Statecharts
Abhik Roychoudhury
School of Computing
National University of Singapore

Warm up – the big picture

System to be built (Dream)
System Model (Rough Idea)
Properties to Satisfy (caution)
Checking Method (Automated)
Violations
Refine the model

Background
- Finite state machines
  - Other variants
  - Model Reactive and transformational systems
- Statecharts is one of the simplest and most popular modeling formalism
  - Very intuitive, visual.
  - An illustration of how to model systems with statecharts will be shown via Rhapsody tool.
    - Also, tested in the first lab assignment.

Readings
- Executable object modeling with statecharts, by David Harel and Eran Gery, IEEE Computer, 1997
- Basic understanding of states/transition is introduced first.

Introducing FSMs --- a puzzle
- A man with a goat, a wolf and a cabbage wants to cross a river.
- A boat can carry only 2 of the 4 entities.
- Wolf wants to eat the goat.
- Goat wants to eat the cabbage.
  - How to transport all the 4 entities?
- Think of modeling the local state of each entity – on which side of the river?
  - A global state is a composition of these local states --- transitions of global states form FSM

State change
Modeling using FSMs

- A solution to our problem is a path from the initial state to a state where all 4 entities are on other side of river.
  - Notion of “termination” of the problem.
  - Shown as accepting states of FSMs

- Minor note:
  - Not all cycles in the FSM for this problem have been shown.

FSM --- Definition

- \( M = (S, S_0, \Sigma, \rightarrow, F) \)
  - \( S \) is a set of states
  - \( S_0 \subseteq S \) is the set of initial states
  - \( \rightarrow \subseteq S \times \Sigma \times S \) is the transition relation
  - \( F \subseteq S \) is the set of final or accepting states

- The set of strings accepted by \( M \) or the language of \( M \)
  - \( L(M) = \) all strings which have a path from an initial state to an accepting state.
  - Using finite state machines for recognizing or distinguishing (infinite) set of (finite) strings.

FSM --- Example

- Accepts all binary strings with odd number of 1s
  - An infinite collection of finite strings

Transition Systems

- FSMs can accept infinite strings too, change accepting condition
  - An infinite string is accepted if it visits at least one final state infinitely often.

- Transition systems go one step further where all states are accepting.
  - \( TS = (S, S_0, \Sigma, \rightarrow) \)
    - No notion of terminating or accepting states
    - The alphabet \( \Sigma \) labeling the transitions is also optional.
    - The traces captured by a transition system are obtained by unrolling the graph from the initial state(s).

TS - Example

- Traces captured by this transition system are
  - \( (0, 1)^* 0^* \)
  - \( (01)^* \)
Transformational Systems

- Conventional notion of a terminating program.
  - Takes in input.
  - Performs computation step.
  - Terminates after producing output.
- System behavior
  - Can be described as a transformation function over the input.
- What about controllers?
  - In continuous interaction with the environment.

Reactive Systems

- Continuously interacts with its environment.
  - No notion of system termination.
- Interaction with environment is typically asynchronous.
- Often consists of a concurrent composition of processes.
  - Often, its response to environment needs to obey time constraints.

Reactive system behavior

- (Infinite) collection of infinite traces.
- Traces denote ongoing interaction with environment.
- Use state transition systems to describe behavior of a reactive system
  - Too much complexity
  - Many processes ↔ concurrency
  - Each process has many states ↔ hierarchy
  - What kind of inter-process communication?
- The language of Statecharts addresses these practical issues!!

Visual Formalisms

- Important/imperative at initial design stages.
- Vital for communication.
- Formal visual languages can help in:
  - Documentation
  - Initial analysis.
  - Developing correct-by-construction translation to more detailed (non-visual) descriptions.

Statecharts

- Statecharts =
  - FSMs +
  - Depth +
  - Orthogonality +
  - Structured transitions +
  - Broadcast communication
- Used in the Rhapsody tool.
- Included in UML 2.0 as state diagrams.

General Idea

- Statecharts
  - = FSM + many features to contain complexity.
- What does the FSM denote?
  - System response to external triggers.
  - States of the FSM = internal states of the system
  - Transitions of the FSM are labeled by such triggers.
    - An external trigger may in turn generate internal triggers which can also form the labels.
  - Traces of the FSM are sequence of transitions.
    - System response should stabilize eventually, waits for the next external trigger from environment.
**Statecharts**

Depth:
- States can have internal structure.
- OR type states

Orthogonality
- Independent states
- Concurrency
- AND type states

Structured transitions
- Succinct descriptions of transition families.

