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 this 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 Büchi Automata (TBA) and its (accepted) language.
The Concept of History, and Other Pseudo-States

History and Deep History: By Example

What happens on... (right of continuation)

- $R_s$? $s_0, s_2$
- $R_d$? $s_0, s_2$
- $A, B, C, S, R_s$? $s_0, s_1, s_3, s_5, s_7, s_9$
- $A, B, S, R_d$? $s_0, s_1, s_3, s_5, s_7$
- $A, B, C, D, E, R_s$? $s_0, s_1, s_3, s_5, s_7, s_9$
- $A, B, C, D, R_d$? $s_0, s_1, s_3, s_5, s_7$
Junction and Choice

- Junction ("static conditional branch"):
  - 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")

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"):
  - **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 so sure about naming and symbols, e.g., I'd guessed it was just the other way round...

---

**Entry and Exit Point, Submachine State, Terminate**

- 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, 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.
Deferred Events in State-Machines

Deferred Events: Idea

For ages, UML state machines comprises the feature of deferred events.

The idea is as follows:

• Consider the following state machine:

```
   s1 --E/--> s2 --F/--> s3
```

• 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 ($= \text{extend } \varepsilon$) or into the local state of the object ($= \text{extend } \sigma$))
  - 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.

---

You are here.
Motivation: Reflective, Dynamic Descriptions of Behaviour
Recall: Constructive vs. Reflective Descriptions

[Harel, 1997] proposes to distinguish constructive and reflective descriptions:

- “A language is constructive if it contributes to the dynamic semantics of the model. That is, its constructs contain information needed in executing the model or in translating it into executable code.”

  A constructive description tells how things are computed (which can then be desired or undesired).

- “Other languages are reflective or assertive, and can be used by the system modeler to capture parts of the thinking that go into building the model – behavior included –, to derive and present views of the model, statically or during execution, or to set constraints on behavior in preparation for verification.”

  A reflective description tells what shall or shall not be computed.

Note: No sharp boundaries!

Recall: What is a Requirement?

Recall:

- The semantics of the UML model $\mathcal{M} = (\mathcal{C} \cup \mathcal{M}, \mathcal{C} \cup \mathcal{P})$ is the transition system $(S, \rightarrow, S_0)$ constructed according to discard/dispatch/commence-rules.

- The computations of $\mathcal{M}$, denoted by $[\mathcal{M}]$, are the computations of $(S, \rightarrow, S_0)$.

Now:

A reflective description tells what shall or shall not be computed.

More formally: a requirement $\vartheta$ is a property of computations, sth. which is either satisfied or not satisfied by a computation

$$\pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons }_0, \text{Snd }_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(\text{cons }_1, \text{Snd }_1)} \cdots \in [\mathcal{M}],$$

denoted by $\pi \models \vartheta$ and $\pi \not\models \vartheta$, resp.
OCL as Reflective Description of Certain Properties

- invariants:
  \[\forall \pi \in [M] \forall i \in \mathbb{N} : \pi^i \models \vartheta,\]

- non-reachability of configurations:
  \[\nexists \pi \in [M] \forall i \in \mathbb{N} : \pi^i \not\models \vartheta\]
  \[\iff \forall \pi \in [M] \forall i \in \mathbb{N} : \pi^i \models \neg \vartheta\]

- reachability of configurations:
  \[\exists \pi \in [M] \exists i \in \mathbb{N} : \pi^i \models \vartheta\]
  \[\iff \neg (\forall \pi \in [M] \forall i \in \mathbb{N} : \pi^i \models \neg \vartheta)\]

where
- \(\vartheta\) is an OCL expression or an object diagram and
- "\(|=|\)" is the corresponding OCL satisfaction or the "is represented by object diagram" relation.

In General Not OCL: Temporal Properties

**Dynamic** (by example)

- reactive behaviour
  - "for each \(C\) instance, each reception of \(E\) is finally answered by \(F\)"
    \[\forall \pi \in [M] : \pi \models \vartheta\]

- non-reaching of system configuration sequences
  - "there mustn’t be a system run where \(C\) first receives \(E\) and then sends \(F\)"
    \[\nexists \pi \in [M] : \pi \models \vartheta\]

- reaching of system configuration sequences
  - "there must be a system run where \(C\) first receives \(E\) and then sends \(F\)"
    \[\exists \pi \in [M] : \pi \models \vartheta\]

**But:** what is "|=|" and what is "\(\vartheta\)"?
**Interactions: Problem and Plan**

**In general:** \( \forall (\exists) \pi \in [M] : \pi |\equiv (\neq) \vartheta \)

**Problem:** what is “\( \equiv \)” and what is “\( \vartheta \)”?

**Plan:**
- Define the **language** \( \mathcal{L}(I) \) of an **interaction** \( I \) — via Büchi automata.
- Define the **language** \( \mathcal{L}(M) \) of a **model** \( M \) — basically its computations.
  - Each computation \( \pi \in [M] \) corresponds to a **word** \( w_\pi \).
  - Then (conceptually) \( \pi |\equiv \vartheta \) if and only if \( w_\pi \in \mathcal{L}(I) \).

