Softwaretechnik / Software-Engineering

Lecture 09: Live Sequence Charts

2015-06-11

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Scenarios and Anti-Scenarios
- User Stories, Use Cases, Use Case Diagrams
- LSC: abstract and concrete syntax

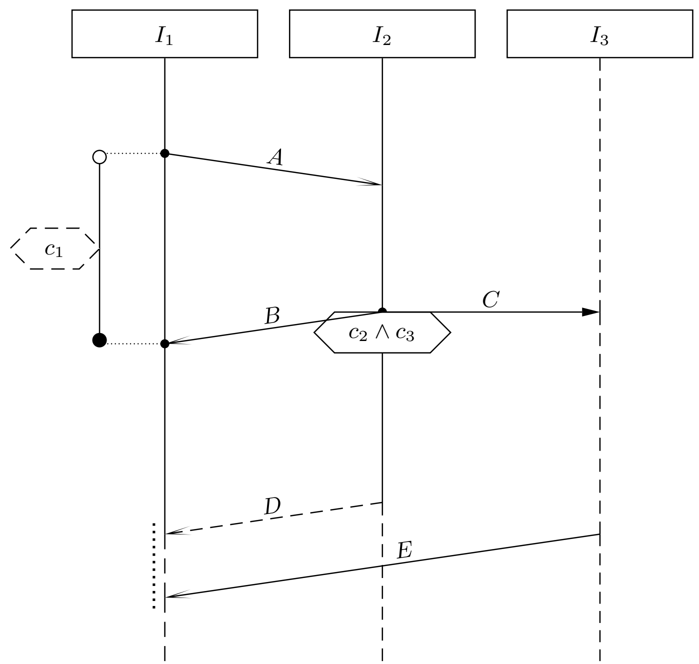
This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
    - Which are the cuts and firedsets of this LSC?
    - Construct the TBA of a given LSC body.
    - Given a set of LSCs, which scenario/anti-scenario/requirement is formalised by them?
    - Formalise this positive scenario/anti-scenario/requirement using LSCs.
- **Content:**
    - Excursion: automata accepting infinite words
    - Cuts and Firedsets, automaton construction
    - existential LSCs, pre-charts, universal LSCs
    - Requirements Engineering: conclusions

Recall: LSC Body Syntax

LSC Body Example

- $\mathcal{L} : l_{1,0} \prec l_{1,1} \prec l_{1,2} \prec l_{1,3}, \quad l_{1,2} \prec l_{1,4}, \quad l_{2,0} \prec l_{2,1} \prec l_{2,2} \prec l_{2,3}, \quad l_{3,0} \prec l_{3,1} \prec l_{3,2},$
  $l_{1,1} \prec l_{2,1}, \quad l_{2,2} \prec l_{1,2}, \quad l_{2,3} \prec l_{1,3}, \quad l_{3,2} \prec l_{1,4}, \quad l_{2,2} \sim l_{3,1},$
- $\mathcal{I} = \{\{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, l_{1,4}\}, \{l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}\}, \{l_{3,0}, l_{3,1}, l_{3,2}\}\},$
- $\mathsf{Msg} = \{(l_{1,1}, A, l_{2,1}), (l_{2,2}, B, l_{1,2}), (l_{2,2}, C, l_{3,1}), (l_{2,3}, D, l_{1,3}), (l_{3,2}, E, l_{1,4})\}$
- $\mathsf{Cond} = \{(\{l_{2,2}\}, c_2 \wedge c_3)\},$
- $\mathsf{LocInv} = \{(l_{1,1}, \circ, c_1, l_{1,2}, \bullet)\}$



