Software Design, Modelling and Analysis in UML

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

2014-01-27

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

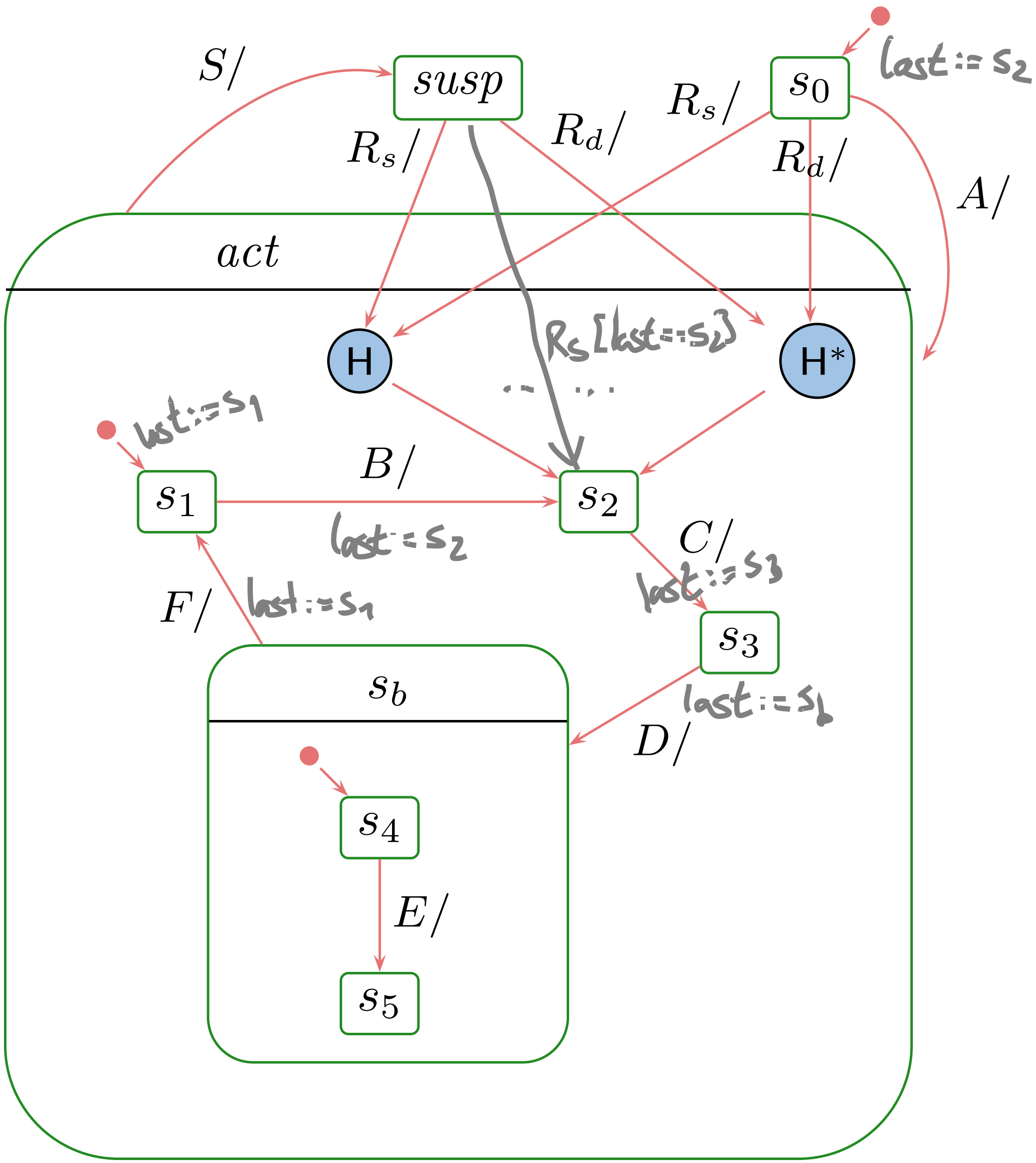
- Hierarchical State Machines
- **Later**: active vs. passive; behavioural feature (aka. methods).

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 after creation)

- R_s?

  s_0, s_2

- R_d?

  s_0, s_2

- A, B, C, S, R_s?

  s_0, s_1, s_2, s_3, susp, $\underline{s_3}$

- $A, B, \overset{C}{,} S, R_d$?

  s_0, s_1, s_2, s_3, susp, s_3

- $A, B, C, D, E, \overset{S}{,} R_s$?

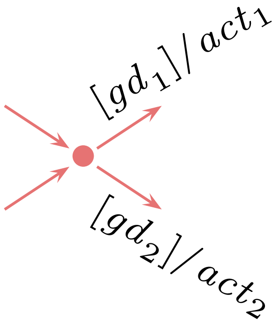
  s_0, s_1, s_2, s_4, s_5, susp, s_4 "shallow"

- $A, B, C, \overset{E, S}{D,} R_d$?

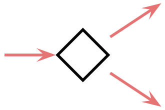
  s_0, s_1, s_2, s_4, s_5, susp, s_5 "deep"

Junction and Choice

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



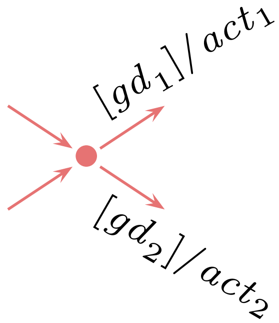
- Choice: (**"dynamic conditional branch"**)



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

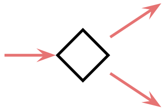
Junction and Choice

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



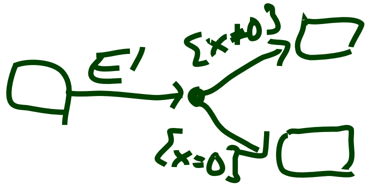
  - **good**: abbreviation
  - unfolds to so many similar transitions with different guards, the unfolded transitions are then checked for enabledness
  - at best, start with trigger, branch into conditions, then apply actions

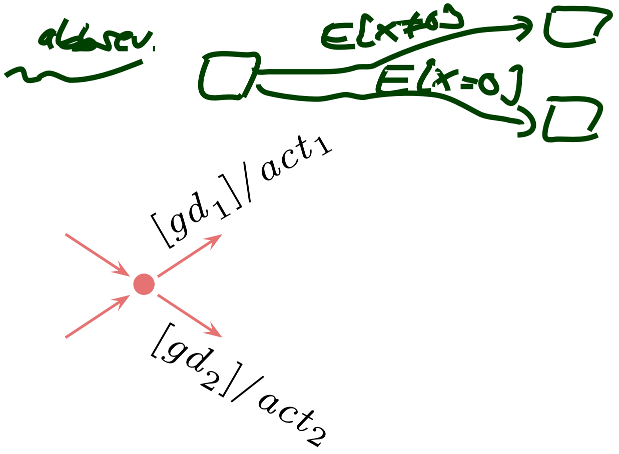
- Choice: (**"dynamic conditional branch"**)



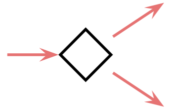
Note: not so sure about naming and symbols, e.g.,
I'd guessed it was just the other way round...

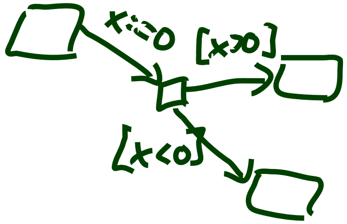
Junction and Choice





- 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

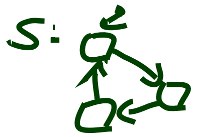




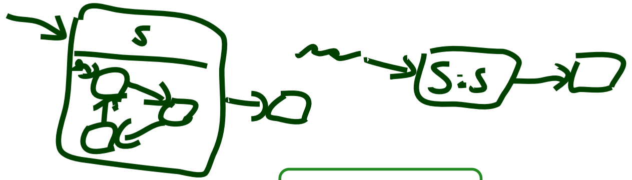
- Choice: (**"dynamic conditional branch"**)
  - **evil**: may get stuck
  - enters the transition **without knowing** whether there's an enabled path
  - at best, use "else" and convince yourself that it cannot get stuck
  - maybe even better: **avoid**

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.)



