Software Design, Modelling and Analysis in UML

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

2012-01-25

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Constructive description of behaviour completed:
  - Remaining pseudo-states, such as shallow/deep history.

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:**
  - Brief: methods/behavioural features.
  - Reflective description of behaviour.
  - LSC concrete and abstract syntax.
  - LSC intuitive semantics.
  - Symbolic Büchi Automata (TBA) and its (accepted) language.

And What About Methods?

And What About Methods?

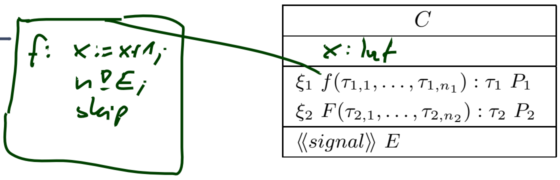
- In the current setting, the (local) state of objects is only modified by actions of transitions, which we abstract to transformers.
- In general, there are also **methods**.
- UML follows an approach to separate
  - the **interface declaration** from
  - the **implementation**.

  In C++ lingo: distinguish **declaration** and **definition** of method.
- In UML, the former is called **behavioural feature** and can (roughly) be
  - a **call interface** $f(\tau_{1_1}, \dots, \tau_{n_1}) : \tau_1$
  - a **signal name** E

C
$\xi_1\ f(\tau_{1,1}, \dots, \tau_{1,n_1}) : \tau_1\ P_1$ $\xi_2\ F(\tau_{2,1}, \dots, \tau_{2,n_2}) : \tau_2\ P_2$
$\langle\langle signal \rangle\rangle\ E$

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



Semantics:

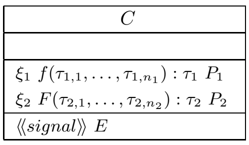
- The **implementation** of a behavioural feature can be provided by:
  - An **operation**. operations to be elements of Acts

    In our setting, we simply assume a transformer like T_f.

    It is then, e.g. clear how to admit method calls as actions on transitions: function composition of transformers (clear but tedious: non-termination).

    In a setting with Java as action language: operation is a method body.
  - The class' **state-machine** ("triggered operation").
    - Calling F with n_2 parameters for a stable instance of C creates an auxiliary event F and dispatches it (bypassing the ether).
    - Transition actions may fill in the return value.
    - On completion of the RTC step, the call returns.
    - For a non-stable instance, the caller blocks until stability is reached again.

Behavioural Features: Visibility and Properties

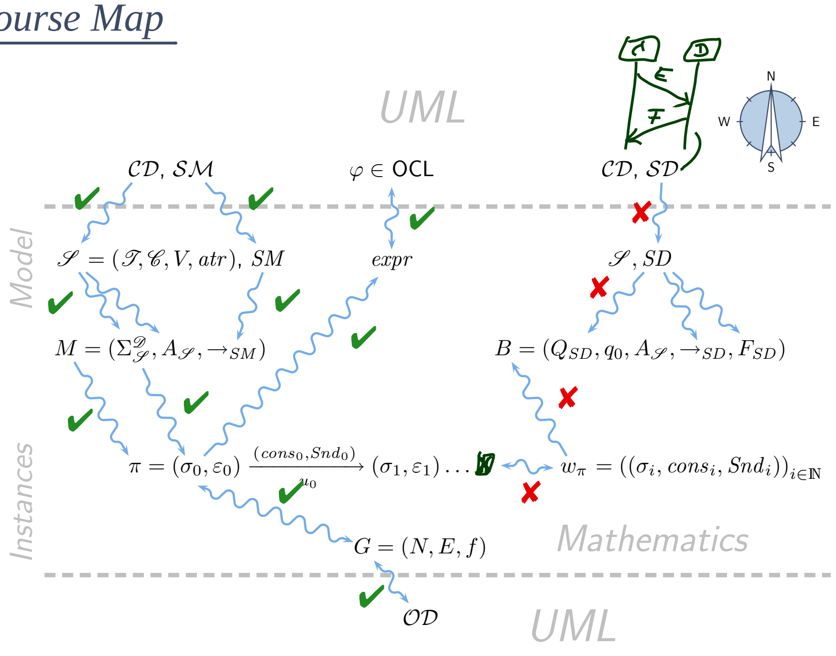


- **Visibility**:
  - Extend typing rules to sequences of actions such that a well-typed action sequence only calls visible methods.
- **Useful properties**:
  - **concurrency**
    - **concurrent** — is thread safe
    - **guarded** — some mechanism ensures/should ensure mutual exclusion
    - **sequential** — 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).