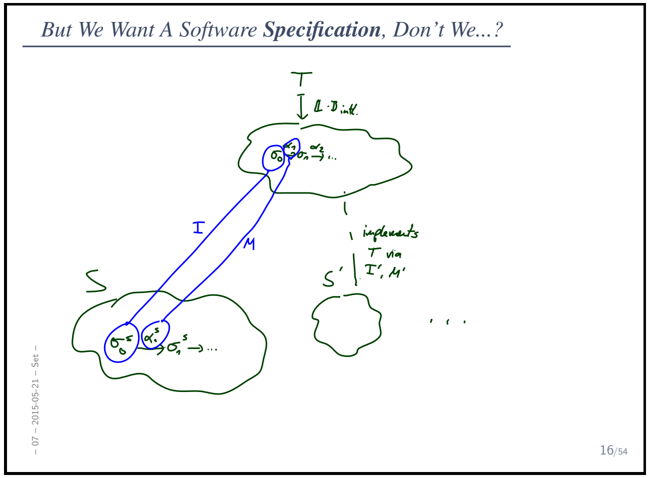
LSC Semantics

The Big Picture

- **Recall**: **decision tables**
- By the standard semantics, a decision table T is **software**,
  $[\![T]\!] = \{\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$ is a set of computation paths.

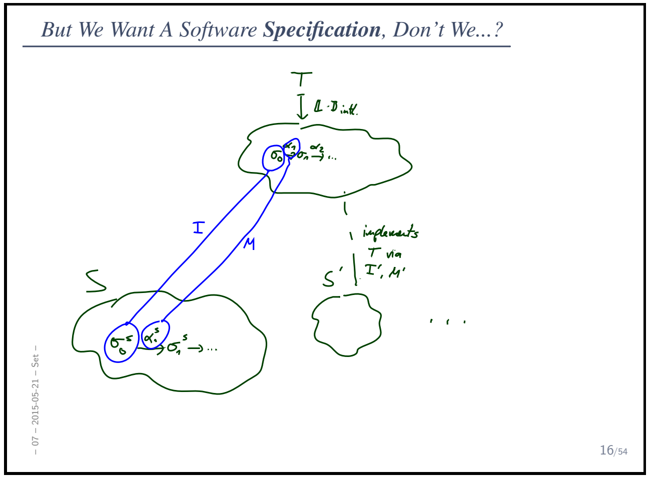
The Big Picture

- **Recall**: **decision tables**
- By the standard semantics, a decision table T is **software**, $[\![T]\!] = \{\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$ is a set of computation paths.
- **Recall**: Decision tables as software specification:



The Big Picture

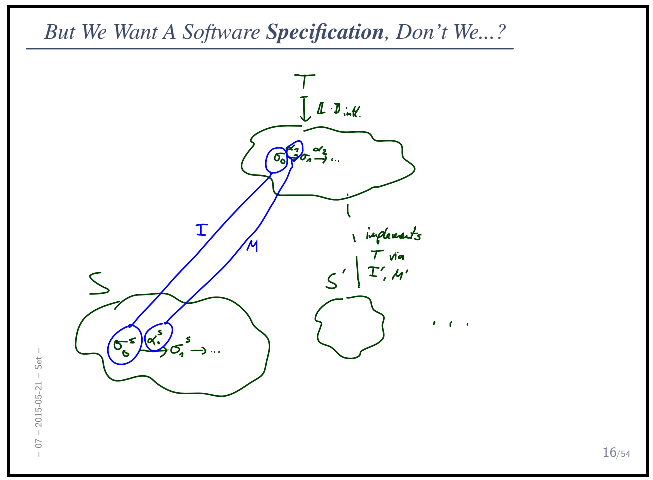
- **Recall**: **decision tables**
- By the standard semantics, a decision table T is **software**,
  $[\![T]\!] = \{\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$ is a set of computation paths.
- **Recall**: Decision tables as software specification:



- We want **the same** for LSCs.

