Contents & Goals

Last Lecture:
• Hierarchical State Machines
• Later active vs. passive; behavioural feature (aka. methods).

This Lecture:
• Educational Objectives:
  - Capabilities for following tasks/questions.
• What does this LSC mean?
• Are these UML model’s state machines consistent with the interactions?
• Please provide a UML model which is consistent with this LSC.
• What is: activation, hot/cold condition, pre-chart, etc.?

Content:
• Remaining pseudo-states, such as shallow/deep history
• Reflective description of behaviour.
• LSC concrete and abstract syntax.
• LSC intuitive semantics.
• Symbolic B¨uchi Automata (TBA) and its (accepted) language.

The Concept of History, and Other Pseudo-States

History and Deep History: By Example

Junction and Choice

• Junction ("static conditional branch"):
  - $[gd_1]/act_1$
  - $[gd_2]/act_2$

• Choice ("dynamic conditional branch")

Note: not so sure about naming and symbols, e.g., I’d guessed it was just the other way round...
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

Note: not sure about naming and symbols, e.g., I’d guessed it was just the other way round...

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.
- $s$: entry/exit points,
  - Provide connection points for finer integration into the current level, 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 the transition from the entrypoint to an internal state,
    - And then the internal state’s entry action.

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

Deferred Events in State-Machines

Deferred Events: Idea
For ages, UML state machines comprise the feature of deferred events. The idea is as follows:
- Consider the following state machine:
  - $s_1$, $s_2$, $s_3$
  - $E/F$
  - 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 (uses 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 ($\epsilon$) or into the local state of the object ($\sigma$)) 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., claims to provide semantics for the complete Hierarchical State Machine language, including deferred events.
Note that an OCL expression or an object diagram and
•
are the corresponding OCL satisfaction of configurations
•
\( \pi_0 \rightarrow \cdots \in \), \( \varepsilon \),

\( \sigma = (\pi_0, SD), Cons \),

either satisfied or not satisfied by a computation \( \varphi_i \): 

Recall: What is a Requirement?

Firstly, the OCL expression will be evaluated, and if the result is true then the desired or undesired.

Then it is decided whether the assertion is true or false.

In general, the OCL expression of certain properties.

A constructive description tells how the model of the system is defined. That is, its constructs contain information needed in preparation for verification.

More formally, what is represented by object diagram relation.

But is finally answered by instance, each reception of each sequence, non-reachability is either satisfied or not satisfied by a computation \( \varphi_i \) shall or shall not be computed.

Mathematics

Course Map

Motion Algorithm Development Description of Behavior

OCL as Reflective Description of Certain Properties

Recent: What is a Requirement?


**Intuitive Semantics:** A Partial Order on Simclasses

(i) **Strictly After:**

(ii) **Simultaneously:** (simultaneous region)

(iii) **Explicitly Unordered:** (co-region)

Intuition:

A computation path violates an LSC if the occurrence of some events doesn't adhere to the partial order obtained as the transitive closure of (i) to (iii).

**LSC Specialty: Activation**

One major defect of MSCs and SDs: they don't say when the scenario has to/maybe observed.

**LSC:** Activation condition (AC ∈ Expr/CB), activation mode (AM ∈ {init, inv}), and pre-chart.

Intuition:

(1) given a computation π, whenever expr holds in a configuration (σi, εi) of ξ (ξ initial, i.e. k = 0, or (AM = initial)) and if the pre-chart is observed from k to k + n, then the main-chart has to follow from k + n + 1.

---

**Course Map**

UML Model Instances

N
S
WE
CD
SM

\[ \begin{align*}
\phi & \in \text{OCL} \\
\text{CD}, & \text{SD}/CB, \text{SD} = (Q_{SD}, q_0, A_{SD}, \rightarrow_{SD}, F_{SD}) \\
\pi & = (\sigma_0, \varepsilon_0) \xrightarrow{u} \cdots \xrightarrow{w} \pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \\
\mathbb{G} & = (\mathbb{N}, \mathbb{E}, f) \\
\end{align*} \]
Let \( l \in \{ l, b, l' \} \).

\[ \exists l \in L, \quad \text{or} \quad l' \in L, \quad \text{or} \quad \text{is minimal wrt.} \quad l. \]

\[ \text{LocInv,} \quad i, \theta, l, \quad \text{i.e.} \quad \text{LocInv} \times (\theta, \epsilon) \times L, \quad \text{or} \quad \epsilon \in L, \quad \text{is a set of} \quad \{ 1 \text{\ and } 3 \text{\ \ conditions} \} \times (\theta, \epsilon) \times L. \]

Example:

\[ \{ \text{BarrierCtrl} \} \times \{ \text{LightsCtrl} \} \times \{ \text{LSCs} \} \]

\[ \text{LSCs: Breathing life into machines: Complete formal semantics via core state machines.} \]

\[ \text{In Brim, L., Haverkort, B. C. and Janssen, J. (1998).} \]

\[ \text{Formal Methods in System Design,} \]

\[ \text{Springer-Verlag, LNCS, volume 1254 of} \text{CA V} \]

\[ \text{Technical Report formal/07-11-02.} \]

\[ \text{Technical Report formal/07-11-04.} \]

\[ \text{Harel, D. (1997).} \]

\[ \text{Somethoughts on statecharts, 13 years later. In Grumberg, O. et al.} \]

\[ \text{Springer-Verlag, LNCS, volume 4346 of} \text{CA V} \]

\[ \text{Operational expression of local invariants on locations, the condition \( i = 0 \), with messages \( t(10) \), is a finite, non-empty, partially ordered set \( \mathbb{P} \).} \]

\[ \text{is a set of} \quad \text{LSC body} \times \text{LSC head} \times \epsilon, \quad \text{for \( \epsilon \in L \).} \]

\[ \text{in LRSL.} \]

\[ \text{Example:} \]

\[ \{ \text{CRS} \} \times \{ \text{LSCs} \} \times \{ \text{LSC body} \} \times \{ \text{LSC head} \} \]