You are here.

Course Map



Motivation: Reflective, Dynamic Descriptions of Behaviour

What Can Be Purposes of Behavioural Models?

Example: Pre-Image **Image**
(the UML model is supposed to be the blue-print for a software system).

A description of behaviour could serve the following purposes:

- **Require** Behaviour. **"System definitely does this"**

  "This sequence of inserting money and requesting and getting water must be possible."
  (Otherwise the software for the vending machine is completely broken.)

- **Allow** Behaviour. **"System does subset of this"**

  "After inserting money and choosing a drink, the drink is dispensed (if in stock)."
  (If the implementation insists on taking the money first, that's a fair choice.)

- **Forbid** Behaviour. **"System never does this"**

  "This sequence of getting both, a water and all money back, must not be possible." (Otherwise the software is broken.)

Note: the latter two are trivially satisfied by doing nothing...

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!

Constructive UML

UML provides two visual formalisms for constructive description of behaviours:

- **Activity Diagrams**
- **State-Machine Diagrams**

We (exemplary) focus on State-Machines because

- somehow "practice proven" (in different flavours),
- prevalent in embedded systems community,
- indicated useful by [Dobing and Parsons, 2006] survey, and
- Activity Diagram's intuition changed from transition-system-like to petri-net-like...

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 $\pi \models \vartheta$ and $\pi \not\models \vartheta$.

OCL as Reflective Description of Certain Properties

- **invariants**:

the i-th (σ,ε)-pair in π

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

- **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.

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"**

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

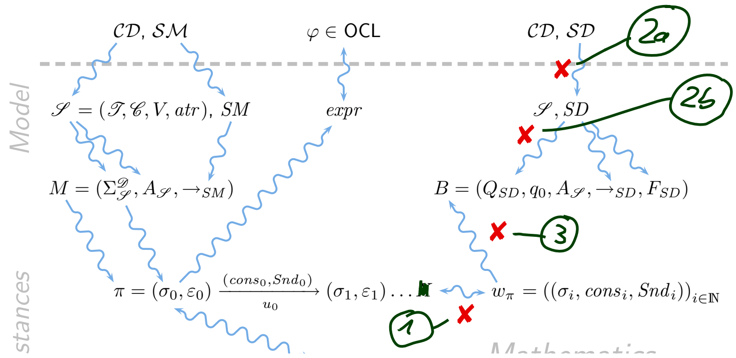
But: what is "$\models$" and what is "ϑ"?

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



Words over Signature

Definition. Let $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ be a signature and $\mathscr{D}$ a structure of $\mathscr{S}$. A **word** over $\mathscr{S}$ and $\mathscr{D}$ is an infinite sequence

$$(\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}_0} \in \left(\Sigma_{\mathscr{S}}^{\mathscr{D}} \times 2^{\mathscr{D}(\mathscr{C}) \times Evs(\mathscr{E}, \mathscr{D})} \times 2^{\mathscr{D}(\mathscr{C}) \times Evs(\mathscr{E}, \mathscr{D}) \times \mathscr{D}(\mathscr{C})} \right)^{\omega}.$$

sender event receiver

infinite sequences

The Language of a Model

Recall: A UML model $\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ and a structure $\mathscr{D}$ denotes a set $[\![\mathcal{M}]\!]$ of (initial and consecutive) **computations** of the form

$$(\sigma_0, \varepsilon_0) \xrightarrow{a_0} (\sigma_1, \varepsilon_1) \xrightarrow{a_1} (\sigma_2, \varepsilon_2) \xrightarrow{a_2} \dots \text{ where}$$

$$a_i = (cons_i, Snd_i, u_i) \in \underbrace{2^{\mathscr{D}(\mathscr{C}) \times Evs(\mathscr{E}, \mathscr{D})} \times 2^{\mathscr{D}(\mathscr{C}) \times Evs(\mathscr{E}, \mathscr{D}) \times \mathscr{D}(\mathscr{C})}}_{=:\tilde{A}} \times \mathscr{D}(\mathscr{C}).$$