The Big Picture

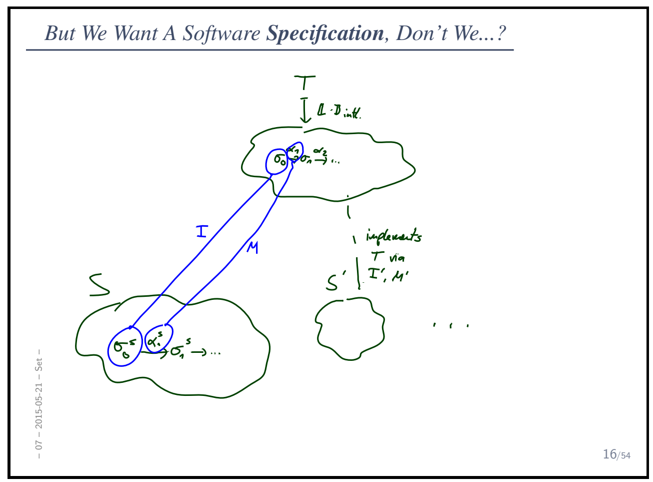
- **Recall**: **decision tables**
- By the standard semantics, a decision table T is **software**, $[\![T]\!] = \{\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$ is a set of computation paths.
- **Recall**: Decision tables as software specification:



- We want **the same** for LSCs.
- We will give a **procedure** to construct for each LSC $\mathscr{L}$ an **automaton** $\mathcal{B}(\mathscr{L})$. The language (or semantics) of $\mathscr{L}$ is the set of comp. paths **accepted** by $\mathcal{B}(\mathscr{L})$. Thus an LSC is also software.

The Big Picture

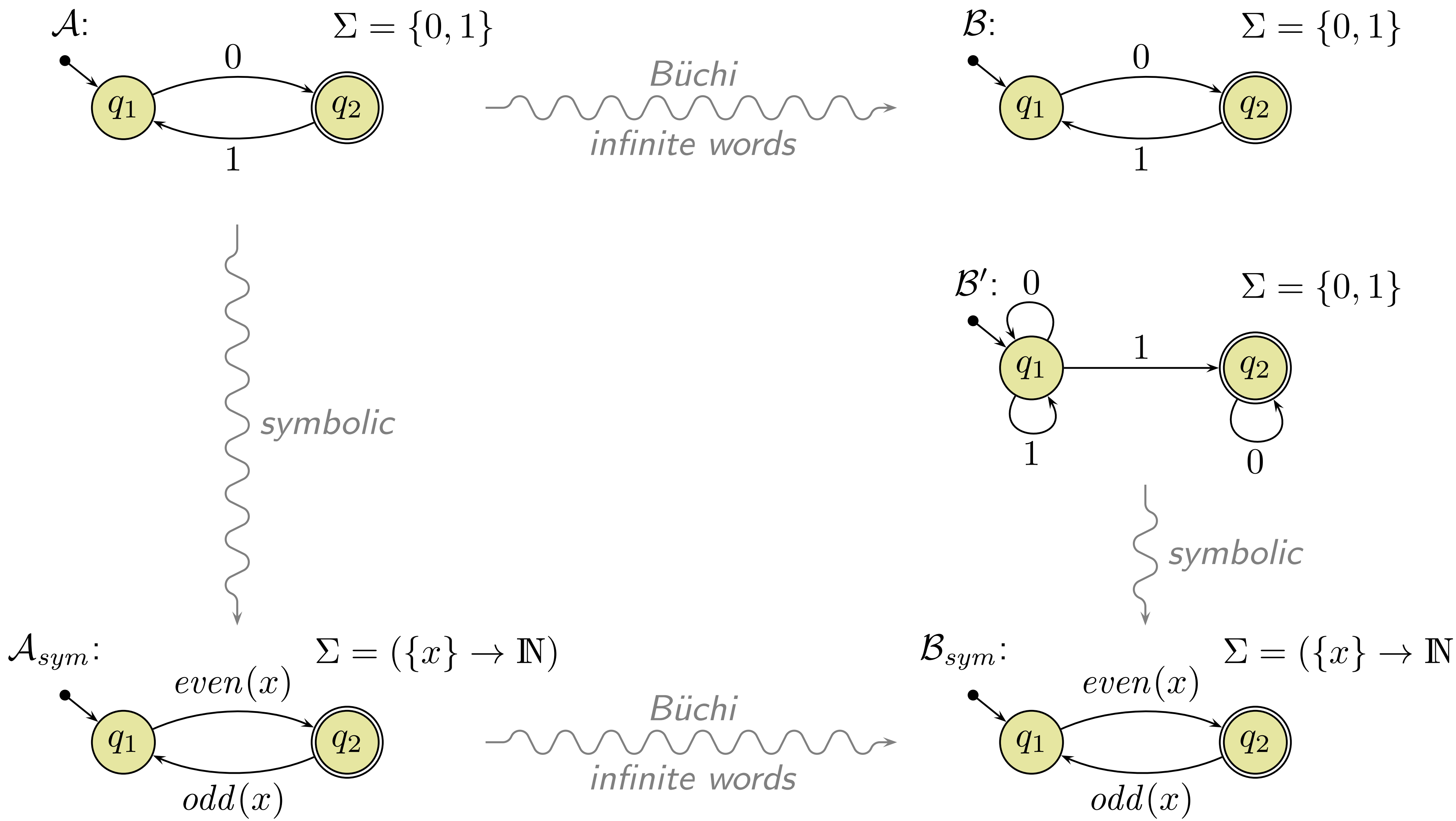
- **Recall**: **decision tables**
- By the standard semantics, a decision table T is **software**,
  $[\![T]\!] = \{\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$ is a set of computation paths.
- **Recall**: Decision tables as software specification:



