state machine modelling

Daniel Jackson

september 5, 2007
why state machines?
software models

why models?

• biggest challenge: bridging the problem-code gap
• models capture essence -- good for exploring problems
• models hide details -- good for exploring solutions
• same concepts in problem and solution models
state machine models

standard model
- states + actions + transitions
- good for ‘control-intensive systems’
- wide use in industry (eg, medical devices, cars, network protocols)

variants
- timed: allow analysis of real-time deadlines
- stochastic: attach probabilities to transitions
- hybrid: handle continuous variables too

but I’m not interested in embedded systems...
- every system has a control aspect
- user interaction usually has strong control aspect
state machines more generally

not just for small machines
• model might be too big to draw, but state machine still best way to express
• state machines don’t have to be finite

for semantics
• state machines are a great basis for semantics
• conceptual model that underlies reasoning about abstract types
• good way to understand the Java language

design models vs. conceptual models
state machine modelling: window manager example
problem: window manager

window manager
  • controls size, shape, stacking, visibility of windows
  • actually a subtle design problem
  • interesting differences between OS X, Windows, Linux

problem: representing behaviour
  • without explicit representation, can't discuss design
  • capture essence: not all details, just important ones
  • gross states of system called "modes"
window manager machine

**elements**

- states
- initial state with bolded boundary
- transitions labelled with action names

max is a ‘toggle’
a state machine consists of

- a set of states
  \[ \text{State} = \{ \text{STD, MAX, DEAD, STDH, MAXH} \} \]
- a set of initial states
  \[ \text{Init} = \{ \text{STD} \} \]
- a set of actions
  \[ \text{Action} = \{ \text{max, min, restore, close} \} \]
- a transition relation
  \[ \text{trans} \subseteq \text{State} \times \text{Action} \times \text{State} = \]
  \[ \{ (\text{STD, max, MAX}), (\text{STD, min, STDH}), (\text{STDH, restore, STD}), \\
  (\text{MAXH, restore, MAX}), (\text{MAX, max, STD}), (\text{MAX, min, MAXH}), \\
  (\text{MAX, close, DEAD}) \} \]
another way to represent the transition relation

instead of a set of transitions

\[
\text{trans: Set (State } \times \text{ Action } \times \text{ State) } = \\
\{ (\text{STD, max, MAX}), (\text{STD, min, STDH}), (\text{STDH, restore, STD}), \\\n(\text{MAXH, restore, MAX}), (\text{MAX, max, STD}), (\text{MAX, min, MAXH}), \\\n(\text{MAX, close, DEAD}) \}
\]

partition the relation, and give state pairs for each action

\[
\text{trans: Action } \rightarrow \text{ Set (State } \times \text{ State) } = \\
\{ (\text{max, \{(STD, MAX), (MAX, STD)\}}), \\\n(\text{min, \{(STD, STDH), (MAX, MAXH)\}}), \\\n(\text{restore, \{(STDH, STD), (MAXH, MAX)\}}), \\\n(\text{close, \{(MAX, DEAD)\}}) \}
\]
variable decomposition

instead of flat state space

- decompose state with variables
- state = assignment to the variables
- transition = variable updates, organized by action
operations on variables
	ransitions for an action
  • called an ‘operation’

precondition
  • what holds before action occurs

postcondition
  • relates state after to state before
  • v’ denotes value of variable v after

vars boolean alive, hide, max;
init alive && !hide && max
op max
pre alive
post max’ == !max
op min
pre alive && !hide
post !hide’
op restore
pre alive && hide
post hide’
op close
pre alive && !hide
post !alive’
resizing

exercise

• can you specify resizing?
• add an operation resize \((s)\) that corresponds to manual resizing by user
• variables added to state?

one solution

• many others possible
• max toggles between maximum and manually set size; initially, default size
• min, restore don’t modify size

```
vars
    boolean alive, hide, max;
    int size, saveSize;

init
    alive && hide && max
    size == DEFSIZE
    saveSize = DEFSIZE

op resize (int s)
    pre alive && !hide
    post size' == s && saveSize' == s

op max
    pre alive
    post
    max' == !max
    size' == max ? saveSize : MAXSIZE
```
can represent as diagram
• even though not all vars shown
• nodes represent sets of states
more modelling examples
not all actions are inputs!

- output actions: controlled by machine, visible to user
- internal actions: controlled by machine, not visible to user

example: a browser cache

- output action: display
- internal actions: flush and load

non-determinism

- when does flush happen?
- model doesn’t say
- but load is only action that can follow request
non-determinism

machine is non-deterministic if in some state
\( >1 \) output or internal action enabled
\( >1 \) input action can lead to \( >1 \) state

use #1: implementation freedom
machine describes range of allowed behaviours

use #2: modelling failure
show that whatever happens, some property holds
example: email delivery

- despite unreliable network
  - at most one **deliver** action
  - no **die** without **bounce**
timing issues

timing is not expressed

• but can assume outputs are immediate

example: tollbooth

• id is recorded immediately
• reasonable assumption, since bits move faster than cars
review
state machine models

what is a state machine?
• states, actions, transitions
• decomposing with variables
• operations view: pre and post conditions

concepts
• initialization
• non-determinism