Contents & Goals

Last Lecture:
• Initial and Final State
• Composite State Semantics started

This Lecture:
• Educational Objectives:
  • What does this State Machine mean? What happens if I inject this event?
  • Can you please model the following behaviour.
  • What does this hierarchical State Machine mean? What may happen if I inject this event?
  • What is: AND-State, OR-State, pseudo-state, entry/exit/do, final state, . . .

Content:
• Composite State Semantics cont'd
• The Rest

A Partial Order on States

The substate- (or child-) relation induces a partial order on states:
• top \leq s, for all \(s \in S\),
• \(s \leq s'\), for all \(s' \in \text{child}(s)\),
• transitive, reflexive, antisymmetric,
• \(s' \leq s\) and \(s'' \leq s\) implies \(s' \leq s''\) or \(s'' \leq s'\).

Least Common Ancestor and Ting

• The least common ancestor is the function \(lca : 2^S \{\emptyset\} \rightarrow S\) such that
  • The states in \(S_1\) are (transitive) children of \(lca(S_1)\), i.e.
    \[ lca(S_1) \leq s, \text{ for all } s \in S_1 \subseteq S, \]
  • \(lca(S_1)\) is minimal, i.e. if \(\hat{s} \leq s\) for all \(s \in S_1\), then \(\hat{s} \leq lca(S_1)\).
  • Note: \(lca(S_1)\) exists for all \(S_1 \subseteq S\) (last candidate: top).

Least Common Ancestor and Ting

• Two states \(s_1, s_2 \in S\) are called orthogonal, denoted \(s_1 \perp s_2\), if and only if
  • they are unordered, i.e. \(s_1 \not\leq s_2\) and \(s_2 \not\leq s_1\), and
  • they "live" in different regions of an AND-state, i.e.
    \[ \exists s, \text{region}(s) = \{S_1, \ldots, S_n\}, \exists 1 \leq i \neq j \leq n: s_1 \in \text{child}^*(S_i) \land s_2 \in \text{child}^*(S_j), \]
Deferred Events: Idea

• Consider the following state machine:

\[
\begin{align*}
S_1 & \quad S_2 & \quad S_3 \\
E & \quad F & \quad / \\
/ & \quad / & \quad /
\end{align*}
\]

• Assume we're stable in \( S_1 \), and \( F \) is ready in the ether.

• In the framework of the course, \( F \) is discarded.

• But we may find it a pity to discard the poor event and may want to remember it for later processing, e.g. in \( S_2 \), in other words, defer it.

General options to satisfy such needs:

• Provide a pattern how to "program" this (use self-loops and helper attributes).

• Turn it into an original language concept. (← OMG's choice)

Deferred Events: Syntax and Semantics

• Syntactically,

• Each state has (in addition to the name) a set of deferred events.

• Default: the empty set.

• The semantics is a bit intricate, something like

• if an event \( E \) is dispatched,

• and there is no transition enabled to consume \( E \),

• and \( E \) is in the deferred set of the current state configuration,

• then stuff \( E \) into some "deferred events space" of the object, (e.g. into the ether (\( = \) extend \( \epsilon \)) or into the local state of the object (\( = \) extend \( \sigma \)))

• and turn attention to the next event.

• Not so obvious:

• Is there a priority between deferred and regular events?

• Is the order of deferred events preserved?

... [Fecher and Schönborn, 2007], e.g., claim to provide semantics for the complete Hierarchical State Machine language, including deferred events.

And What About Methods?

• In the current setting, the (local) state of objects is only modified by actions of transitions, which we abstract to transformers.

• In general, there are also methods.

• UML follows an approach to separate

• the interface declaration from

• the implementation.

In C++ lingo: distinguish declaration and definition of method.

• In UML, the former is called behavioural feature and can (roughly) be

\[
C \xi_1 f(\tau_1, 1, \ldots, \tau_{1,n}) : \tau_1 P_1 \xi_2 F(\tau_2, 1, \ldots, \tau_{2,n}) : \tau_2 P_2 \langle\langle \text{signal} \rangle\rangle E_a
\]

Note: The signal list can be seen as redundant (can be looked up in the state machine) of the class. But: certainly useful for documentation (or sanity check).
Behavioural Features: Visibility and Properties

Visibility:

- Extend typing rules to sequences of actions such that a well-typed action sequence only calls visible methods.

Useful properties:

- Concurrency — is thread safe
- Guarded — some mechanism ensures/should ensure mutual exclusion
- Sequential — is not thread safe, users have to ensure mutual exclusion
- IsQuery — doesn’t modify the state space (thus thread safe)

For simplicity, we leave the notion of steps untouched, we construct our semantics around state machines. Yet we could explain pre/post in OCL (if we wanted to).

Discussion.

Semantic Variation Points

Pessimistic view:

- They are legion...
- For instance, allow absence of initial pseudo-states
- can then “be” in enclosing state without being in any substate; or assume one of the children states non-deterministically (implicitly) enforce determinism, e.g., by considering the order in which things have been added to the CASE tool’s repository, or graphical order
- allow true concurrency

Exercise: Search the standard for “semantical variation point”.

• [Crane and Dingel, 2007], e.g., provide an in-depth comparison of Statemate, UML, and Rhapsody state machines — the bottom line is:
  - the intersection is not empty (i.e., there are pictures that mean the same thing to all three communities)
  - none is the subset of another (i.e., for each pair of communities exist pictures meaning different things)

Optimistic view:

- tools exist with complete and consistent code generation.


