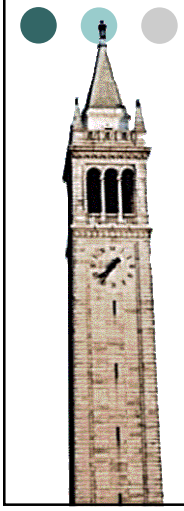


Introduction to Embedded Systems

Edward A. Lee & Sanjit A. Seshia
UC Berkeley
EECS 124
Spring 2008

Copyright © 2008, Edward A. Lee & Sanjit A. Seshia, All rights reserved

Lecture 6: Modeling Modal Behavior, Part I

Roadmap for course (until spring break):
Modeling, Analysis, and Control

What goes into "CAD tools" for embedded systems?

Modeling

Feb 11, 13: "Pure" Modal Modeling (finite-state machines)

Feb 20: Concurrency

Feb 25, 27: Hybrid Systems (modeling discrete + continuous behavior)

Roadmap for course (until spring break)

Analysis and Control

Analysis: Does my model satisfy its specification?

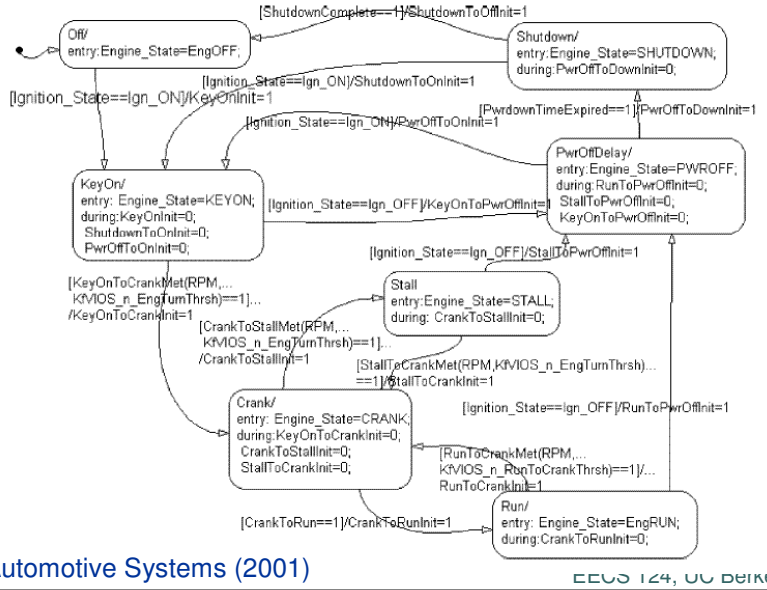
Mar 3, 5: Techniques to simulate discrete/continous/hybrid models

Mar 10, 12: Exhaustive state-space exploration techniques, reachability analysis, model checking

Control: Synthesizing a strategy to achieve a goal

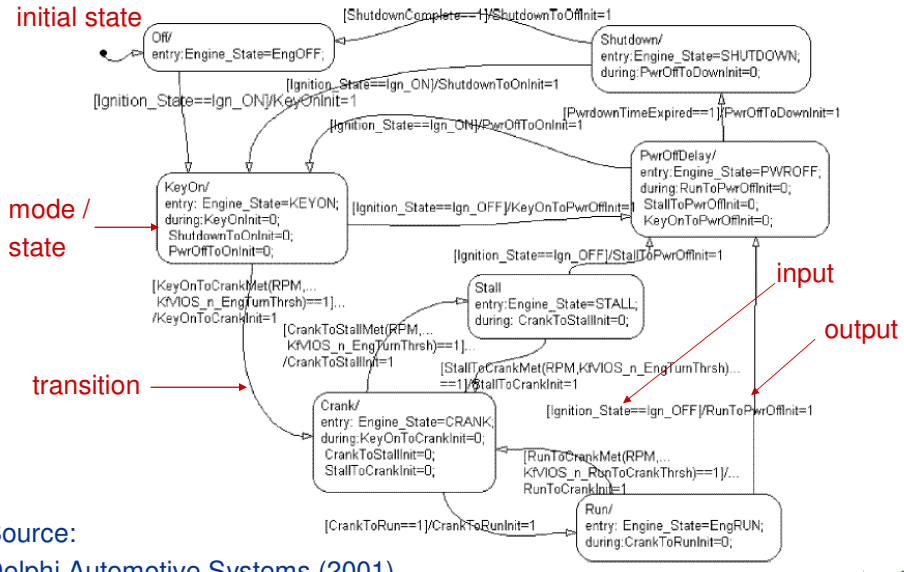
Mar 17, 19: Controller synthesis for hybrid systems

