Software Design, Modelling and Analysis in UML

Lecture 17: Reflective Description of Behaviour, Live Sequence Charts I

2013-01-16

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

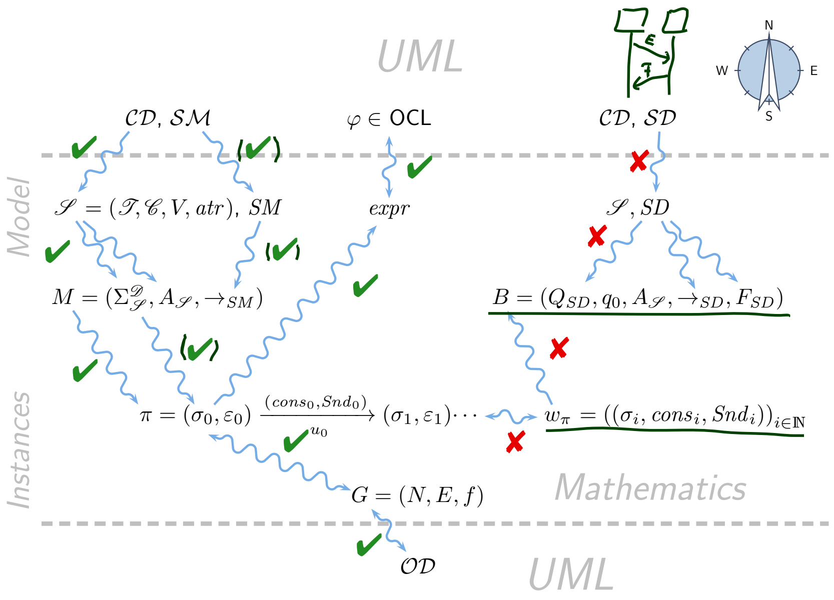
- Hierarchical State Machines
- **Later**: Remaining pseudo-states, such as shallow/deep history; active vs. passive; behavioural feature.

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.

Course Map



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** $\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ is the **transition system** $(S, \to, S_0)$ constructed according to discard/dispatch/commence-rules.
- The **computations of** $\mathcal{M}$, denoted by $[\![\mathcal{M}]\!]$, are the computations of $(S, \to, S_0)$.

Now:

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

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

$$\pi = (\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(cons_1, Snd_1)} \cdots \in [\![\mathcal{M}]\!],$$

denoted by $\pi \models \vartheta$ and $\pi \not\models \vartheta$, resp.

– 17 – 2013-01-16 – Sreflective –

OCL as Reflective Description of Certain Properties

- **invariants**:

the i-th (σ,ε)-pair in π

$$\mathcal{M} \models \vartheta \quad \text{iff} \quad \forall \pi \in [\![\mathcal{M}]\!]\ \forall i \in \mathbb{N} : \pi^i \models \vartheta,$$

- **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

- ϑ is an OCL expression or an object diagram and
- "$\models$" is the corresponding OCL satisfaction or the "is represented by object diagram" relation.

– 17 – 2013-01-16 – Sreflective –

In General Not OCL: Temporal Properties

Dynamic (by example)

- **reactive behaviour**
  - **"for each C instance, each reception of E is finally answered by F"**

$$\forall \pi \in [\![\mathcal{M}]\!] : \pi \models \vartheta$$

- **non-reachability** of system configuration **sequences**
  - **"there mustn't be a system run where C first receives E and then sends F"**

$$\nexists \pi \in [\![\mathcal{M}]\!] : \pi \models \vartheta$$

- **reachability** of system configuration **sequences**
  - **"there must be a system run where C first receives E and then sends F"**

$$\exists \pi \in [\![\mathcal{M}]\!] : \pi \models \vartheta$$

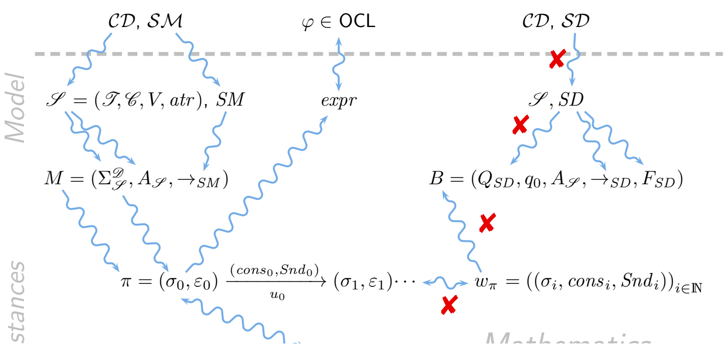
But: what is "$\models$" and what is "ϑ"?