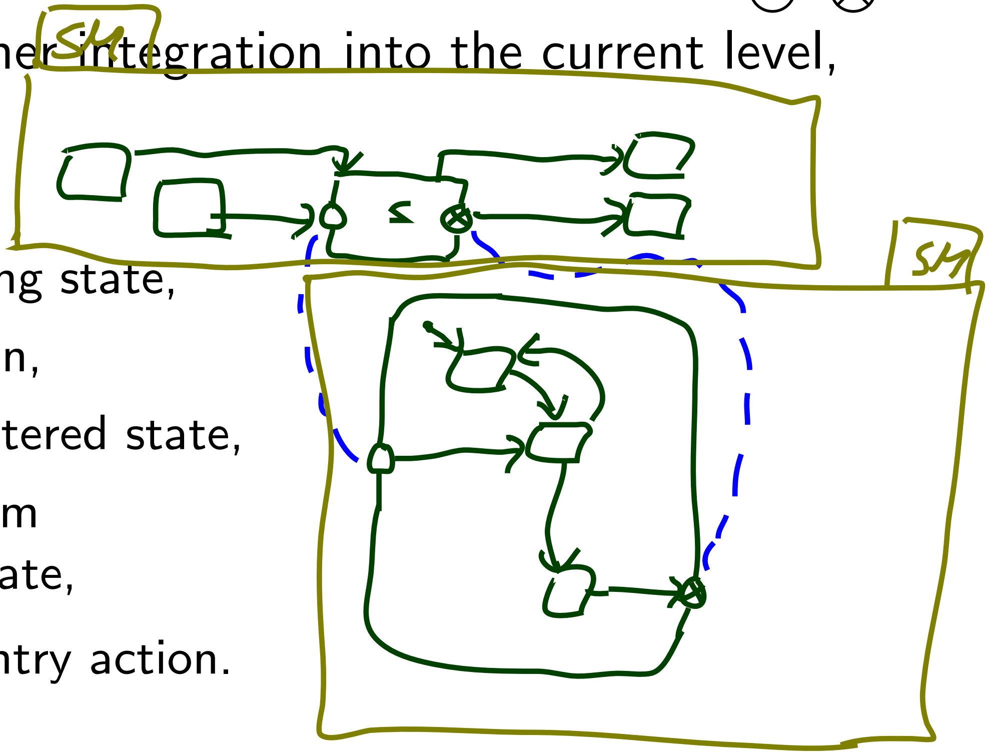
- Can even be taken from a different state-machine for re-use.

$S : s$

- **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** 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.

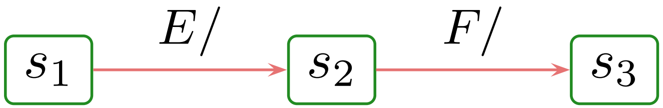
Deferred Events in State-Machines

Deferred Events: Idea

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

The idea is as follows:

- Consider the following state machine:



- 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.

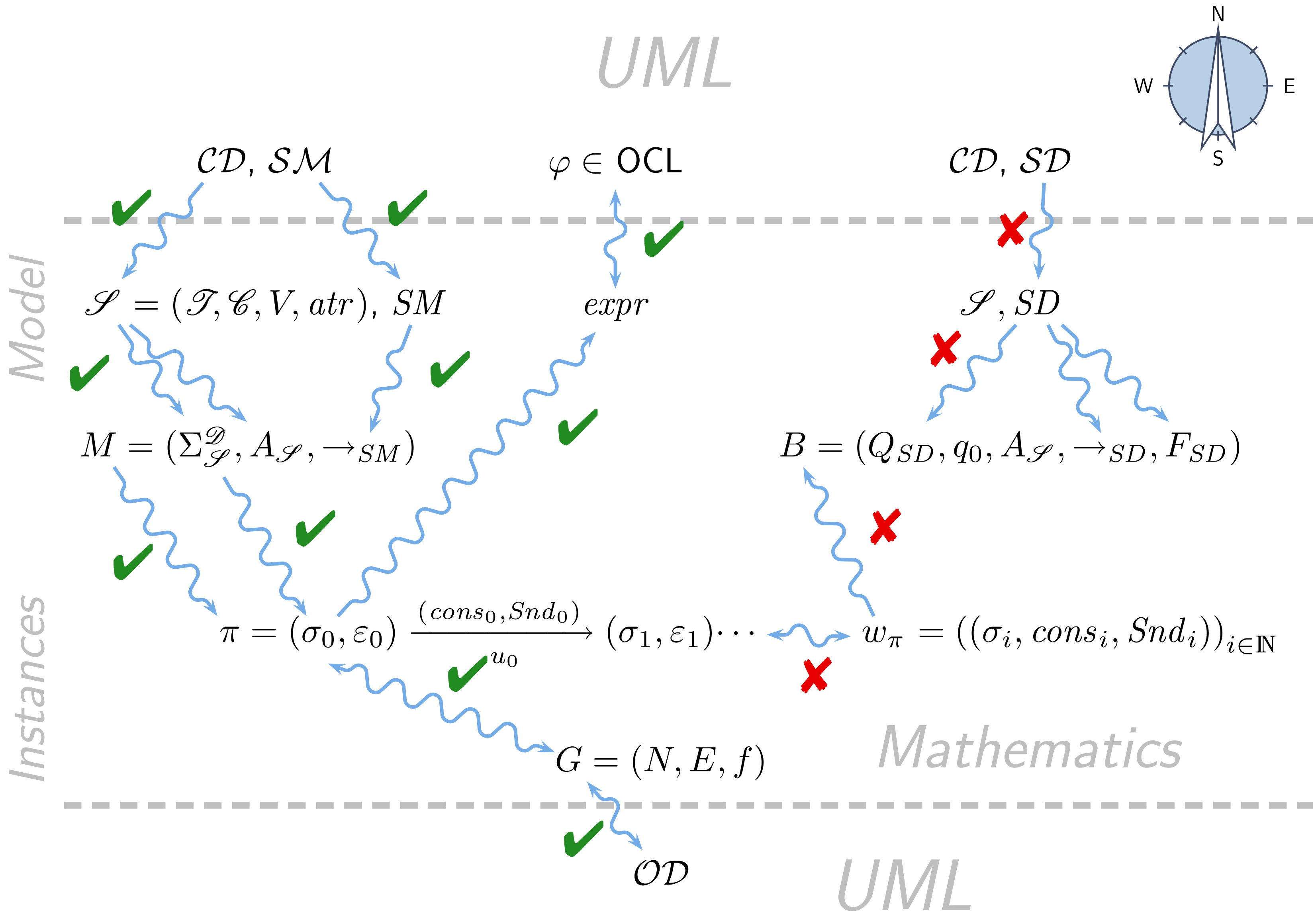
- The **semantics** is a bit intricate, something like
  - if an event E is dispatched,
  - and there is no transition enabled to consume E,
  - and E is in the deferred set of the current state configuration,
  - then stuff E into some "deferred events space" of the object, (e.g. into the ether ($=$ extend ε) or into the local state of the object ($=$ extend σ))
  - 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.

Course Map



UML

$\mathcal{CD}$, $\mathcal{SM}$

$\varphi \in$ OCL

$\mathcal{CD}$, $\mathcal{SD}$

Model

$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$, SM

$expr$

$\mathscr{S}, SD$

$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \to_{SM})$

$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \to_{SD}, F_{SD})$

Instances

$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$

$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$

$G = (N, E, f)$

Mathematics

$\mathcal{OD}$

UML

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** $(S, \rightarrow, S_0)$ 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 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.

OCL as Reflective Description of Certain Properties

the i-th (σ, ε)-pair in π

- **invariants**:

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

- **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

- 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"?