Modal Model (Finite-State Machine) for Engine Control



Source:
Delphi Automotive Systems (2001)

Elements of a Modal Model (FSM)



Source:
Delphi Automotive Systems (2001)

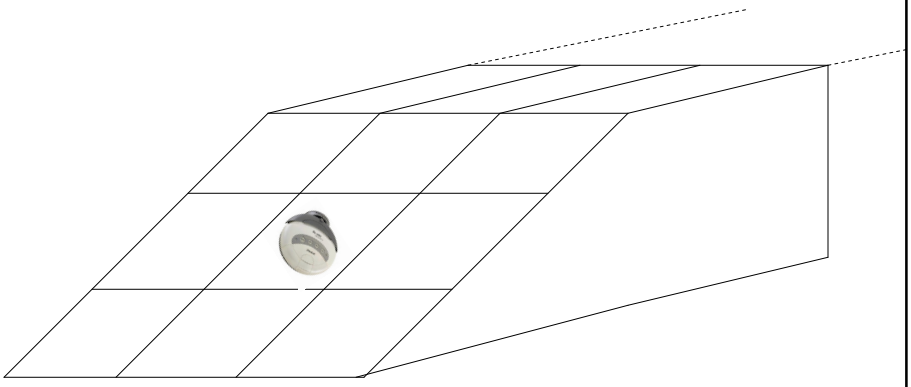
Topic: Modeling with Finite-State Machines (FSMs)

Suppose that our only modeling formalism is the Finite-State Machine

Four questions:

1. How to represent the system for:
   - Mathematical analysis
   - So that a computer program can manipulate it
2. How to model its environment?
3. How to represent what the system MUST do – its specification?
4. How to check whether the system satisfies its specification in its operating environment?

Example: Discretized iRobot Hill Climber



Main Concepts in this Lecture

Q1: System representation

- Finite-state machine – syntax and semantics
- Behavior/trace on infinite inputs

Q2 & 3: Environment modeling/specification

- Non-determinism
  - Abstraction in modeling

Q4: Comparing state machines

- Simulation, bisimulation, trace equivalence and trace containment
  - How do we know whether implementation does what the specification states?

State Machines: Formal Definition

A state machine is a tuple:
(States, Inputs, Outputs, update, initialState)
where

States = set of states/modes of the system

Inputs = set of input symbols

Outputs = set of output symbols

update: States $\times$ Inputs → States $\times$ Outputs

- called “transition function” or “update function”

initialState = the starting state/mode of the system

- denoted by incoming arrow or special “init” label

Finite State Machines

The definition on the previous slide is general – the state machine could have *infinitely many* states

We will restrict ourselves to a **finite** set of states, in this and the next lecture