Interactions: Problem and Plan

In general: $\forall(\exists)\ \pi \in [\![\mathcal{M}]\!] : \pi \models(\not\models)\ \vartheta$
Problem: what is "$\models$" and what is "ϑ"?

Plan:

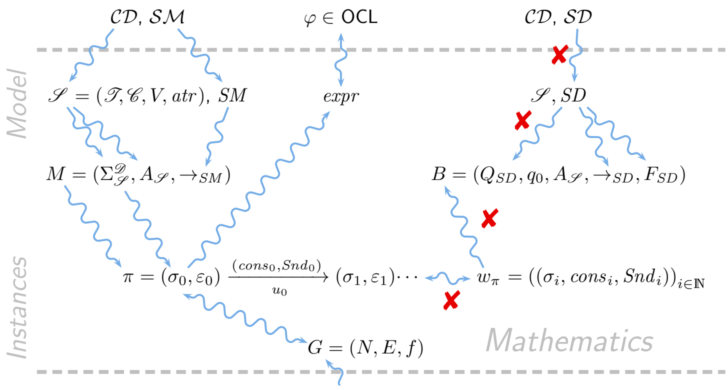
- Define the **language** $\mathcal{L}(\mathcal{I})$ of an **interaction** $\mathcal{I}$ — via Büchi automata.
- Define the **language** $\mathcal{L}(\mathcal{M})$ of a **model** $\mathcal{M}$ — basically its computations. Each computation $\pi \in [\![\mathcal{M}]\!]$ corresponds to a **word** w_π.
- Then (conceptually) $\pi \models \vartheta$ 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** [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 $w_\pi \in \mathcal{L}(\mathcal{I})$.

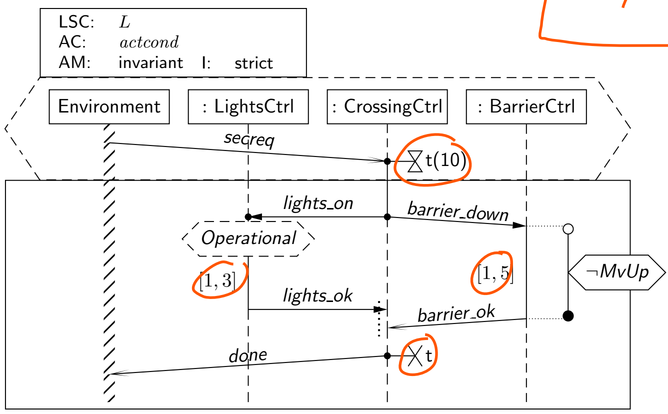
Why LSC, relation LSCs/UML SDs, other kinds of interactions: **later**.



Live Sequence Charts — Concrete Syntax

Example

ADVERTISEMENT: Lecture Real-Time Systems Summer 2013

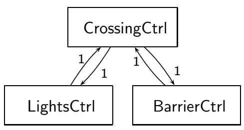


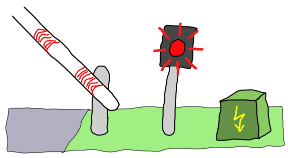
«signal» lights_on

«signal» secreq

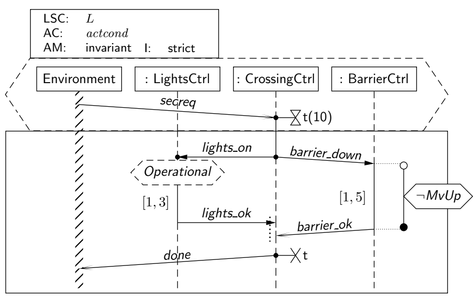
⋮

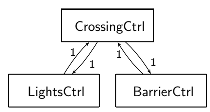
«signal» done

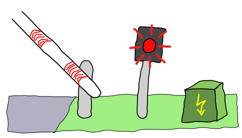




Example: What Is Required?

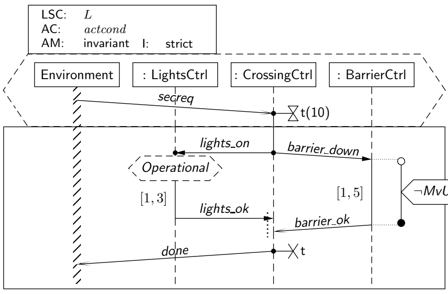


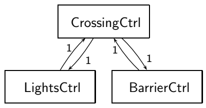


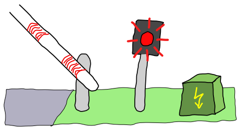


- **Whenever** the CrossingCtrl has consumed a ‘secreq’ event
- **then** it shall finally send ‘lights_on’ and ‘barrier_down’ to LightsCtrl and BarrierCtrl,
- if LightsCtrl **is not** ‘operational’ when receiving that event,
  the rest of this scenario doesn’t apply; maybe there’s another LSC for that 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.