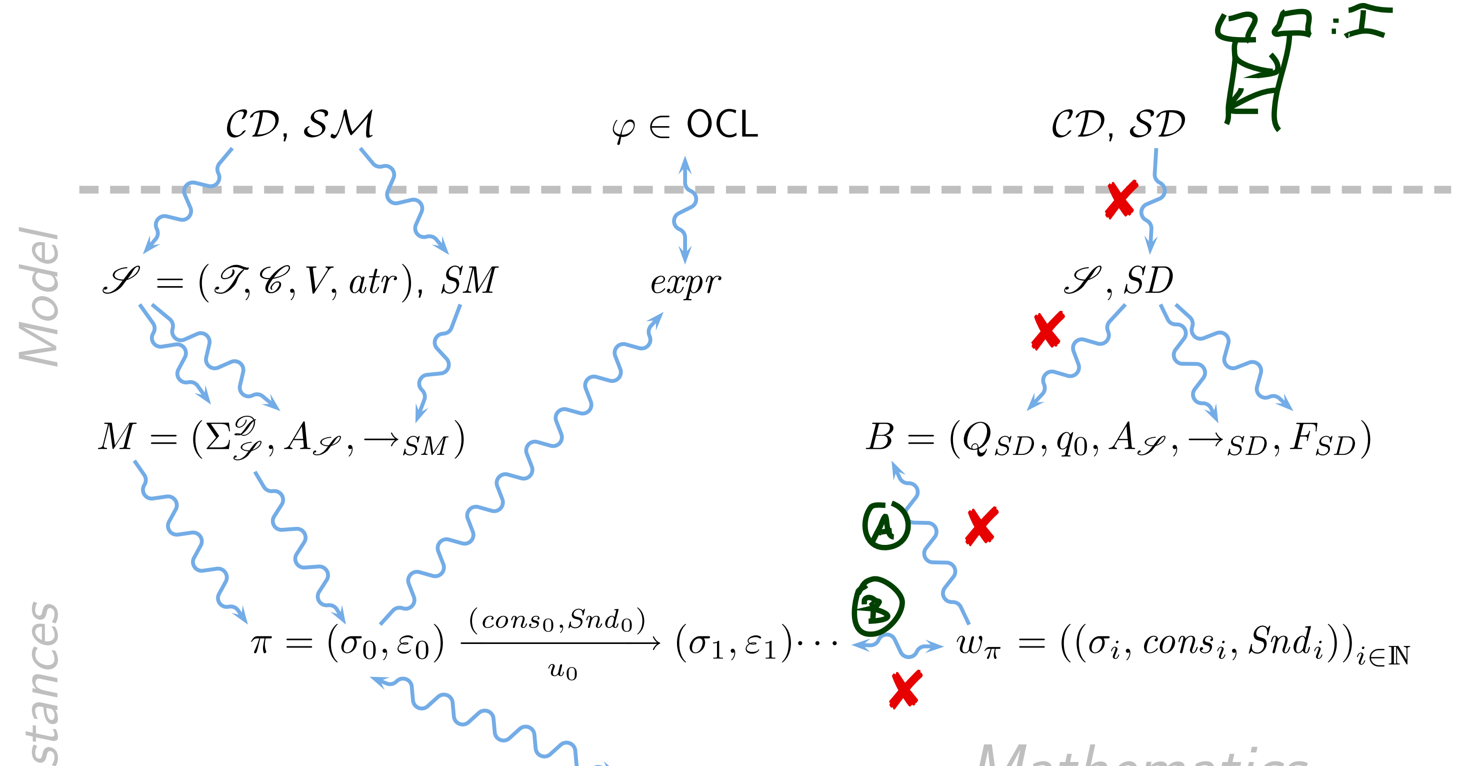
Interactions: Problem and Plan

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

Plan:

A
- Define the **language** $\mathcal{L}(\mathcal{I})$ of an **interaction** $\mathcal{I}$ — via Büchi automata.

B
- 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 $w_\pi \in \mathcal{L}(\mathcal{I})$.

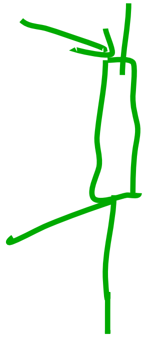


:$\mathcal{I}$
$\mathcal{CD}$, $\mathcal{SM}$
$\varphi \in$ OCL
$\mathcal{CD}$, $\mathcal{SD}$
Model
$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$, SM
$expr$
$\mathscr{S}$, SD
$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \to_{SM})$
$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \to_{SD}, F_{SD})$
A
B
$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$
$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$
stances
Mathematics

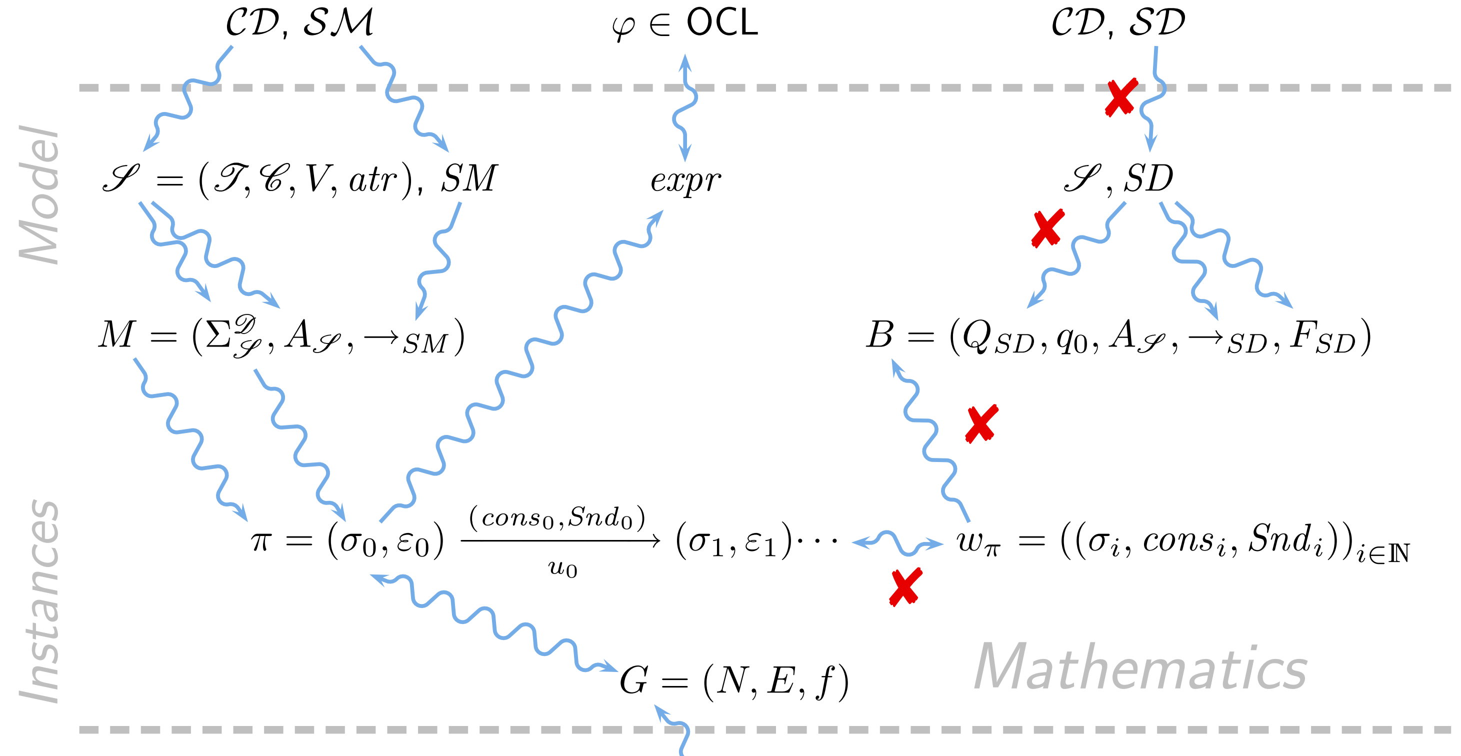
Interactions: Plan

- 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 $w_\pi \in \mathcal{L}(\mathcal{I})$.

without "rectangles":