**Interactions: Plan**

- In the following, we consider **Sequence Diagrams** as **interaction** \( I \),
- more precisely: **Live Sequence Charts** [Damm and Harel, 2001].
- We define the **language** \( \mathcal{L}(I) \) of an LSC — via Büchi automata.
- Then (conceptually) \( \pi |\equiv \vartheta \) if and only if \( w_\pi \in \mathcal{L}(I) \).

Why LSC, relation LSCs/UML SDs, other kinds of interactions: **later**.
Example

Live Sequence Charts — Concrete Syntax
Example: What Is Required?

- Whenever the CrossingCtrl has consumed a 'secreq' event
- then it shall finally send 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl is not 'operational' when receiving that event, the rest of this scenario doesn’t apply; maybe there’s another LSC for that case.
- if LightsCtrl is 'operational' when receiving that event, it shall reply with 'lights_ok' within 1–3 time units,
- the BarrierCtrl shall reply with 'barrier_ok' within 1–5 time units, during this time (dispatch time not included) it shall not be in state 'MvUp',
- 'lights_ok' and 'barrier_ok' may occur in any order.
- After having consumed both, CrossingCtrl may reply with 'done' to the environment.

Building Blocks

- Instance Lines:
Building Blocks

- **Conditions and Local Invariants:** \((expr_1, expr_2, expr_3 \in Expr_{\mathcal{S}})\)

- **Messages:** (asynchronous or synchronous/instantaneous)

Note: angle of shifter
ursg. does not match
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: Modes**

With LSCs,

- whole charts,
- locations, and
- elements

have a mode — one of hot or cold (graphically indicated by outline).
**LSC Specialty: Activation**

One **major defect** of MSCs and SDs: they don’t say **when** the scenario has to/may be observed.

---

**LSCs**: Activation condition \( \text{AC} \in \text{Expr} \), activation mode \( \text{AM} \in \{ \text{init}, \text{inv} \} \), and pre-chart.

---

**Intuition**: (universal case)
- given a computation \( \pi \), whenever \( \text{expr} \) holds in a configuration \( (\sigma_k, \varepsilon_k) \) of \( \xi \)
  - which is initial, i.e. \( k = 0 \), or
  - whose \( k \) is not further restricted, and if the pre-chart is observed from \( k \) to \( k + m \)
  - then the main-chart has to follow from \( k + m + 1 \).

---

**Course Map**

---

**UML**

---

**Mathematics**

---

**OD**
Example
LSC Body: Abstract Syntax

Let Θ = {hot, cold}. An LSC body is a tuple

\((I, (\mathcal{L}, \preceq), \sim, \mathcal{S}, \text{Msg}, \text{Cond}, \text{LocInv})\)

- \(I\) is a finite set of instance lines,
- \((\mathcal{L}, \preceq)\) is a finite, non-empty, partially ordered set of locations; each \(l \in \mathcal{L}\) is associated with a temperature \(\theta(l) \in \Theta\) and an instance line \(i_l \in I\),
- \(\sim \subseteq \mathcal{L} \times \mathcal{L}\) is an equivalence relation on locations, the simultaneity relation,
- \(\mathcal{S} = (F, E, V, atr, \delta)\) is a signature,
- \(\text{Msg} \subseteq \mathcal{L} \times \mathcal{S} \times \mathcal{L}\) is a set of asynchronous messages with \((l, b, l') \in \text{Msg}\) only if \(l \preceq l'\),
- \(\text{Cond} \subseteq (2^\mathcal{L} \setminus \emptyset) \times \mathcal{OCL} \times \Theta\) is a set of conditions where \(\mathcal{OCL}\) are OCL expressions over \(W = I \cup \{\text{self}\}\) with \((L, \text{expr}, \theta) \in \text{Cond}\) only if \(l \sim l'\) for all \(l, l' \in L\),
- \(\text{LocInv} \subseteq \mathcal{L} \times \{\circ, \bullet\} \times \mathcal{OCL} \times \Theta \times \mathcal{L} \times \{\circ, \bullet\}\) is a set of local invariants.

Well-Formedness

Bondedness/no floating conditions: (could be relaxed a little if we wanted to)

- For each location \(l \in \mathcal{L}\), if \(l\) is the location of
  - a condition, i.e.
    \[\exists (L, \text{expr}, \theta) \in \text{Cond} : l \in L,\]
  - a local invariant, i.e.
    \[\exists (l_1, i_1, \text{expr,} , \theta, l_2, i_2) \in \text{LocInv} : l \in \{l_1, l_2\},\]
then there is a location \(l'\) equivalent to \(l\), i.e. \(l \sim l'\), which is the location of
- an instance head, i.e. \(l'\) is minimal wrt. \(\preceq\), or
- a message, i.e.
  \[\exists (l_1, b, l_2) \in \text{Msg} : l \in \{l_1, l_2\}.\]

Note: if messages in a chart are cyclic, then there doesn’t exist a partial order (so such charts don’t even have an abstract syntax).
References