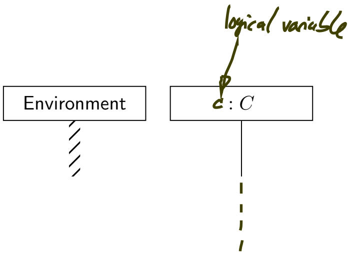
Building Blocks



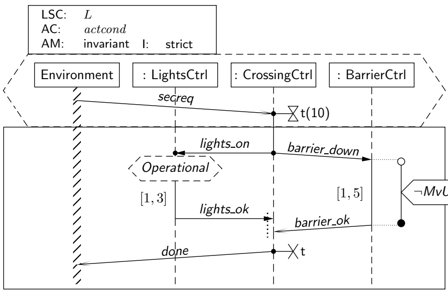


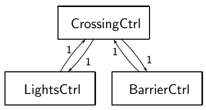


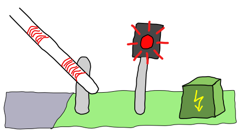
- **Instance Lines:**



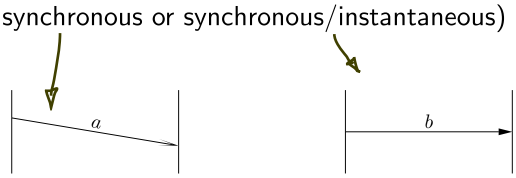
Building Blocks

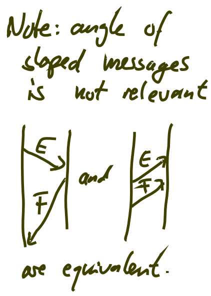




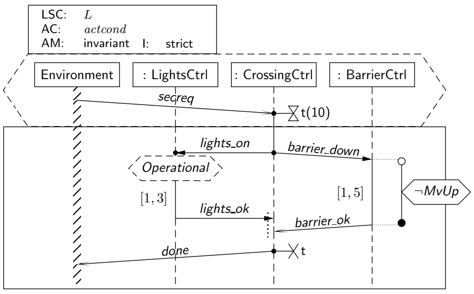


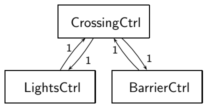
- **Messages:** (asynchronous or synchronous/instantaneous)

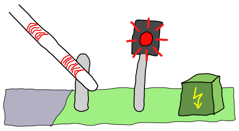




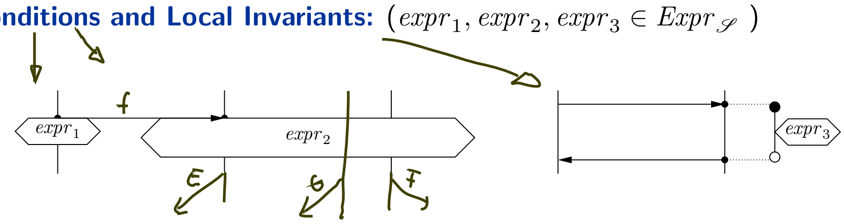
Building Blocks





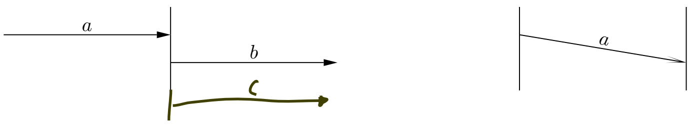


- **Conditions and Local Invariants:** $(expr_1, expr_2, expr_3 \in Expr_{\mathscr{S}})$

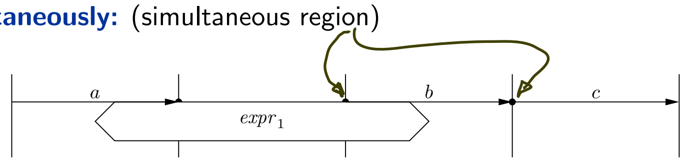


Intuitive Semantics: A Partial Order on Simclasses

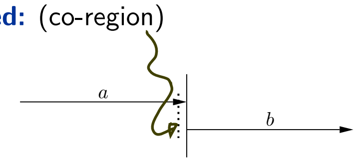
(i) **Strictly After:**



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

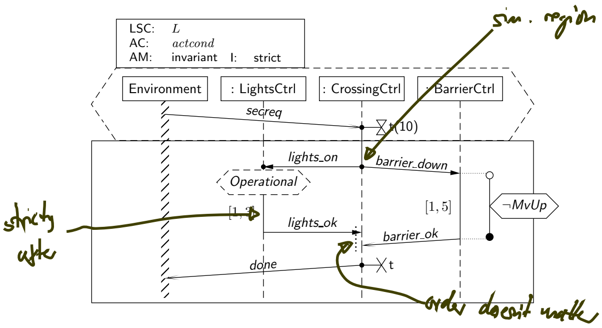


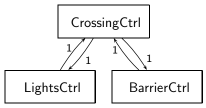
(iii) **Explicitly Unordered:** (co-region)