- We want **the same** for LSCs.
- We will give a **procedure** to construct for each LSC $\mathscr{L}$ an **automaton** $\mathcal{B}(\mathscr{L})$.
  The language (or semantics) of $\mathscr{L}$ is the set of comp. paths **accepted** by $\mathcal{B}(\mathscr{L})$.
  Thus an LSC is also software.
- **Problem**: computation paths may be infinite $\rightarrow$ Büchi acceptance.

Excursion: Symbolic Büchi Automata

From Finite Automata to Symbolic Büchi Automata



Symbolic Büchi Automata

Definition. A **Symbolic Büchi Automaton** (TBA) is a tuple

$$\mathcal{B} = (\mathcal{C}, Q, q_{ini}, \rightarrow, Q_F)$$

where

- $\mathcal{C}$ is a set of atomic propositions,
- Q is a finite set of **states**,
- $q_{ini} \in Q$ is the initial state,
- $\rightarrow \subseteq Q \times \Phi(\mathcal{C}) \times Q$ is the finite **transition relation**.

  Each transitions $(q, \psi, q') \in \rightarrow$ from state q to state q' is labelled with a formula $\psi \in \Phi(\mathcal{C})$.
- $Q_F \subseteq Q$ is the set of **fair** (or accepting) states.

Run of TBA

Definition. Let $\mathcal{B} = (\mathcal{C}, Q, q_{ini}, \to, Q_F)$ be a TBA and

$$w = \sigma_1, \sigma_2, \sigma_3, \dots \in (\mathcal{C} \to \mathbb{B})^\omega$$

an infinite word, each letter is a valuation of $\mathcal{C}_\mathcal{B}$.

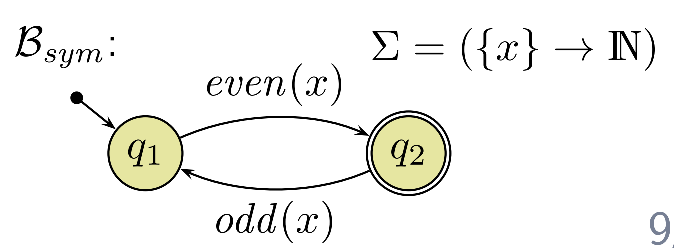
An infinite sequence

$$\varrho = q_0, q_1, q_2, \ldots \in Q^\omega$$

of states is called **run** of $\mathcal{B}$ over w if and only if

- $q_0 = q_{ini}$,
- for each $i \in \mathbb{N}_0$ there is a transition $(q_i, \psi_i, q_{i+1}) \in \to$ s.t. $\sigma_i \models \psi_i$.