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

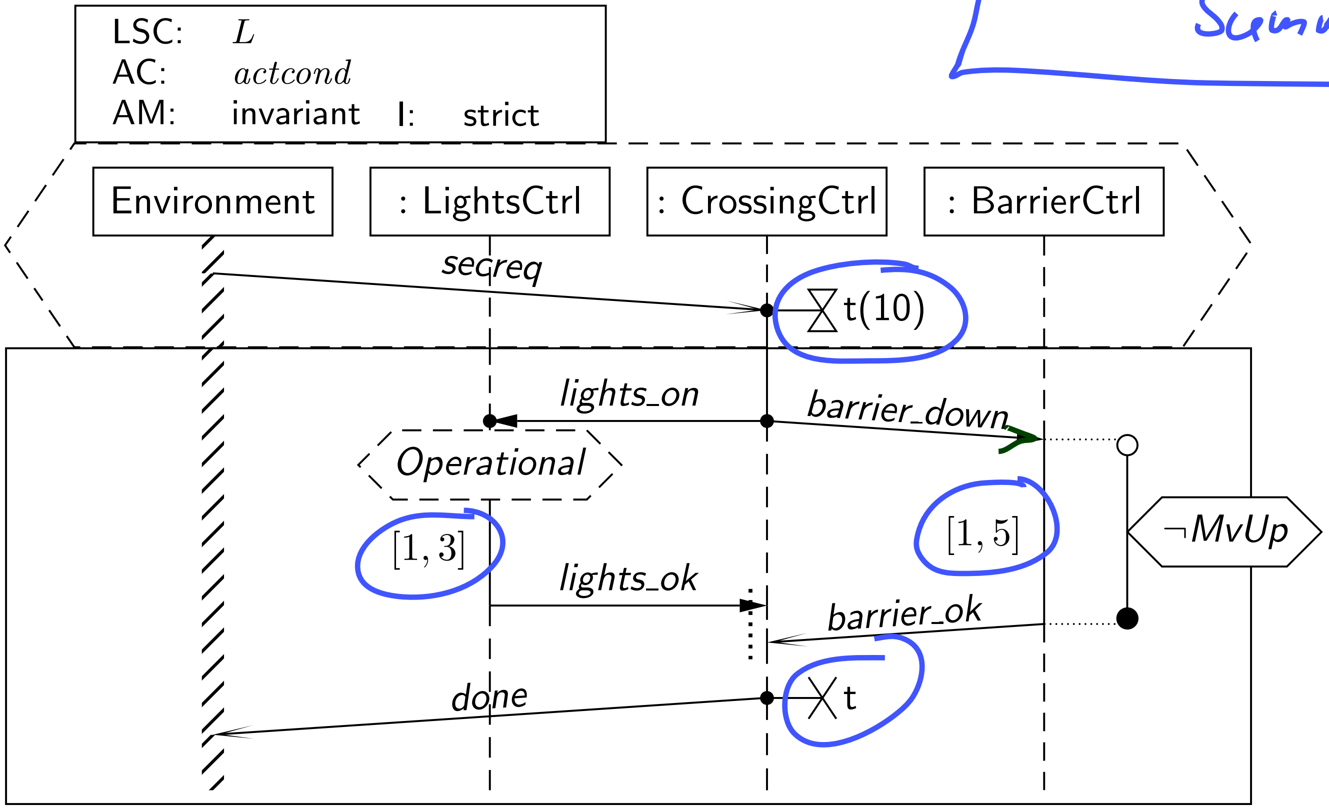


$\mathcal{CD}$, $\mathcal{SM}$
$\varphi \in$ OCL
$\mathcal{CD}$, $\mathcal{SD}$
Model
$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$, SM
$expr$
$\mathscr{S}$, SD
$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \to_{SM})$
$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \to_{SD}, F_{SD})$
$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$
$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$
Instances
$G = (N, E, f)$
Mathematics

Live Sequence Charts — Concrete Syntax

Example

ADVERTISEMENT:
Lecture
Real-Time Systems
Summer 2014

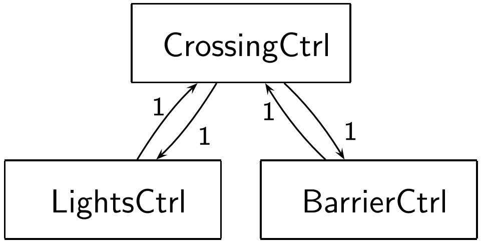


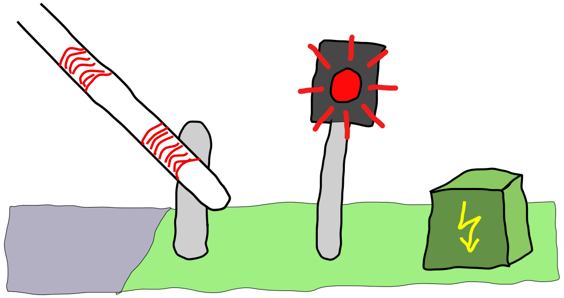
CD

«signal»
lights_on

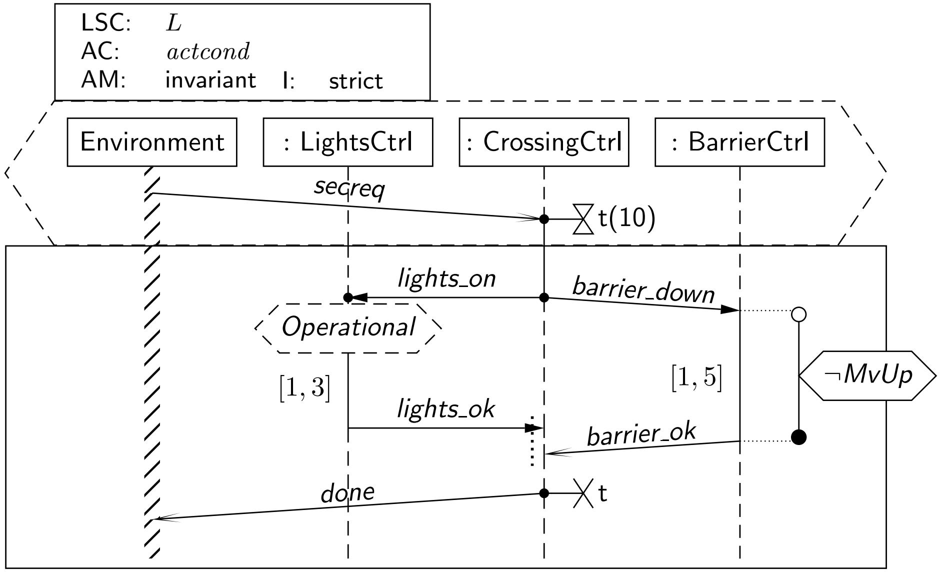
«signal»
secreq

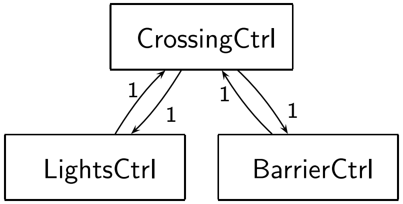
!

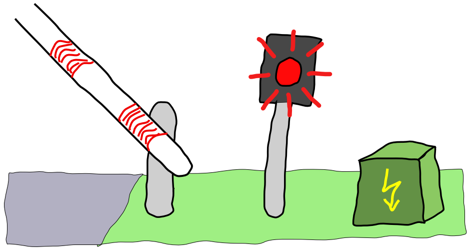




Example: What Is Required?