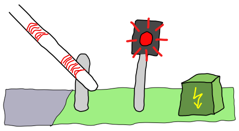


Intuition: A computation path **violates** an LSC if the occurrence of some events doesn't adhere to the partial order obtained as the **transitive closure** of (i) to (iii).

Partial Order Requirements







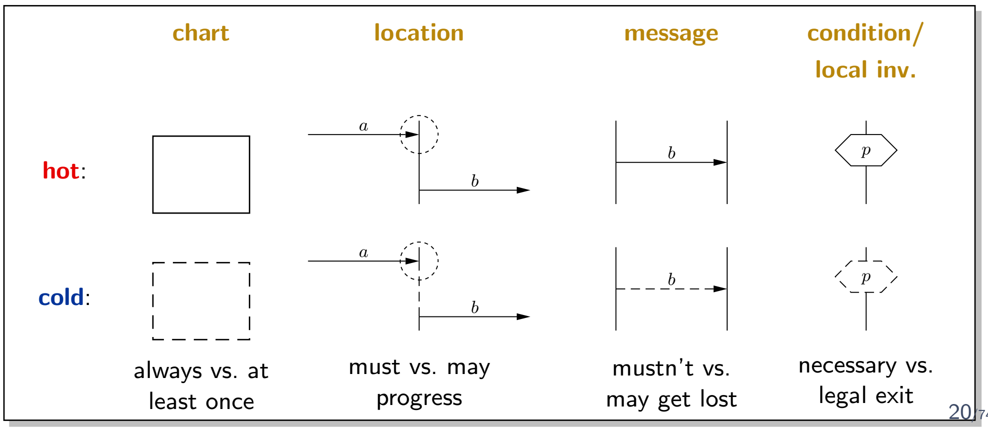
- **Whenever** the CrossingCtrl has consumed a 'secreq' event
- **then** it shall finally send 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl **is not** 'operational' when receiving that event,
  the rest of this scenario doesn't apply; maybe there's another LSC for that case.
- if LightsCtrl **is** 'operational' when receiving that event,
  it shall reply with 'lights_ok' ~~within 1–3 time units~~, strictly late
- 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

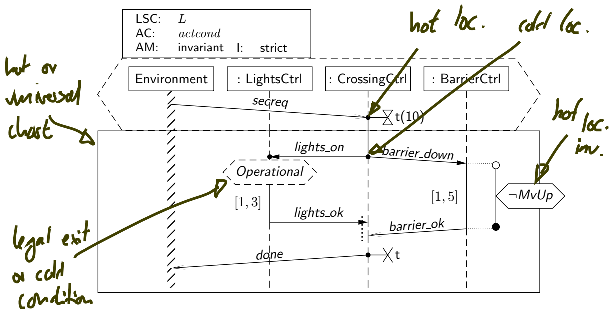
With LSCs,

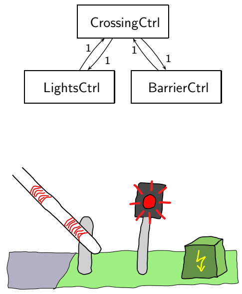
- whole charts,
- locations, and
- elements

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



Example: Modes



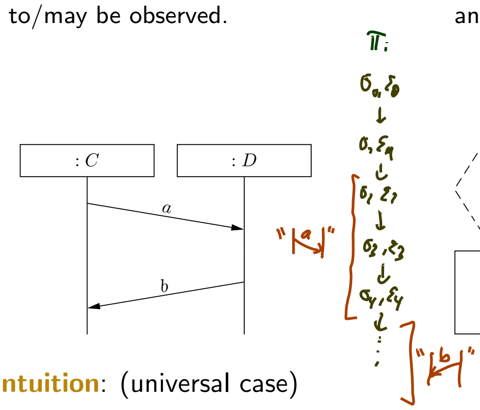


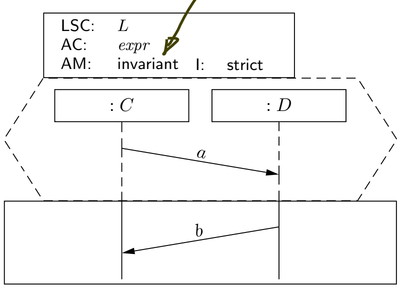
- **Whenever** the CrossingCtrl has consumed a 'secreq' event
- **then** it shall **finally send** 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl **is not** 'operational' when receiving that event, the rest of this scenario doesn't apply; maybe there's another LSC for that 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.