For the connection between models and interactions, we **disregard** the configuration of **the ether** and **who** made the step, and define as follows:

Definition. Let $\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ be a UML model and $\mathscr{D}$ a structure. Then

$$\mathcal{L}(\mathcal{M}) := \{(\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}_0} \in (\Sigma_{\mathscr{S}}^{\mathscr{D}} \times \tilde{A})^{\omega} \mid \exists\, (\varepsilon_i, u_i)_{i \in \mathbb{N}_0} : (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots \in [\![\mathcal{M}]\!]\}$$

is the **language** of $\mathcal{M}$.

Model Consistency wrt. Interaction

- We assume that the set of interactions $\mathscr{I}$ is partitioned into two (possibly empty) sets of **universal** and **existential** interactions, i.e.

$$\mathscr{I} = \mathscr{I}_\forall \,\dot\cup\, \mathscr{I}_\exists.$$

Definition. A model

$$\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD}, \mathscr{I})$$

is called **consistent** (more precise: the constructive description of behaviour is consistent with the reflective one) if and only if

$$\forall \mathcal{I} \in \mathscr{I}_\forall : \mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{I})$$

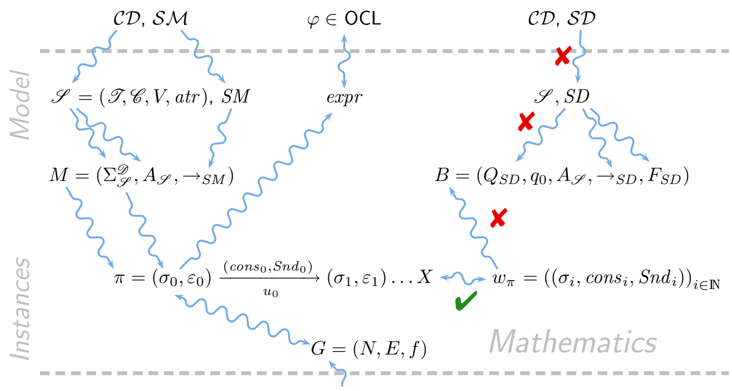
and

$$\forall \mathcal{I} \in \mathscr{I}_\exists : \mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\mathcal{I}) \neq \emptyset.$$

Interactions: Plan

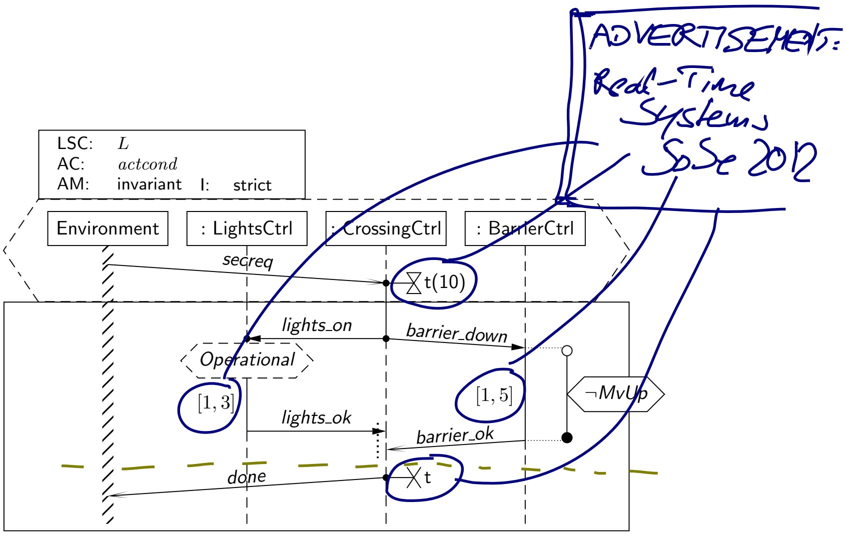
- In the following, we consider **Sequence Diagrams** as **interaction** $\mathcal{I}$,
- more precisely: **Live Sequence Charts** [Damm and Harel, 2001].
- We define the **language** $\mathcal{L}(\mathcal{I})$ of an LSC — via Büchi automata.
- Then (conceptually) $\pi \models \vartheta$ if and only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

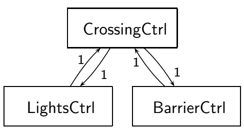
Why LSC, relation LSCs/UML SDs, other kinds of interactions: **later**.