Example:



The Language of a TBA

Definition.
We say TBA $\mathcal{B} = (\mathcal{C}, Q, q_{ini}, \to, Q_F)$ **accepts** the word $w = (\sigma_i)_{i \in \mathbb{N}_0} \in (\mathcal{C} \to \mathbb{B})^\omega$ if and only if $\mathcal{B}$ **has** a run

$$\varrho = (q_i)_{i \in \mathbb{N}_0}$$

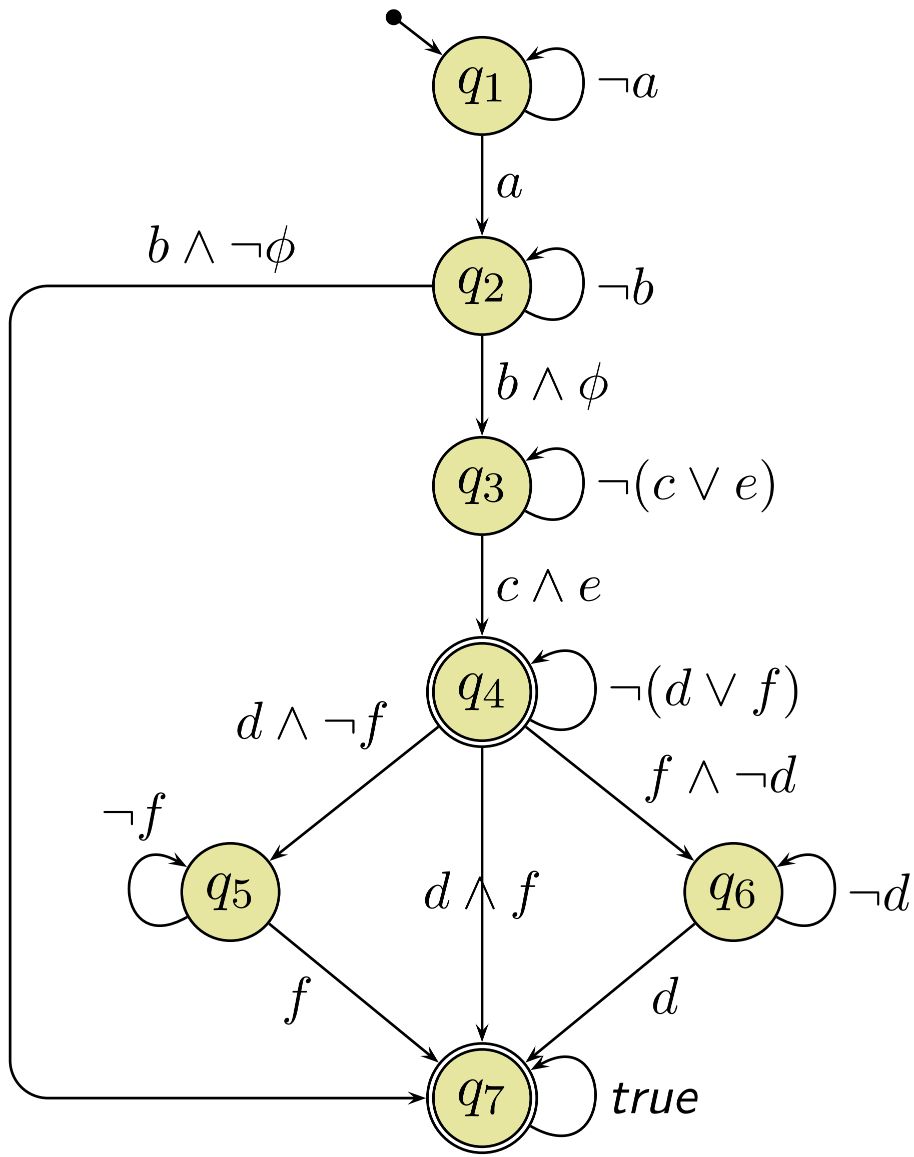
over w such that fair (or accepting) states are **visited infinitely often** by ϱ, i.e., such that