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 (AC $\in Expr_{\mathscr{S}}$), activation mode (AM $\in \{init, inv\}$), and pre-chart.



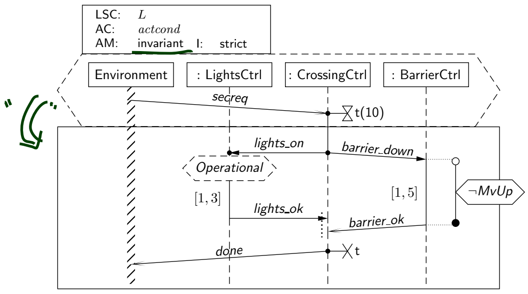


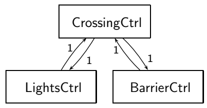
Intuition: (universal case)

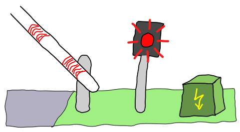
- given a computation π, **whenever** $expr$ holds in a configuration $(\sigma_k, \varepsilon_k)$ of π
  - which is initial, i.e. $k = 0$, or (AM = *initial*)
  - whose k is not further restricted, (AM = *invariant*)

  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?

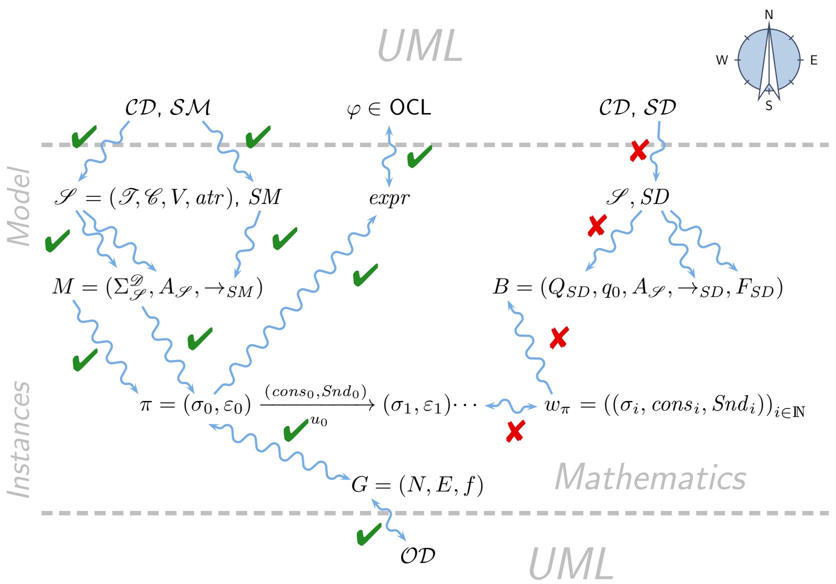






- **Whenever** the CrossingCtrl has consumed a 'secreq' event
- **then** it shall finally send 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl **is not** 'operational' when receiving that event, the rest of this scenario doesn't apply; maybe there's another LSC for that 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.

Course Map



References

References

[Damm and Harel, 2001] Damm, W. and Harel, D. (2001). LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80.

[Harel, 1997] Harel, D. (1997). Some thoughts on statecharts, 13 years later. In Grumberg, O., editor, *CAV*, volume 1254 of *LNCS*, pages 226–231. Springer-Verlag.

[Harel and Maoz, 2007] Harel, D. and Maoz, S. (2007). Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and System Modeling (SoSyM)*. To appear. (Early version in SCESM'06, 2006, pp. 13-20).

[Harel and Marelly, 2003] Harel, D. and Marelly, R. (2003). *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag.

[Klose, 2003] Klose, J. (2003). *LSCs: A Graphical Formalism for the Specification of Communication Behavior*. PhD thesis, Carl von Ossietzky Universität Oldenburg.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Störrle, 2003] Störrle, H. (2003). Assert, negate and refinement in UML-2 interactions. In Jürjens, J., Rumpe, B., France, R., and Fernandez, E. B., editors, *CSDUML 2003*, number TUM-I0323. Technische Universität München.