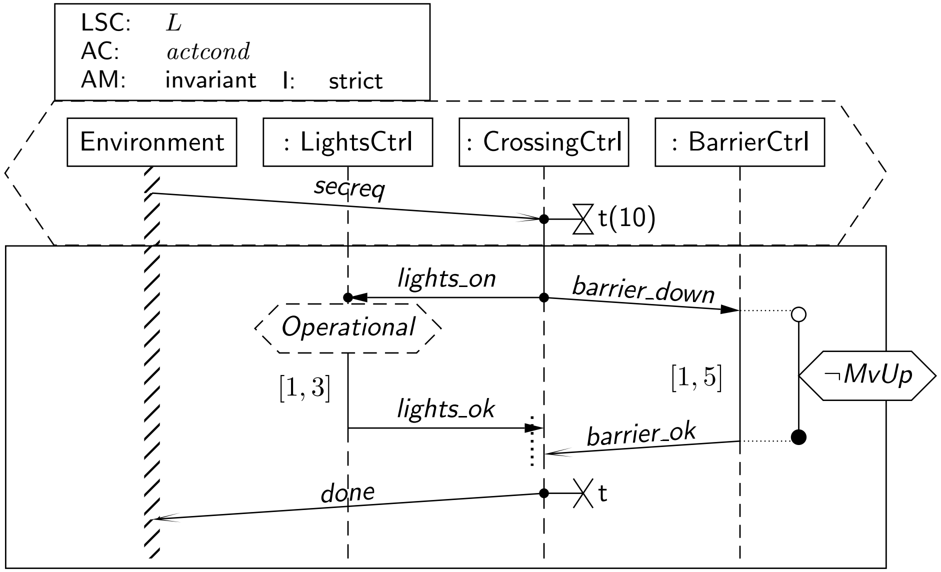


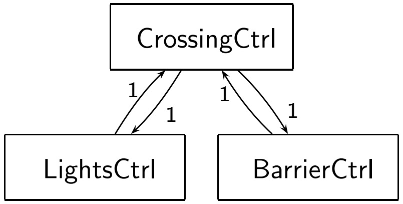


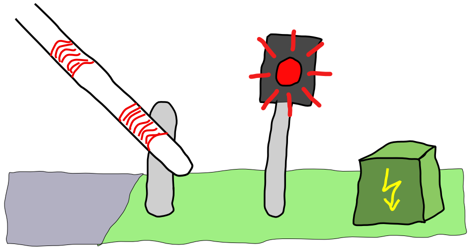


- **Whenever** the CrossingCtrl has consumed a 'secreq' event
- **then** it shall finally send 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl **is not** 'operational' when receiving that event,
  the rest of this scenario doesn't apply; maybe there's another LSC for that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

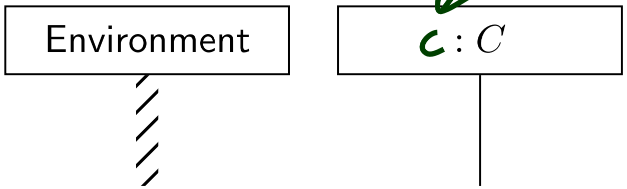




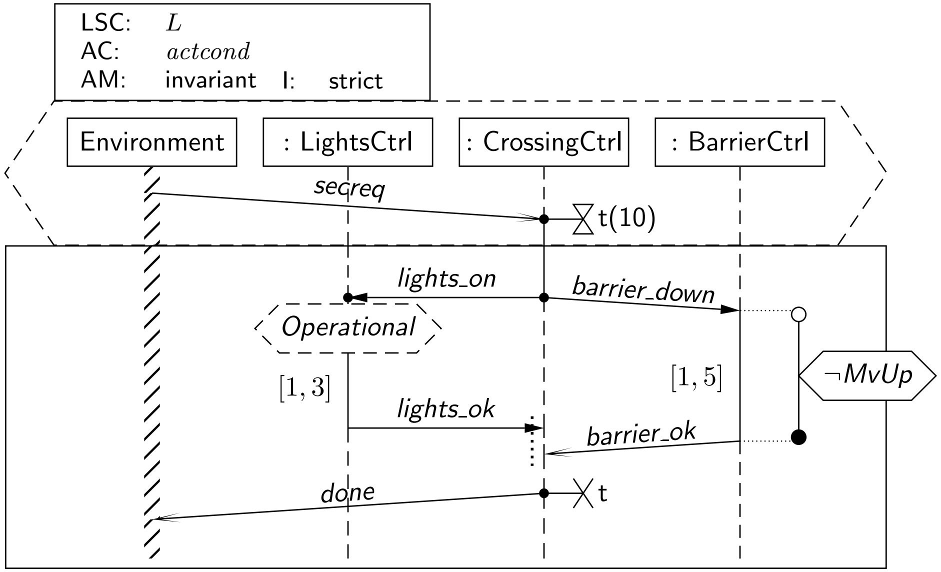


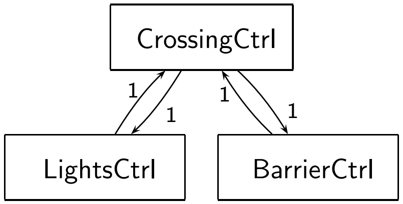
- **Instance Lines:**

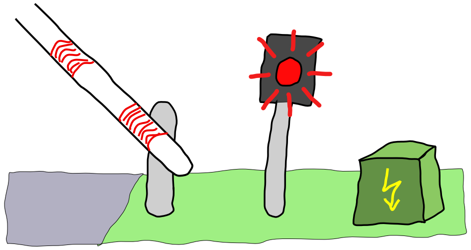
optional: logical variables



Building Blocks

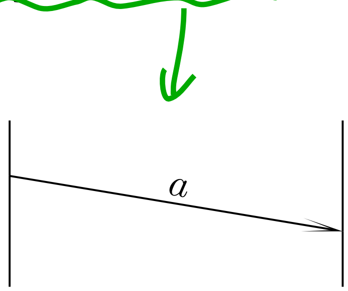


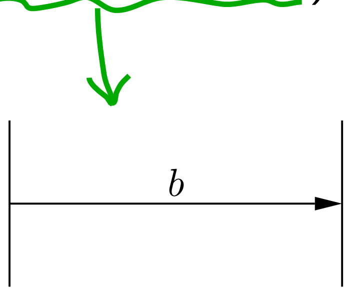


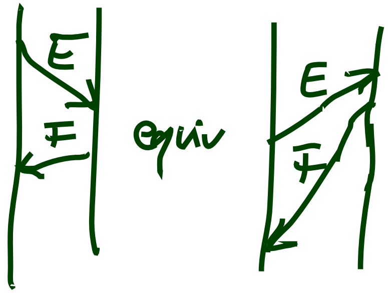


Note: angle of skewed msgs. does not matter

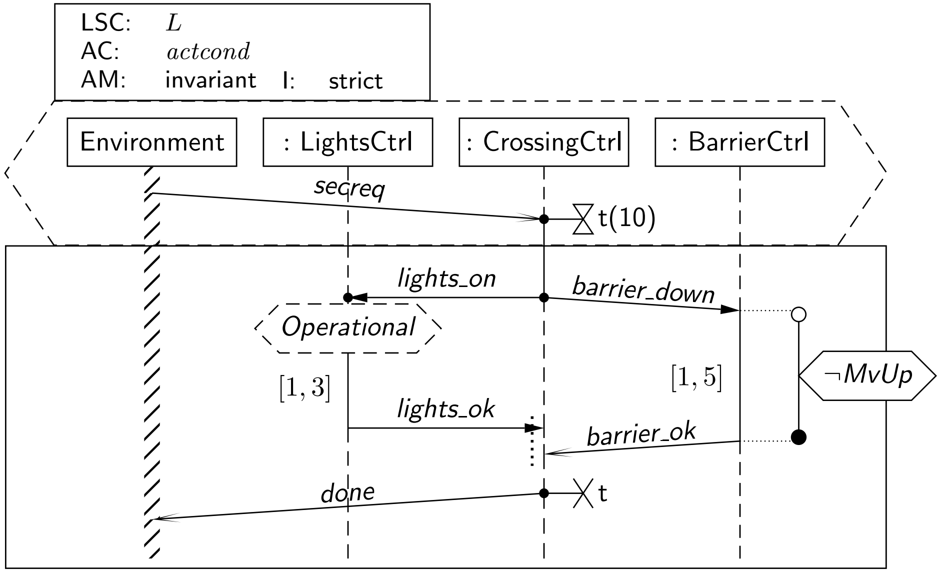
- **Messages:** (asynchronous or synchronous/instantaneous)