$$\forall\, i \in \mathbb{N}_0\ \exists\, j > i : q_j \in Q_F.$$

We call the set $Lang(\mathcal{B}) \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ of words that are accepted by $\mathcal{B}$ the **language of** $\mathcal{B}$.

Example

run: $\varrho = q_0, q_1, q_2, \ldots \in Q^\omega$ s.t. $\sigma_i \models \psi_i$, $i \in \mathbb{N}_0$.



LSC Semantics: TBA Construction

LSC Semantics: It's in the Cuts!

LSC Semantics: It's in the Cuts!

Definition. Let $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$ be an LSC body.

A non-empty set $\emptyset \neq C \subseteq \mathcal{L}$ is called a **cut** of the LSC body iff C

- is **downward closed**, i.e.

$$\forall\, l, l' \in \mathcal{L} \bullet l' \in C \wedge l \preceq l' \implies l \in C,$$

- is **closed** under **simultaneity**, i.e.

$$\forall\, l, l' \in \mathcal{L} \bullet l' \in C \wedge l \sim l' \implies l \in C, \text{ and}$$

- comprises at least **one location per instance line**, i.e.

$$\forall\, I \in \mathcal{I} \bullet C \cap I \neq \emptyset.$$

LSC Semantics: It's in the Cuts!

Definition. Let $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$ be an LSC body.

A non-empty set $\emptyset \neq C \subseteq \mathcal{L}$ is called a **cut** of the LSC body iff C

- is **downward closed**, i.e.
$$\forall\, l, l' \in \mathcal{L} \bullet l' \in C \wedge l \preceq l' \implies l \in C,$$
- is **closed** under **simultaneity**, i.e.
$$\forall\, l, l' \in \mathcal{L} \bullet l' \in C \wedge l \sim l' \implies l \in C, \text{ and}$$
- comprises at least **one location per instance line**, i.e.
$$\forall\, I \in \mathcal{I} \bullet C \cap I \neq \emptyset.$$

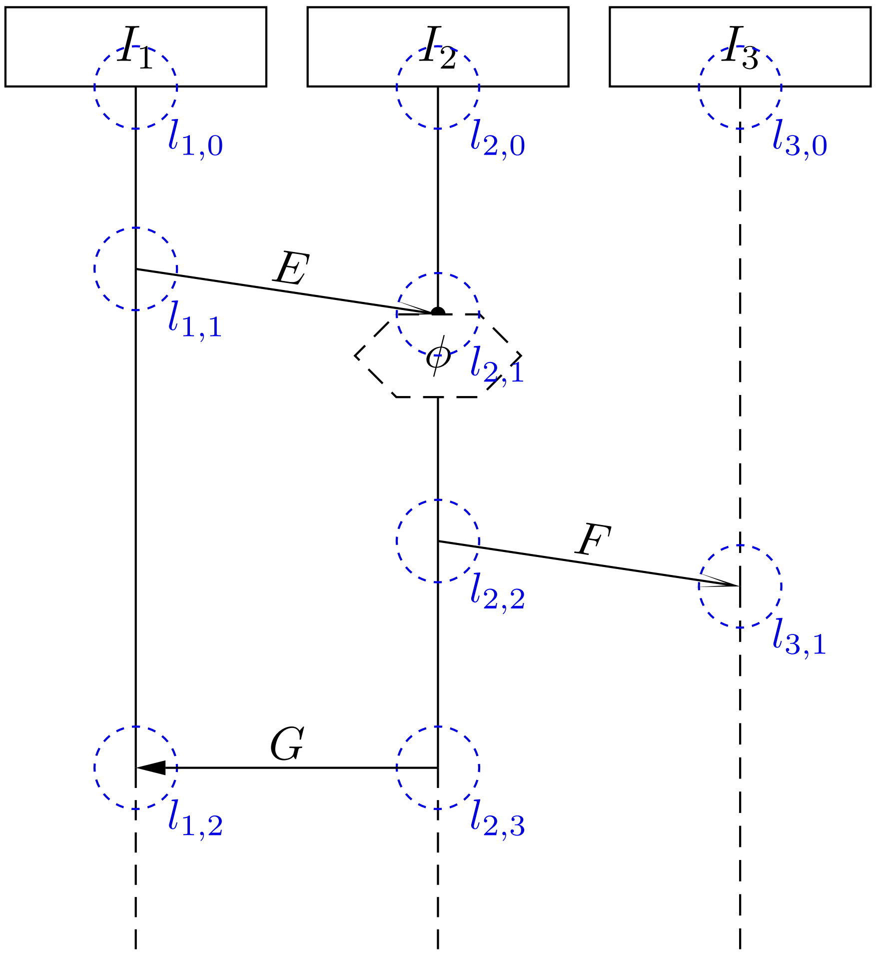
The temperature function is extended to cuts as follows:

$$\Theta(C) = \begin{cases} \text{hot} & \text{, if } \exists\, l \in C \bullet (\nexists\, l' \in C \bullet l \prec l') \wedge \Theta(l) = \text{hot} \\ \text{cold} & \text{, otherwise} \end{cases}$$

that is, C is **hot** if and only if at least one of its maximal elements is hot.

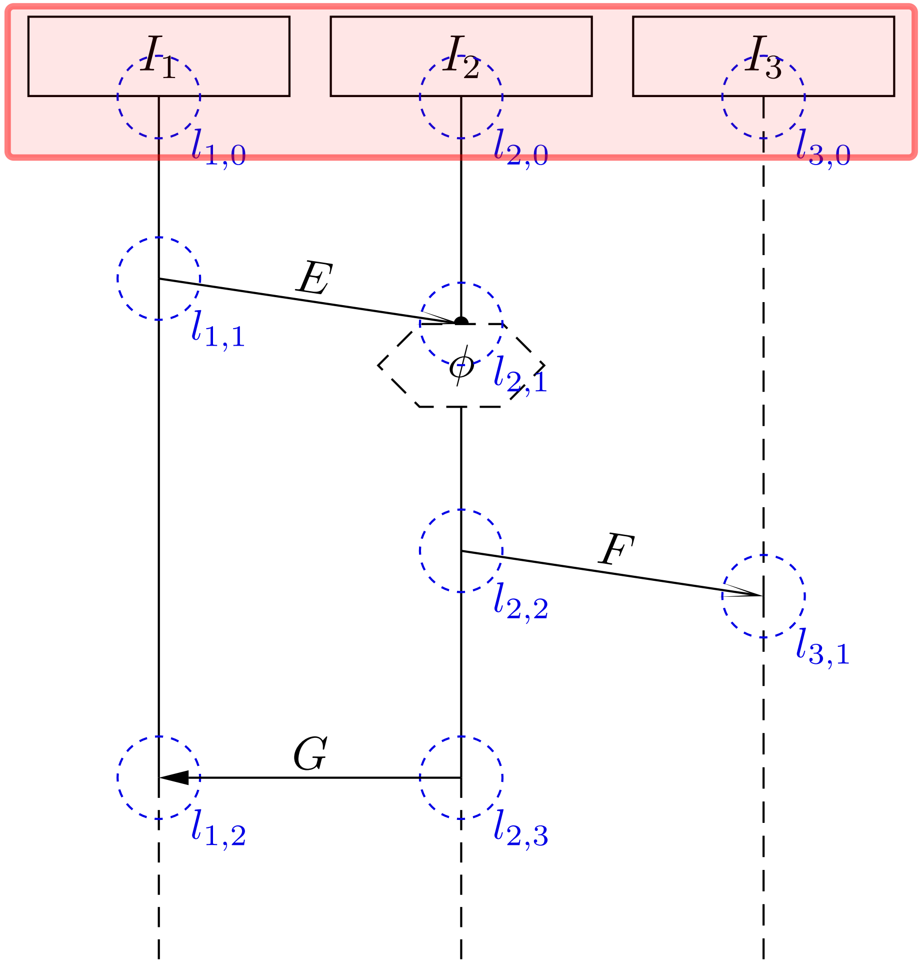
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