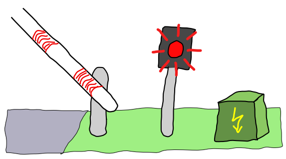


Live Sequence Charts — Concrete Syntax

Example

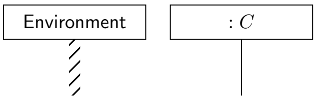






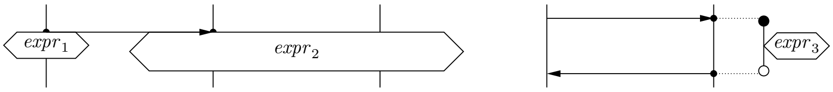
Building Blocks

- **Instance Lines:**



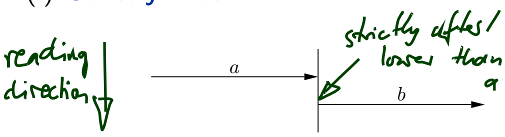
- **Messages:** (asynchronous or synchronous/instantaneous)

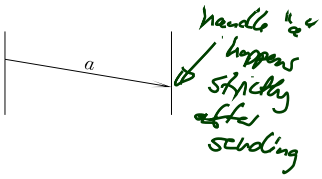
- **Conditions and Local Invariants:** $(expr_1, expr_2, expr_3 \in Expr_{\mathscr{S}})$



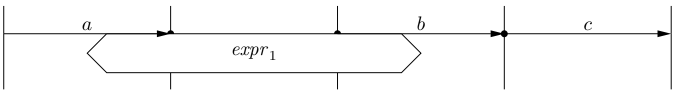
Intuitive Semantics: A Partial Order on Simclasses

(i) **Strictly After:**

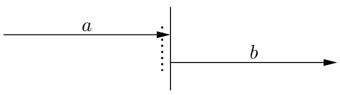




(ii) **Simultaneously:** (simultaneous region)

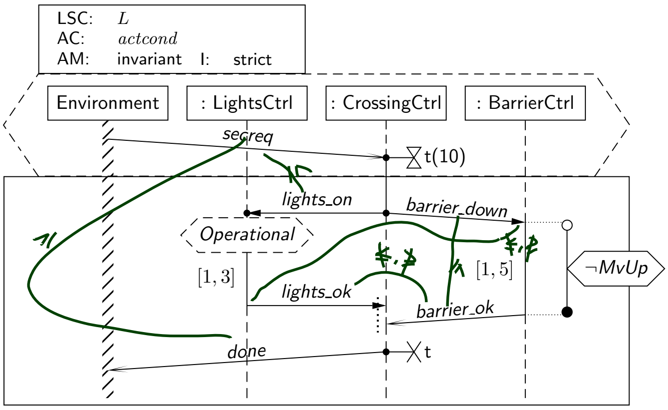


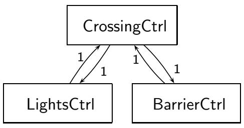
(iii) **Explicitly Unordered:** (co-region)

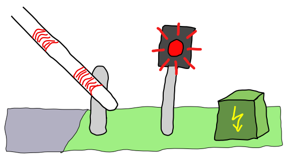


Intuition: A computation path **violates** an LSC if the occurrence of some events doesn't adhere to partial order obtained as the **transitive closure** of (i) to (iii).

Example: Partial Order Requirements





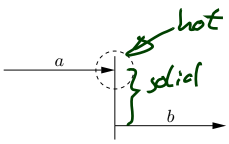
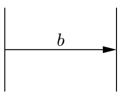
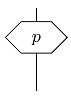
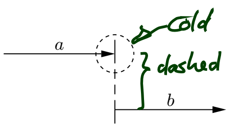
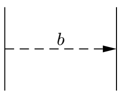
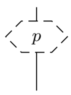