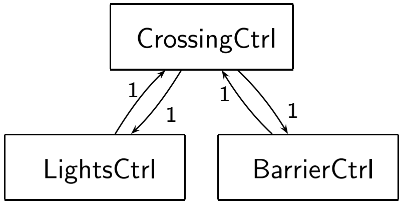


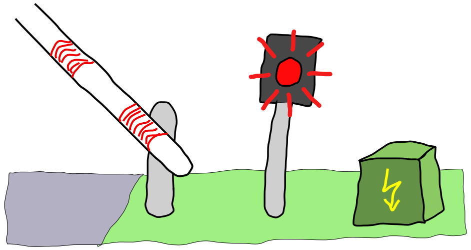




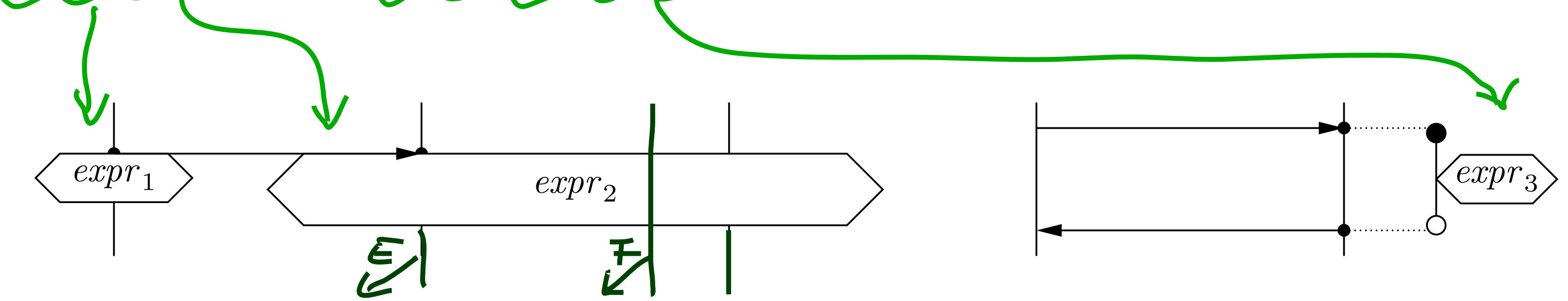
Building Blocks





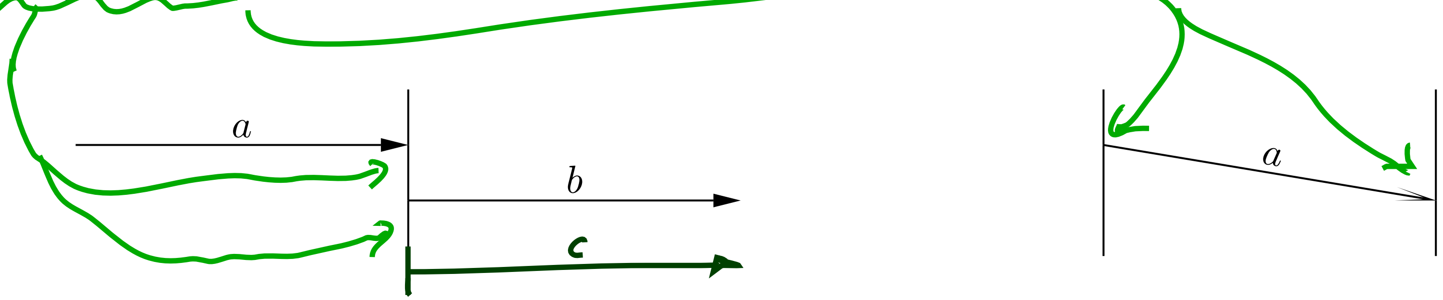


- **Conditions and Local Invariants:** $(expr_1, expr_2, expr_3 \in Expr_{\mathscr{S}})$

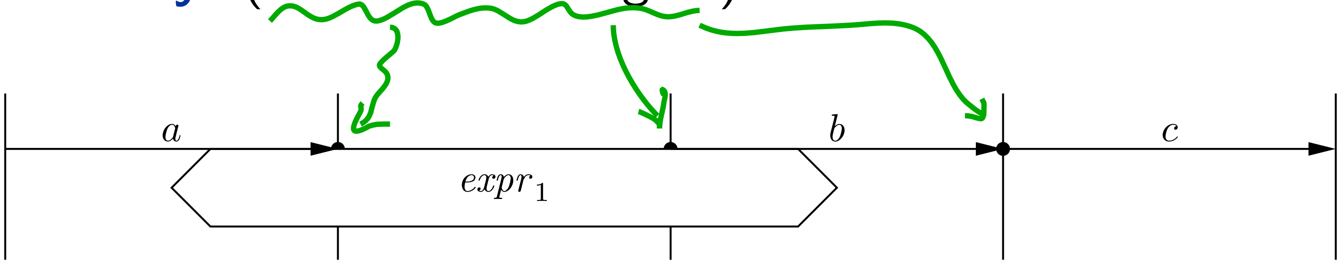


Intuitive Semantics: A Partial Order on Simclasses

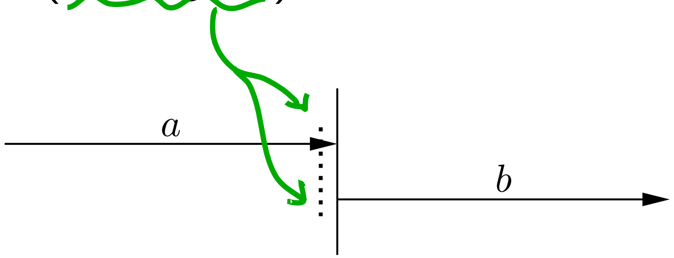
(i) **Strictly After:**



(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).

LSC Specialty: Modes

With LSCs,

- whole charts,
- locations, and
- elements

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

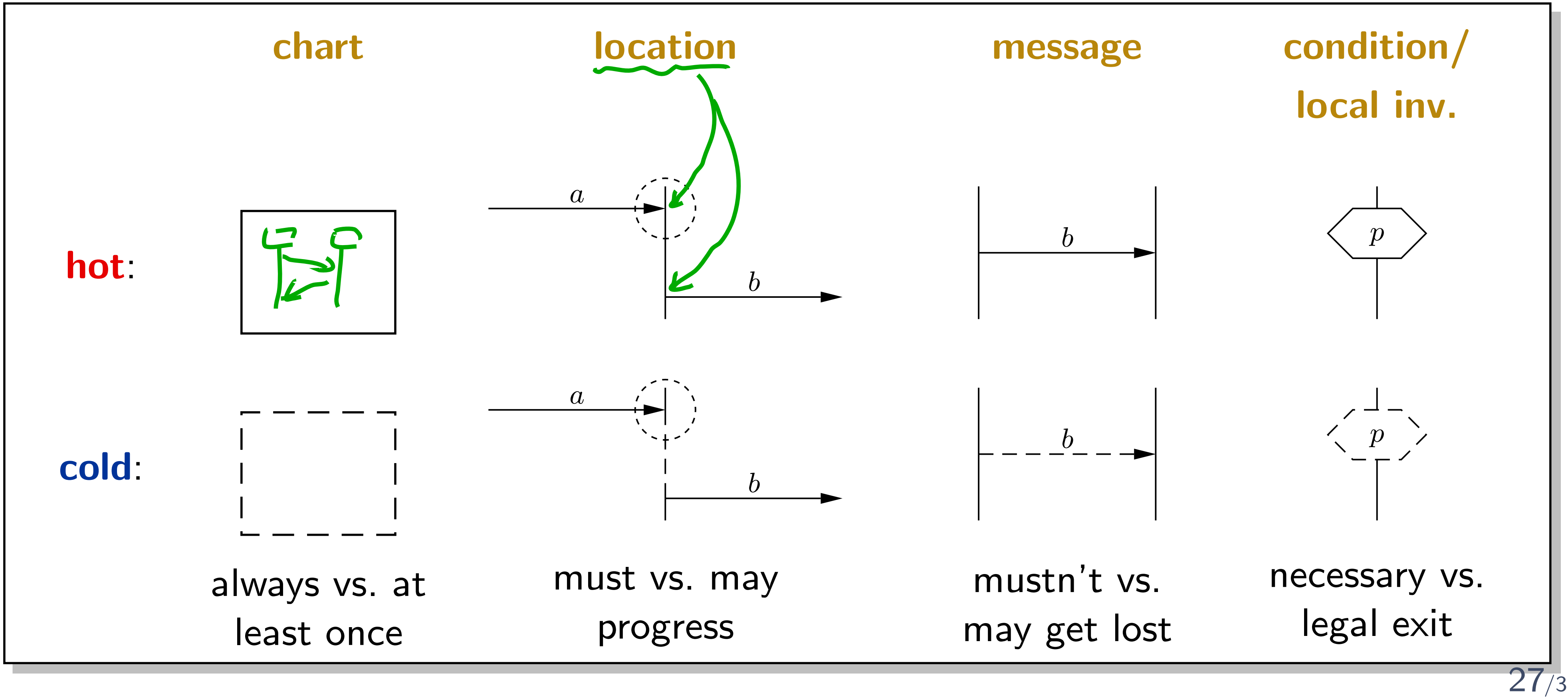
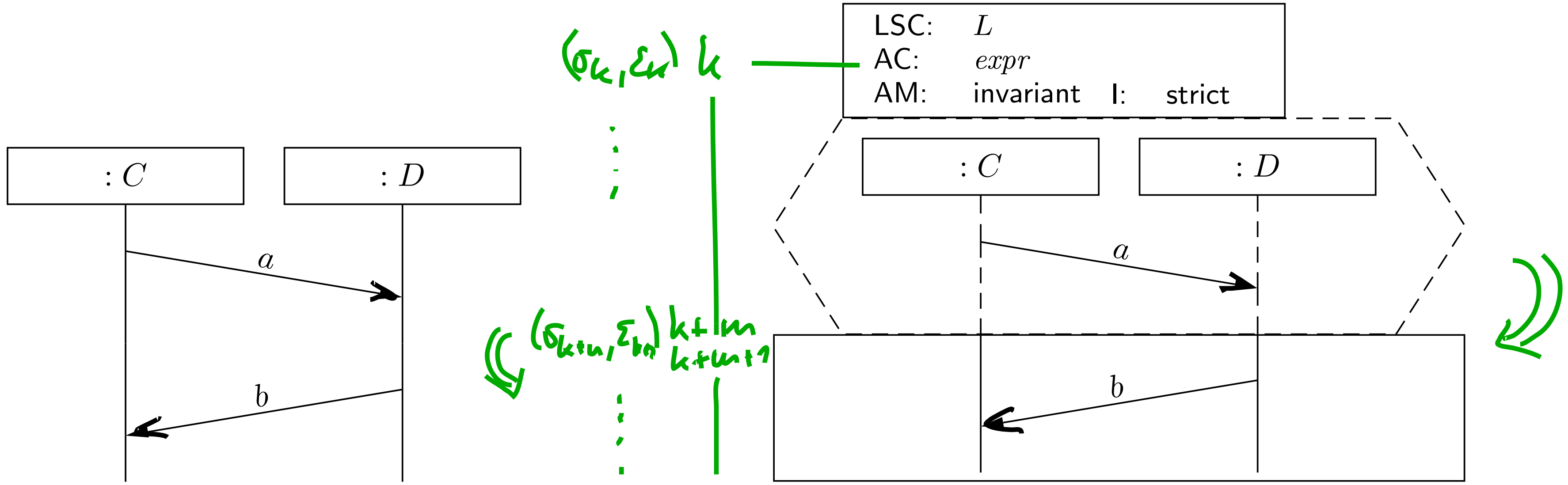


