Contents & Goals

Last Lecture:
- Hierarchical State Machines
- Later: Remaining pseudo-states, such as shallow/deep history; active vs. passive; behavioural feature.

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:**
  - Reflective description of behaviour.
  - LSC concrete and abstract syntax.
  - LSC intuitive semantics.
  - Symbolic Büchi Automata (TBA) and its (accepted) language.
You are here.
\[ M = (\Sigma_{\mathcal{G}}, A_{\mathcal{G}}, \rightarrow_{SM}) \]

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

\[ w_\pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \]

\[ G = (N, E, f) \]

\[ \mathcal{G} = (\mathcal{F}, \mathcal{C}, V, \text{atr}) \]

\[ \varphi \in \text{OCL} \]

\[ CD, SM \]

\[ CD, SD \]

\[ \mathfrak{O} \in \text{UML} \]

\[ \mathfrak{M} \in \text{UML} \]

\[ \mathfrak{D} \in \text{UML} \]

\[ N, S, W, E, CD, SM \]
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 $\mathcal{M} = (\mathcal{C}, \mathcal{S}, \mathcal{O})$ 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:**
  \[ \mathcal{M} \models \vartheta \iff \forall \pi \in [\mathcal{M}] \forall i \in \mathbb{N} : \pi^i \models \vartheta, \]

- **non-reachability of configurations:**
  \[ \nexists \pi \in [\mathcal{M}] \ nexists i \in \mathbb{N} : \pi^i \models \vartheta \]
  \[ \iff \forall \pi \in [\mathcal{M}] \forall i \in \mathbb{N} : \pi^i \models \neg \vartheta \]

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

where

- \( \vartheta \) is an OCL expression or an object diagram and
- "\( \models \)" is the corresponding OCL satisfaction
  or the "is represented by object diagram" relation.
Dynamic (by example)

- reactive behaviour
  - “for each $C$ instance, each reception of $E$ is finally answered by $F$”
    \[ \forall \pi \in \llbracket M \rrbracket : \pi \models \vartheta \]

- non-reachability of system configuration sequences
  - “there mustn’t be a system run where $C$ first receives $E$ and then sends $F$”
    \[ \nexists \pi \in \llbracket M \rrbracket : \pi \models \vartheta \]

- reachability of system configuration sequences
  - “there must be a system run where $C$ first receives $E$ and then sends $F$”
    \[ \exists \pi \in \llbracket M \rrbracket : \pi \models \vartheta \]

But: what is “$\models$” and what is “$\vartheta$”?
Interactions: Problem and Plan

**In general:** \(\forall (\exists \pi \in \llbracket \mathcal{M} \rrbracket : \pi \models (\not\models) \varphi)\)

**Problem:** what is “\(=\)” and what is “\(\varphi\)”?

**Plan:**

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

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

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

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

Note: angle of sloped messages is not relevant.

are equivalent.
### Building Blocks

**LSC:** $L$

**AC:** actcond

**AM:** invariant $I$: strict

**Environment**

- **LightsCtrl**
- **CrossingCtrl**
- **BarrierCtrl**

**Operational**

- $t(10)$

- **CrossingCtrl**
- **LightsCtrl**
- **BarrierCtrl**

---

**Conditions and Local Invariants:**

\[(\text{expr}_1, \text{expr}_2, \text{expr}_3 \in \text{Expr} \not\emptyset)\]
**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).
Partial Order Requirements

- **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.
LSC Specialty: Modes

With LSCs,

- whole charts,
- locations, and
- elements

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

---

**chart**

**location**

**message**

**condition/local inv.**

<table>
<thead>
<tr>
<th>Hot:</th>
<th>Location</th>
<th>Message</th>
<th>Condition/Loc inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="chart" alt="Hot Chart" /></td>
<td><img src="location" alt="Hot Location" /></td>
<td><img src="message" alt="Hot Message" /></td>
<td><img src="condition" alt="Hot Condition" /></td>
</tr>
</tbody>
</table>

- always vs. at least once
- must vs. may progress
- mustn’t vs. may get lost
- necessary vs. legal exit
Example: Modes

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

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

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