LSC Specialty: Modes

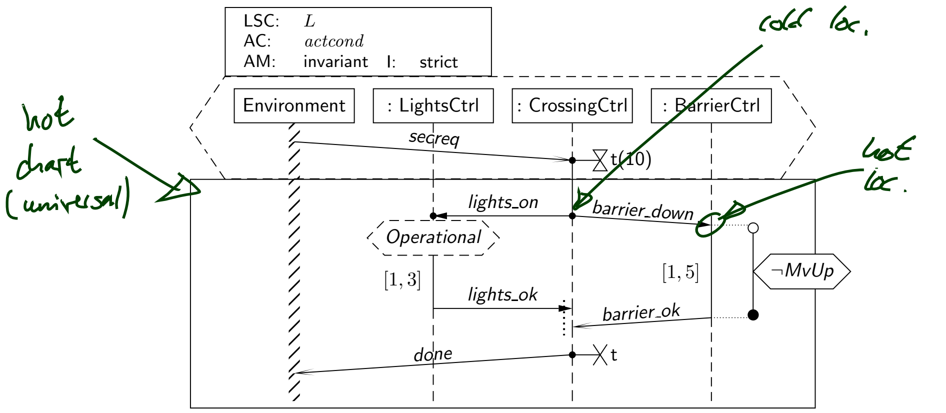
With LSCs,

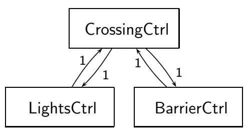
- whole charts,
- locations, and
- elements

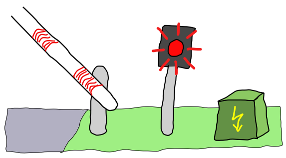
have a **mode** — one of **hot** or **cold** (graphically indicated by outline).

	chart	location	message	condition/ local inv.
hot:				
cold:				
	always vs. at least once	must vs. may progress	mustn't vs. may get lost	necessary vs. legal exit

Example: Modes



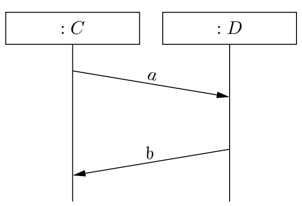


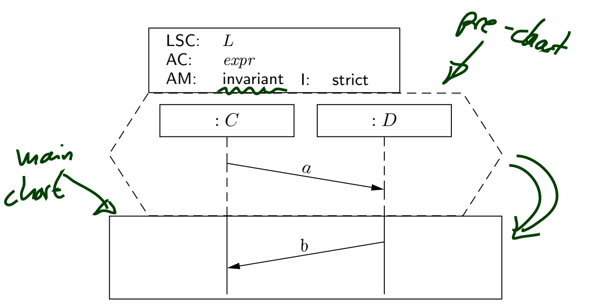


LSC Specialty: Activation

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.



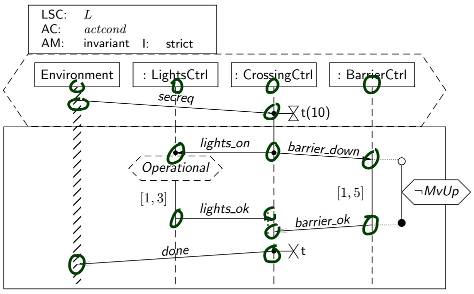


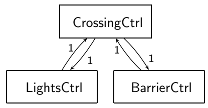
Intuition: (universal case)

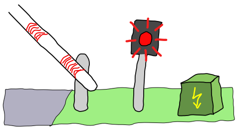
- given a computation π, **whenever** $expr$ holds in a configuration $(\sigma_i, \varepsilon_i)$ of ξ
  - which is initial, i.e. $k = 0$, or (AM $=$ *initial*)
  - whose k is not further restricted, (AM $=$ *invariant*)

  and if the pre-chart is observed from k to $k + n$,
  then the main-chart has to follow from $k + n + 1$. (otherwise system not consistent with interaction)

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 sequence diagram 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 replies with 'done' to the environment.