	chart	location	message	condition/ local inv.
hot:				
cold:				
	always vs. at least once	must vs. may progress	mustn't vs. may get lost	necessary vs. legal exit

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.



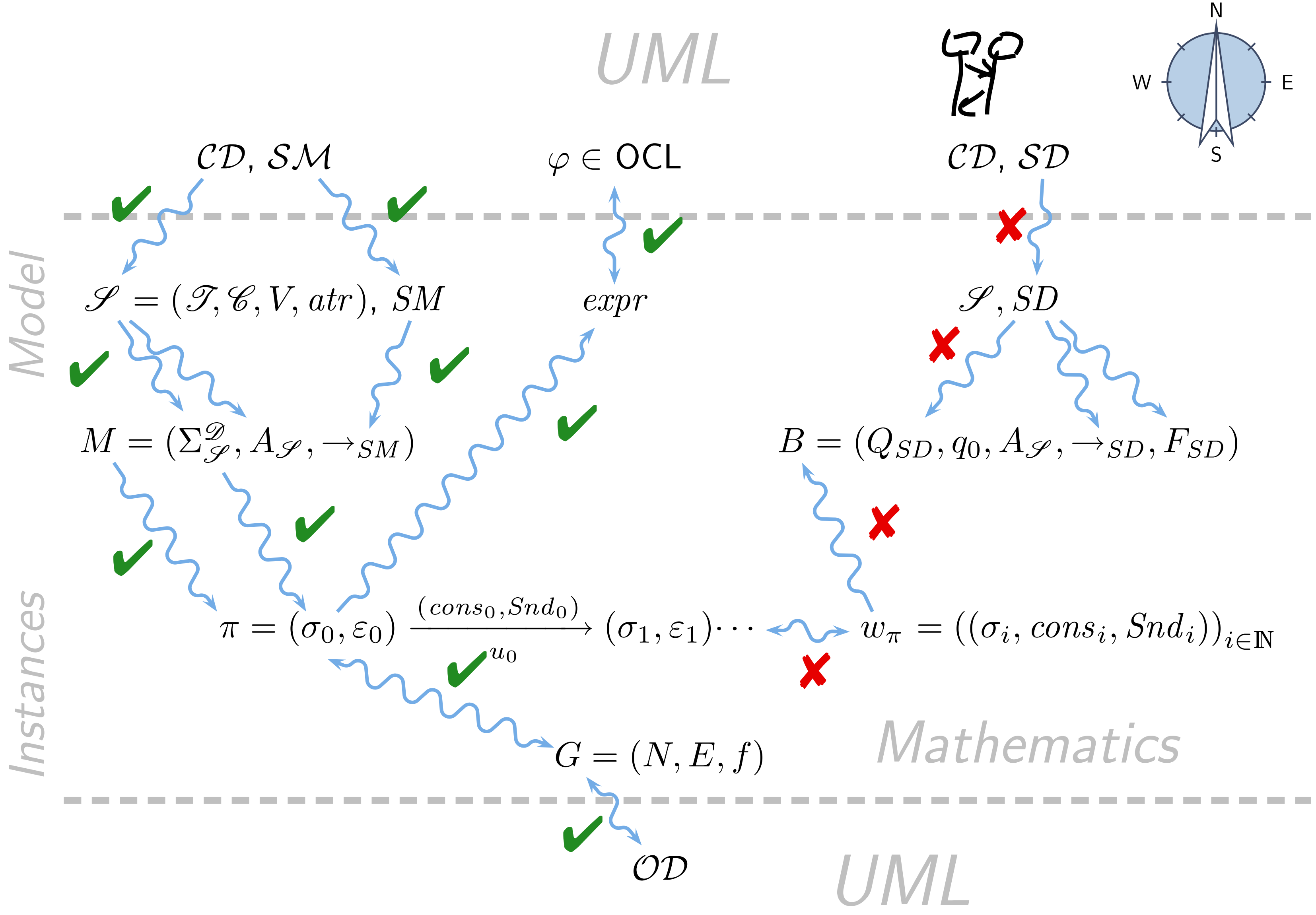
Intuition: (universal case)

- given a computation π, **whenever** $expr$ holds in a configuration $(\sigma_k, \varepsilon_k)$ of 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 $k + m$

  then the main-chart has to follow from $k + m + 1$.

