designed for expressing the behavior of computer-based systems
- ◆ A finite state machine is a four tuple, $\langle S, S^I, V, T \rangle$, where
 - S is a finite set of control states
 - S^I is a finite set of initial control states
 - V is a finite set of variables
 - T is a finite set of transitions

CS6133

2

Elements of FSM

- ◆ Control state represents progress in the execution of the system by indicating what to execute next
- ◆ System state is an assignment of values to the system variables.
 - System state captures some information of the system at a certain moment in the execution
- ◆ Transition transforms a system from one state into another state
- ◆ Initial states indicate the starting points of execution

CS6133

3

FSM-Based Notations

- ◆ Statecharts
 - Harel's statecharts
 - RSML
 - UML state machines
- ◆ Labeled transition systems (LTS)
- ◆ Hierarchical transition systems (HTS)

CS6133

4

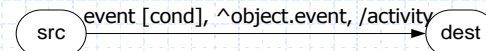
UML State Machine

- ◆ Shows how an object's behavior changes over time
 - How to react to events
 - How to react to messages received, and shows when it outputs messages to other objects
- ◆ State partitions class behavior
- ◆ Transition describes a change in state due to the occurrence of an event

CS6133

5

Transition



- event(args): the event or message that triggers the transition
- [condition]: boolean value; the transition cannot fire unless the guard condition is "true"
- /activity: simple, fast, non-interruptible action
e.g., variable assignment,
- ^object.event(args): send a message to an object
- ◆ Each of these parts is optional

CS6133

6

Event

- ◆ An event is:
 - Calls or signals an object can detect
 - External: a change in the environment
e.g., "off-hook"
 - Internal: a message from another object (operation call)
 - Change in a condition: external or internal
e.g., when(temperature > 100 degrees)
 - Temporal: the occurrence of a specific date/time or passage of time
e.g., after (20 minutes)

CS6133

7

Events

- ◆ Events make an object change state
- ◆ If the object is in a state and there is not outgoing transition triggered by an event, the event is ignored

CS6133

8

State Activities

- ◆ A state can have activities associated with it
- ◆ State activities can manipulate object attributes or other variables
 - Activity: simple, non-interruptible
 - Associated with a transition or a state
 - Performed on state entry or exit
 - Do activity: interruptible; may require much computation
 - Associated with a state
 - Interrupted by a transition

CS6133

9

State Activities

- ◆ States can be annotated with entry or exit activities, internal activities, and activities:
 - ◆ entry/activity
 - ◆ event/activity (way to describe reactions to events that don't cause a state change -- internal transitions)
 - ◆ exit/activity
 - ◆ do/activity

CS6133

10

State Activities

- ◆ In a transition, the order of effects:
 - Exit activities of source
 - Transition activities
 - Entry activities of destination, and
 - State activities

CS6133

11

Self-Transition

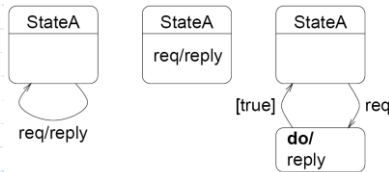
- ◆ A self-transition will cause reactivation of exit and then entry events
- ◆ If you want a self-transition that does not activate these events, you can use an internal transition, labeled with the event and the associated activity
e.g., "req/reply" in state "S"

CS6133

12

Self-Transition

- ◆ Compared to an activity on a transition, one could achieve the same behavior
 - As activity in state
 - as activity in another, special state



CS6133

13

More on Transition

- ◆ If a transition has no event label, it can occur once any activity associated with the state is complete
- ◆ Guards on transitions (triggered by the same event) leaving a state should be mutually exclusive

CS6133

14

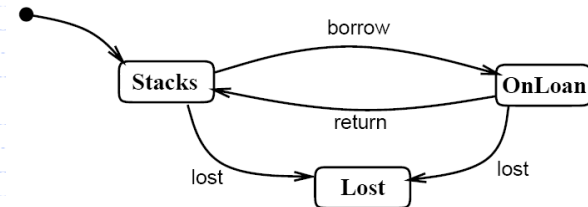
State Hierarchy

- ◆ Super-state: a state can contain other states
- ◆ Super-state combines states and transitions that work together towards a common goal
 - ◆ If a transition leaves a super-state, this transition applies to all the sub-states and has higher priority than the transitions within the super-state
 - ◆ The sub-states "inherit" the transitions of the super-state
 - ◆ Basic state: a state do not contain other states

CS6133

15

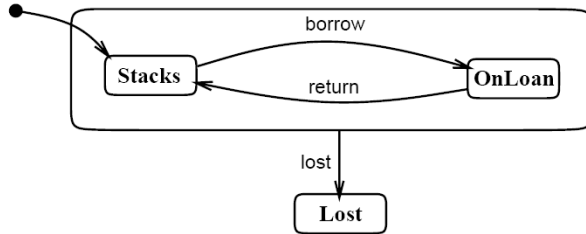
State Diagram Without Super-State



CS6133

16

State Diagram with Super-State



CS6133

17

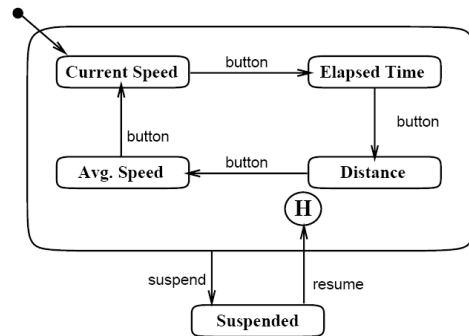
History State

- ◆ History is a mechanism by which a re-entered super-state can continue executing from the sub-state that was current when control last transitioned out of the super-state
- ◆ Deep history
- ◆ Shallow history

CS6133

18

History State - An Example



CS6133

19

Concurrent State

- ◆ A concurrent state captures two or more behaviors of the object that happen concurrently each with its own control thread
- ◆ If there are many concurrent states for one object, you should consider dividing the behavior between more objects

CS6133

20

Composite State

- ◆ Nonorthogonal composite state
 - Super-state contains no concurrent states
 - System (object) is in exactly one basic state
- ◆ Orthogonal composite state
 - Concurrent state contains two or more orthogonal regions, which have their own threads of control respectively
 - System (object) is in every region concurrently

CS6133

21

Types of Orthogonal States

- ◆ Aggregation concurrency
 - State diagram for an aggregate object is a collection of state diagrams, each of which corresponds to a constituent object
- ◆ Synchronization of concurrent activities within one object
 - Object performs two or more activities concurrently
 - Object completes all concurrent activities before it progress to another state

CS6133

22

Semantics of UML State Machine

- ◆ An object executes by performing run-to-completion steps
 1. An event is removed from the object's event pool for the object to process
 2. A maximal set of non-conflicting enabled transitions are executed
 3. The events generated by these transitions are sent to the targeted objects
 4. Step 2 and step 3 are repeated, until no more transitions are enabled

CS6133

23

UML State Machine Case Study

The heating system consists of a furnace, a controller, and a room to be heated. The room has a valve that controls airflow into the room; the valve can be open, half open, or closed. The room also has a sensor that measures the room's temperature and a thermostat by which a user can set the desired temperature. If the room temperature is lower than desired, the system warms the room by opening the valve, to increase the inflow of heated air; if the room continues to be too cold, the room requests heat. The system behaves analogously when the room temperature is too hot. The controller activates and deactivates the furnace on request from the room.

CS6133

24