Live Sequence Charts — Abstract Syntax

LSC Body: Abstract Syntax

main. or pre-chart

Let $\Theta = \{\text{hot}, \text{cold}\}$. An **LSC body** is a tuple

$$(I, (\mathscr{L}, \preceq), \sim, \mathscr{S}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv})$$

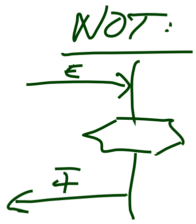
where

- I is a finite set of **instance lines**,
- $(\mathscr{L}, \preceq)$ is a finite, non-empty, partially ordered set of **locations**, each $l \in \mathscr{L}$ is associated with a temperature $\theta(l) \in \Theta$ and an instance line $i_l \in I$,
- $\sim \subseteq \mathscr{L} \times \mathscr{L}$ is an **equivalence relation** on locations, the **simultaneity** relation,
- $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ is a signature,
- $\mathsf{Msg} \subseteq \mathscr{L} \times \mathscr{E} \times \mathscr{L}$ is a set of **asynchronous messages** with $(l, b, l') \in \mathsf{Msg}$ only if $l \not\preceq l'$,

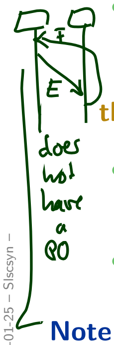
  Not: **instantaneous messages** — could be linked to method/operation calls.
- $\mathsf{Cond} \subseteq (2^{\mathscr{L}} \setminus \emptyset) \times Expr_{\mathscr{S}} \times \Theta$ is a set of **conditions** with $(L, expr, \theta) \in \mathsf{Cond}$ only if $l \sim l'$ for all $l, l' \in L$,
- $\mathsf{LocInv} \subseteq \mathscr{L} \times \{\circ, \bullet\} \times Expr_{\mathscr{S}} \times \Theta \times \mathscr{L} \times \{\circ, \bullet\}$ is a set of **local invariants**,

Well-Formedness

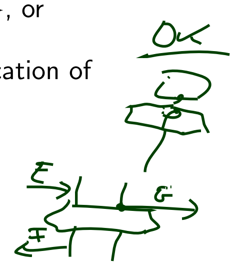
Bondedness/no floating conditions: (could be relaxed a little if we wanted to)



- For each location $l \in \mathscr{L}$, **if** l is the location of
  - a **condition**, i.e.
    $$\exists (L, expr, \theta) \in \mathsf{Cond} : l \in L,$$
  - a **local invariant**, i.e.
    $$\exists (l_1, i_1, expr, \theta, l_2, i_2) \in \mathsf{LocInv} : l \in \{l_1, l_2\}, \text{ or}$$

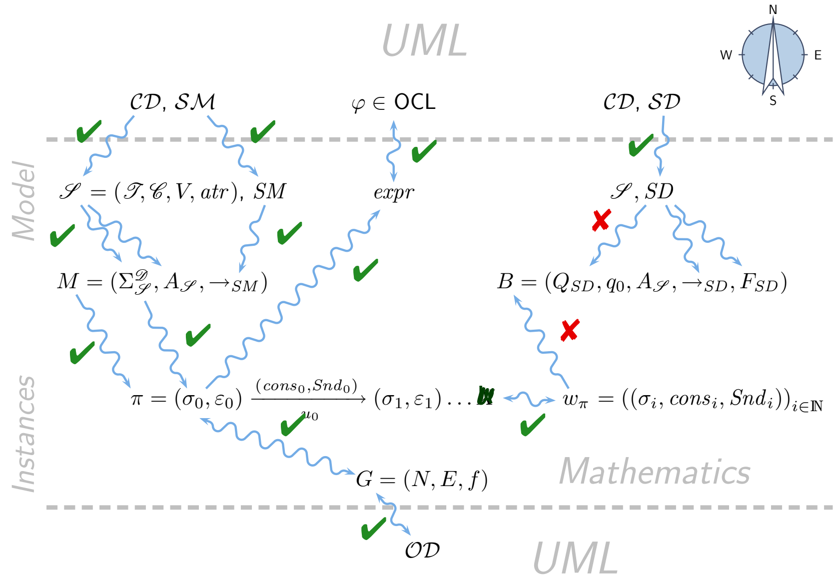
  

  then there is a location l' **equivalent** to l which is the location of
  - a **message**, i.e.
    $$\exists (l_1, b, l_2) \in \mathsf{Msg} : l \in \{l_1, l_2\}, \text{ or}$$
  - an **instance head**, i.e. l' is minimal wrt. $\preceq$.



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).

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.

[Dobing and Parsons, 2006] Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, 49(5):109–114.

[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.