Content

• Hierarchical State Machines
• Additional Well-Formedness Constraints
• An intuition for hierarchical states
• Entry and Exit Actions
• Initial and Final States
• Rhapsody Demo: Automated Tests
• Hierarchical State Machines: The Rest
• History Connectors
• Junction and Choice
• Entry and Exit Points
• Terminate
• Active vs. Passive Objects
Now, what is it exactly while the do action is executing...

• a transformer corresponding to one initiation transition of the top region.

\[ C \]

\[ C \]

\( \text{the transformer of the " constructor" of } C \)

If class

\[ C \]

\[ C \]

\( \text{has a state-machine, then " create-} \)

\[ C \]

\[ C \]

\( \text{C} \)

\( \text{If the do-action terminates, then the state is considered } \)

\[ \text{completed} \]

\[ \text{special case} \]

\( \text{if the state is left before termination, the do-action is stopped.} \)

\( \text{Otherwise, } \)

\( \text{could also be: '' at least one'' and choosing one non-deterministically.} \)

For simplicity, we assume exactly one initiation transition per non-empty region.

Initial and Final States

\[ \text{Principle} \]

\[ \text{Initial Pseudostate} \]

\[ \text{Do Actions} \]

\[ \text{Initial and Final States} \]

\[ \text{Special case} \]

\( \text{when entering a non-simple state, } \)

\( \text{between } \)

\( \text{entry and exit actions.} \)

\[ \text{Intuition} \]

\[ \text{Alternative View: Entry / Exit / Internal as Abbreviations} \]

\[ \text{Entry / Internal / Exit don’t add expressive power to Core State Machines.} \]

\[ \text{That is} \]

\[ \text{if internal actions should have priority, adjust Rules (ii), (iii), and (v) accordingly.} \]

\( \text{Note:} \)

\( \text{each action is supposed to have a transformer; assume } \)

\[ \text{end and final states} \]

\[ \text{over regular transitions} \]

\[ \text{Note:} \]

\( \text{intuition} \)
• If \((\sigma, \epsilon)\) \(-\rightarrow u(\sigma', \epsilon')\) and all simple states \(s\) in \(\sigma\) \((u(st))\) are final, i.e. \(\text{kind}(s) = \text{fin}\), then
  - stay unstable if there is a common parent of the simple states in \(\sigma\) \((u(st))\) which is source of a transition without trigger and satisfied guard,
  - otherwise kill (destroy) object \(u\).

\[\Rightarrow\] adjust Rules (i), (ii), (iii), and (v) accordingly.

Observation: \(u\) never "survives" reaching a state \((s, \text{fin})\) with \(s \in \text{child}(\text{top})\).
Junction and Choice

• Junction ("static conditional branch")
  • $gd_1/act_1$ \[ gd_2/act_2 \]
  • good: abbreviation
  • unfolds to so many similar transitions with different guards, the unfolded transitions are then checked for enabledness
  • at best, start with trigger, branch into conditions, then apply actions

• Choice ("dynamic conditional branch")
  • evil: may get stuck
  • enters the transition without knowing whether there's an enabled path
  • at best, use "else" and convince yourself that it cannot get stuck
  • maybe even better: avoid

Hierarchical states can be "folded" for readability.
(but: this can also hinder readability.)
Can even be taken from a different state-machine for re-use.

• Entry/exit points
  • Provide connection points for finer integration into the current level, finer than just via initial state.
  • Semantically a bit tricky:
    • First the exit action of the exiting state,
    • then the actions of the transition,
    • then the entry actions of the entered state,
    • then action of the transition from the entry point to an internal state,
    • and then that internal state's entry action.

• Terminate Pseudo-State
  • When a terminate pseudo-state is reached, the object taking the transition is immediately killed.

OR- and AND-states could also be explained as an "unfolding" into core state machines.
They add conciseness, not expressive power.
The remaining pseudo-states (history, junction, choice, etc.) are not so difficult.
Modelling guideline: Avoid choice.
Rhapsody also supports non-active objects — their instances share an event pool with an active object.

References