Course Map



UML

$\mathcal{CD}$, $\mathcal{SM}$

$\varphi \in$ OCL

$\mathcal{CD}$, $\mathcal{SD}$

N W E S

Model

$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$, SM

$expr$

$\mathscr{S}$, SD

$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \rightarrow_{SM})$

$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \rightarrow_{SD}, F_{SD})$

Instances

$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$

$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$

$G = (N, E, f)$

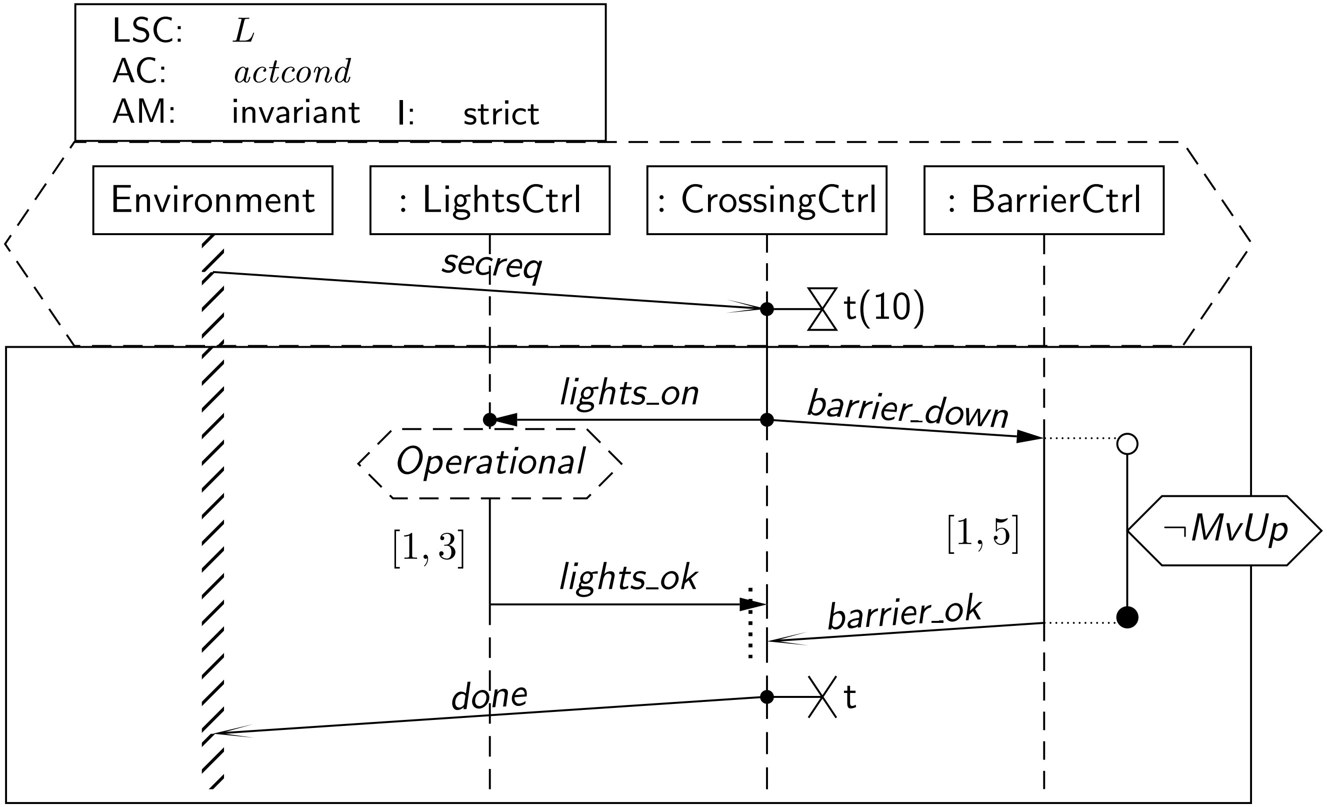
Mathematics

$\mathcal{OD}$

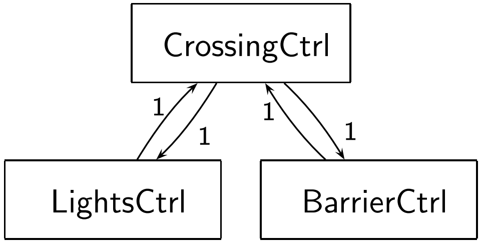
UML

Live Sequence Charts — Abstract Syntax

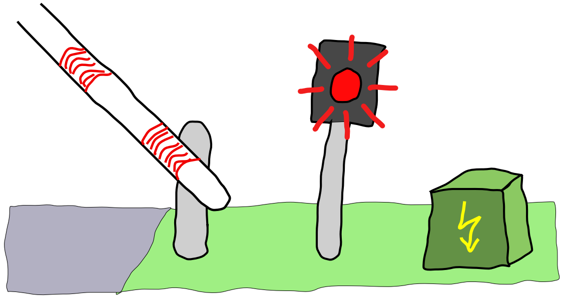
Example



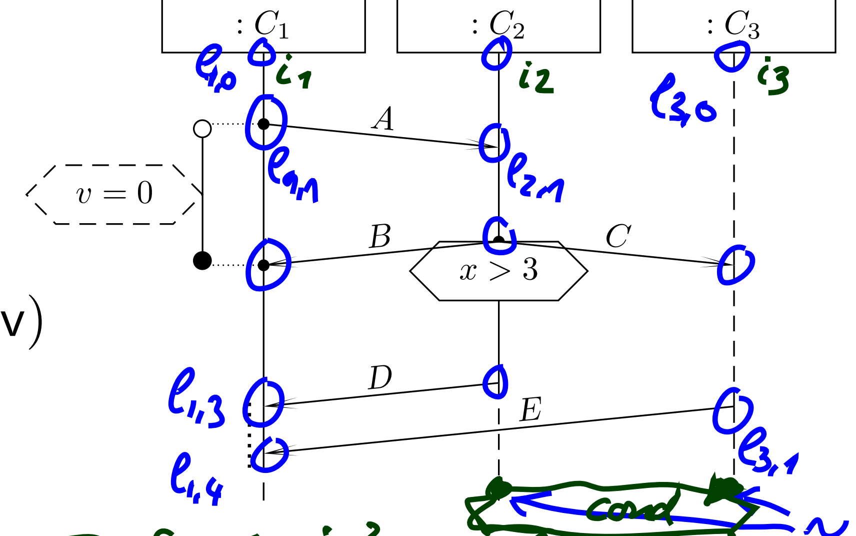
LSC: L
AC: $actcond$
AM: invariant I: strict
Environment
: LightsCtrl
: CrossingCtrl
: BarrierCtrl
secreq
t(10)
lights_on
barrier_down
Operational
$[1,3]$
$[1,5]$
$\neg$MvUp
lights_ok
barrier_ok
done
t



CrossingCtrl
1
1
1
1
LightsCtrl
BarrierCtrl



LSC Body: Abstract Syntax



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

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

- 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 $l \preceq l'$,
  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** where $Expr_{\mathscr{S}}$ are OCL expressions over $W = I \cup \{self\}$ 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**,

$I = \{i_1, i_2, i_3\}$

$\mathscr{L} = \{l_{1,0}, \dots, l_{3,1}\}$,

$l_{1,0} < l_{1,1}$

$l_{1,1} < l_{2,1}$ $\quad l_{1,3} \not< l_{1,4}$

$l_{1,2} < l_{1,3}$ $\quad l_{1,4} \not< l_{1,3}$

$l_{1,2} < l_{1,4}$

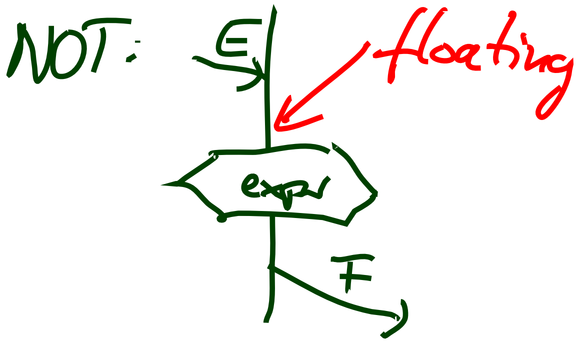
$\mathsf{Msg} = \{(l_{1,1}, A, l_{2,1}), \dots\}$

$\mathsf{Cond} = \{(\{l_{2,2}\}, x > 3, hot), \dots\}$

$\mathsf{LocInv} = \{(l_{1,1}, \circ, v = 0, l_{1,2}, \bullet), \dots\}$

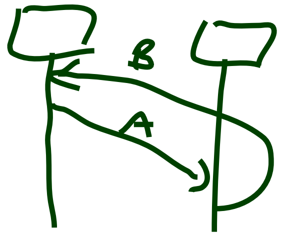
Well-Formedness

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.
    $$\exists\,(L, expr, \theta) \in \mathsf{Cond} : l \in L, \text{ or}$$
  - 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, 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.
    $$\exists\,(l_1, b, l_2) \in \mathsf{Msg} : l \in \{l_1, l_2\}.$$



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

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.

[Fecher and Schönborn, 2007] Fecher, H. and Schönborn, J. (2007). UML 2.0 state machines: Complete formal semantics via core state machines. In Brim, L., Haverkort, B. R., Leucker, M., and van de Pol, J., editors, *FMICS/PDMC*, volume 4346 of *LNCS*, pages 244–260. Springer.

[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.

[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.