Broadcast communication
- Succinct descriptions of synchronizations

---

**Depth: OR States**

(b) is the statechart representation of the FSM (a).

---

A and C are clustered into a superstate D
A and C are the internal exclusive-or components of the D state.

---

e, f: are trigger (external) events.
g [ c ]: g, a trigger event and c a condition

---

f is a transition from D to B.
From any D-state (A or C) there is an f-move to B

---

h is transition from B to D (A or C).
The actual state entered is the default entry state; the state C
D is the initial state.
The actual initial state within D is not the default state C.
Instead, it is A.

Which state will transition e yield in (b) and (c)?
Which state will transition h yield in (b) and (c)?
What's the default state for the superstate E in (c)? Hierarchically!

An OR-state can contain other states as its internal substates (hierarchical internal structure);
A super OR-state is active, if and only if one of its immediate substates is active (exclusive or);
When the control enters a (super) OR-state, its default substate is entered and becomes active;
When the control leaves a (super) OR-state, all its substates become inactive!
More issues: history, priority, ...

Y is an AND state.
It has two orthogonal components A and D.
A is an OR state with components B and C.
D is an OR state with components E, F and G.
Y is an AND state. It has two orthogonal components A and D. A state of Y is composed of a state of A and a state of D. What is the default initial state of Y?

f belongs to only A. e belongs to both A and D. From (B,F) there is a simultaneous e-move to (C,G)

f belongs to only A. e belongs to both A and D. From (B,F) there is a simultaneous e-move to (C,G)
Orthogonality: AND States

From every Y state (how many?) there is a p-move to I

From every Y state (6!) there is a p-move to I
From I there is an e-move to the Y-state (?, ?)

From I there is an e-move to the Y-state (C, G)
What if there is an e-arrow from I to just the surface of Y

For each (?, F) state there is an m-move to I
Note the [in G] condition attached to the f-move from C (state reference!).

AND-state: in a nutshell

- An AND-state is composed of several independent (OR-)states that run in parallel (concurrency);
- An active state of an AND-state comprises a state of each concurrent component, i.e., (s₁₁, s₂₂, ..., sₙₙ);
- When the control enters (leaves) an AND-state, it simultaneously enters (leaves) all its components;
- An AND-state can even occur inside an OR-state (different from conventional programming languages)

Broadcast Communication

A transition has a trigger and an action (output!) But the output of a transition can be inputs for other orthogonal components!
Suppose m (external event) occurs.

H goes to I from J; e-moves are enabled in A and D.

Suppose event n comes, what happens now?

Now suppose event n comes, what happens? Transition \( \frac{m}{n} \) is fired, f is generated, which fires transition \( \frac{n}{f} \), which again fires \( \frac{f}{d} \).
Now suppose event \( n \) comes, transition \( T \) is fired, \( f \) is generated, which fires transition \( G \), which again fires \( J \), finally yielding \( B.E.J \).

What are the triggers/actions

- Method call
  - \( \text{Method\_name(parameters)} \)
- Or, Event
  - \( \text{Event\_name(parameters)} \)
- Is there a difference?
  - Lots, in terms of semantics
    - A method call involves a transfer of control
      - If there are nested method calls, they can cause further transfer of control
    - An event will be lodged in a system queue
      - It will be removed by the recipient later.

Events and Method calls

- Event based communication
  - Inherently asynchronous
    - Designer does not worry about controlling all interaction sequences (this is taken care of by the system queue)
- Method call based communication
  - Synchronous, involving transfer of control
  - Involves close control by the designer over interaction sequences
    - getting closer to code level

Most General form of ...

- ... annotation for a transition
  - \( \text{Trigger[condition]/Action} \)
  - \( \text{Trigger is event expression or method invocation} \)
  - \( \text{Condition is like a branch condition on data variables} \)
  - \( \text{Action is a program} \)
    - Sequence of event generation or method invocation or even code in a programming language.

Summary

- Practical Use of Statecharts in Modeling Object-based systems
  - Use statecharts to describe behavior of classes (of active objects)
  - Class Associations given by class diagrams.
  - Contains code in the actions for realistic designs
  - A realistic approach for modeling (distributed) embedded controllers.