FSM Steps and Signals

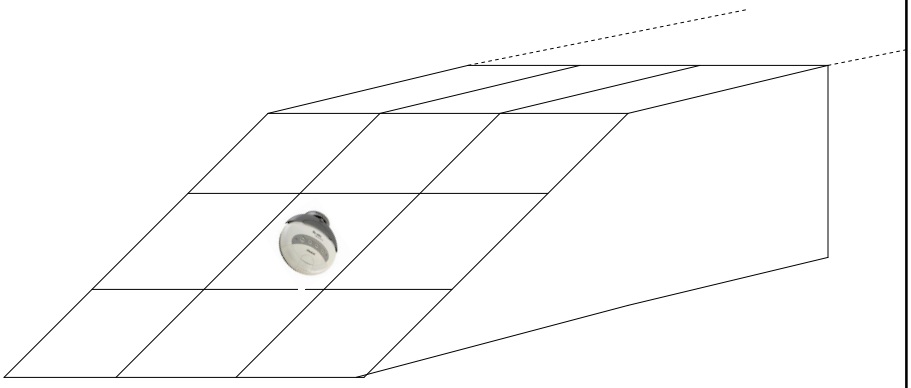
Each time a transition occurs, the FSM takes a **step**
(note: time need not advance)
StepNumbers = {0, 1, 2, … } = $\mathbb{N}$

A **signal** is a function mapping $\mathbb{N}$ to symbols

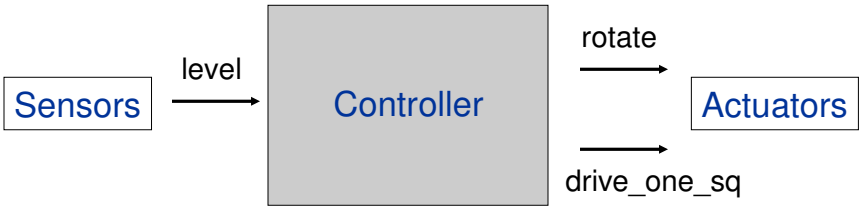
- Input, output, and control (state-holding) signals
- We will restrict attention to "pure" signals
  - Signal models presence/absence of an event

An **input/output symbol** is an element of
{present, absent}^{n} (assuming n input/output signals)

Example: Discretized iRobot Hill Climber

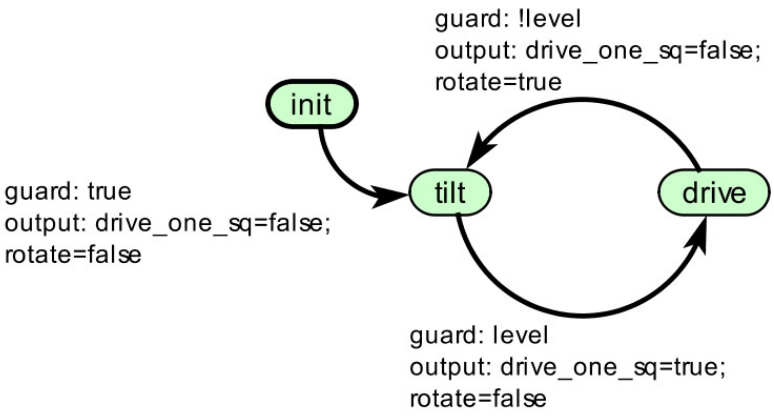


The Input/Output Interface of the iRobot Controller



Signal that is present	What it means
level	Robot is facing the right way up/down hill
rotate	Robot rotates by 45^o clockwise
drive_one_sq	Robot drives to neighboring square

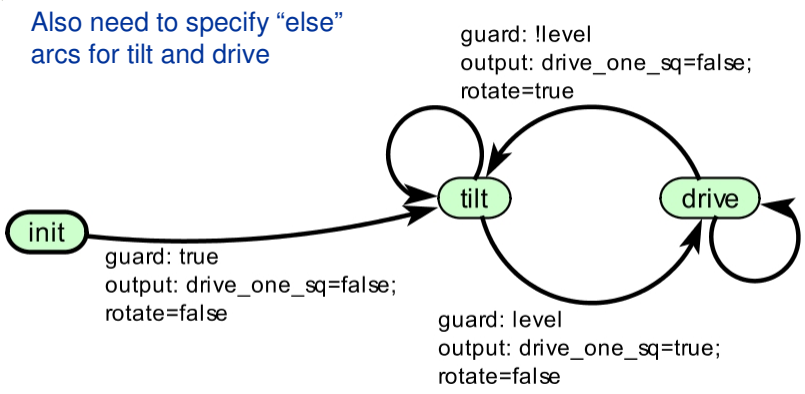
FSM Controller for iRobot



States = {init, tilt, drive} Inputs = ? Outputs = ?

update = ? Any transitions missing?

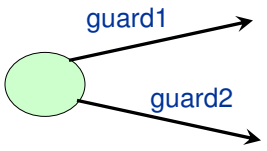
FSM Controller for iRobot (version 2)



Will this robot always drive uphill?
(assume that it starts facing uphill)

Properties of FSMs: **Determinacy**

- An FSM is deterministic if
  - Different transitions out of a state do not have overlapping guards



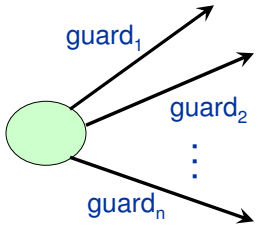
$guard1 \cap guard2 = \emptyset$

Is the iRobot FSM (prev. slide) *deterministic*?

Note: A guard is a subset of Inputs

Properties of FSMs: **Receptiveness**

- An FSM is receptive if
  - It has a transition defined for *every input symbol* from *every state*



$\bigcup_i \text{guard}_i = \text{Inputs}$

Is the iRobot FSM *receptive*?

Behavior of a FSM

How do we model the input-output behavior of an FSM?

Let
InputSeq = set of infinite sequences of input symbols
OutputSeq = set of infinite sequences of output symbols

Then, the behavior of an FSM is
a function from InputSeq *to* OutputSeq

Trace

A trace of a FSM is an infinite sequence of triples

$(i_0, s_0, o_0), (i_1, s_1, o_1), (i_2, s_2, o_2), \ldots$

where $\text{update}(s_j, i_j) = (s_{j+1}, o_j)$

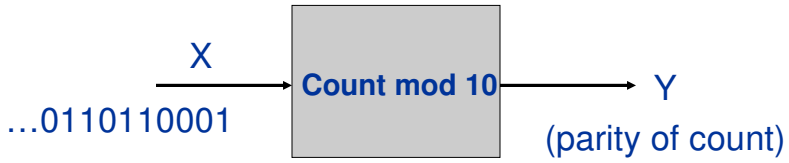
Also written as:

$$s_0 \xrightarrow{i_0 / o_0} s_1 \xrightarrow{i_1 / o_1} s_2 \xrightarrow{i_2 / o_2} s_3 \quad \ldots.$$

Sometimes we are only interested in the "observable" trace of the system, which includes only input & output symbols $(i_0, o_0), (i_1, o_1), (i_2, o_2), \ldots$

- Note: as above, the trace represents a single system behavior

Example: a Counter



Suppose the count starts with 0

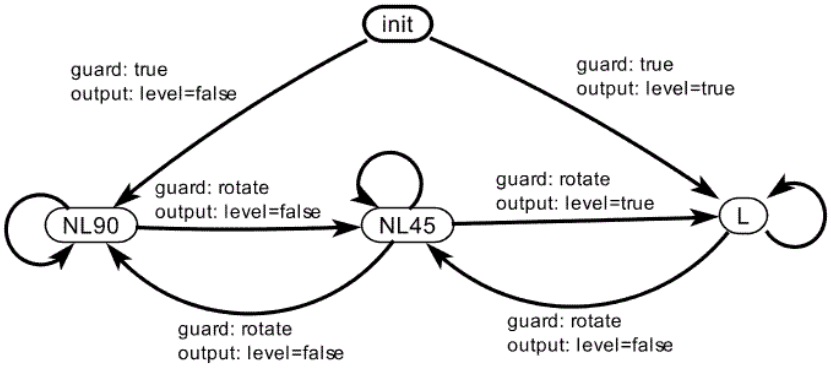
Trace is

(1,0,0), (0,1,1), (0,1,1), (0,1,1), (1, 2, 0), (1, 3, 1), ….

Example 2:

Give an example trace for the iRobot modal model on slide 15

Modeling the iRobot's environment

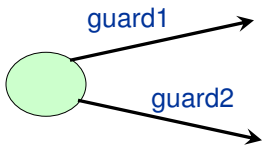


Is this model **deterministic**?

L	level=true
NL45	level=false, 45^o offset
NL90	level=false, 90^o offset

Self loops on: rotate=false

Non-determinism

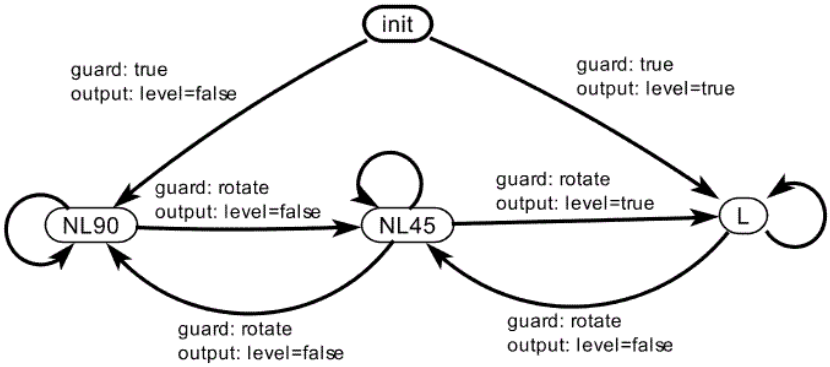


guard1 ∩ guard2 ≠ ∅

More than one transition possible on an input symbol

For the iRobot environment:
non-determinism in transition from initial state

More non-determinism in iRobot's environment



Self loops on rotate=false

How will the FSM change if we want to model this:
Lifting up the robot at any step and placing it with a different orientation

Uses of non-determinism

1. Modeling unknown aspects of the environment or system
   - Such as: how the environment changes the iRobot's orientation
2. Hiding detail in a *specification* of the system
   - We will see an example of this later (see notes to be posted)

Any other reasons why non-deterministic FSMs might be better than deterministic FSMs?

Size Matters

Non-deterministic FSMs are more compact than deterministic FSMs

- ND FSM → D FSM: Exponential blow-up in #states in worst case

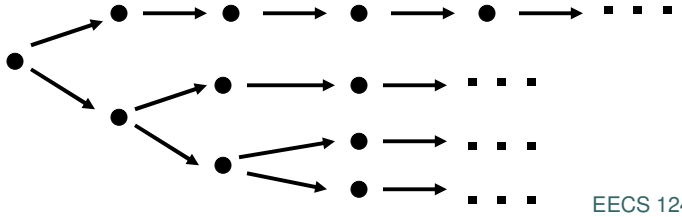
Non-deterministic Behavior: Tree of Computations

For a fixed input sequence:

- A deterministic system exhibits a single behavior
- A non-deterministic system exhibits a **set of behaviors**
  - visualized as a *computation tree*

Deterministic FSM behavior:

Non-deterministic FSM behavior:



Non-determinism: Formal Definition

A non-deterministic state machine is a tuple:
(States, Inputs, Outputs, possibleUpdates, initialStates)
where

States = set of states/modes of the system

Inputs = set of input symbols

Outputs = set of output symbols

possibleUpdates: States × Inputs → $2^{\text{States} \times \text{Outputs}}$

- also termed as "transition relation"

initialStates = the starting states/modes of the system

Related points

What does receptiveness mean for non-deterministic state machines?

Non-deterministic ≠ Probabilistic

Representing a state machine

1. Pictorial notation
2. Table representing transition relation
3. Functional notation

When would you use each representation?