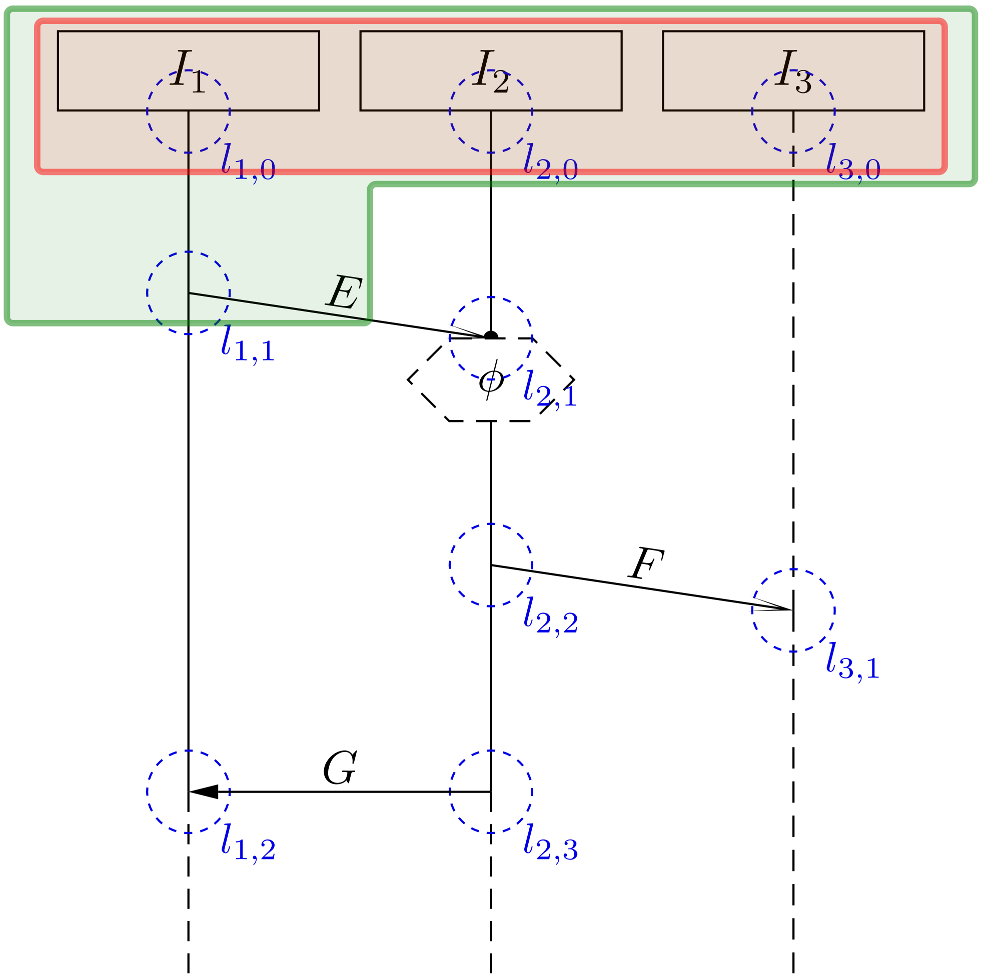
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



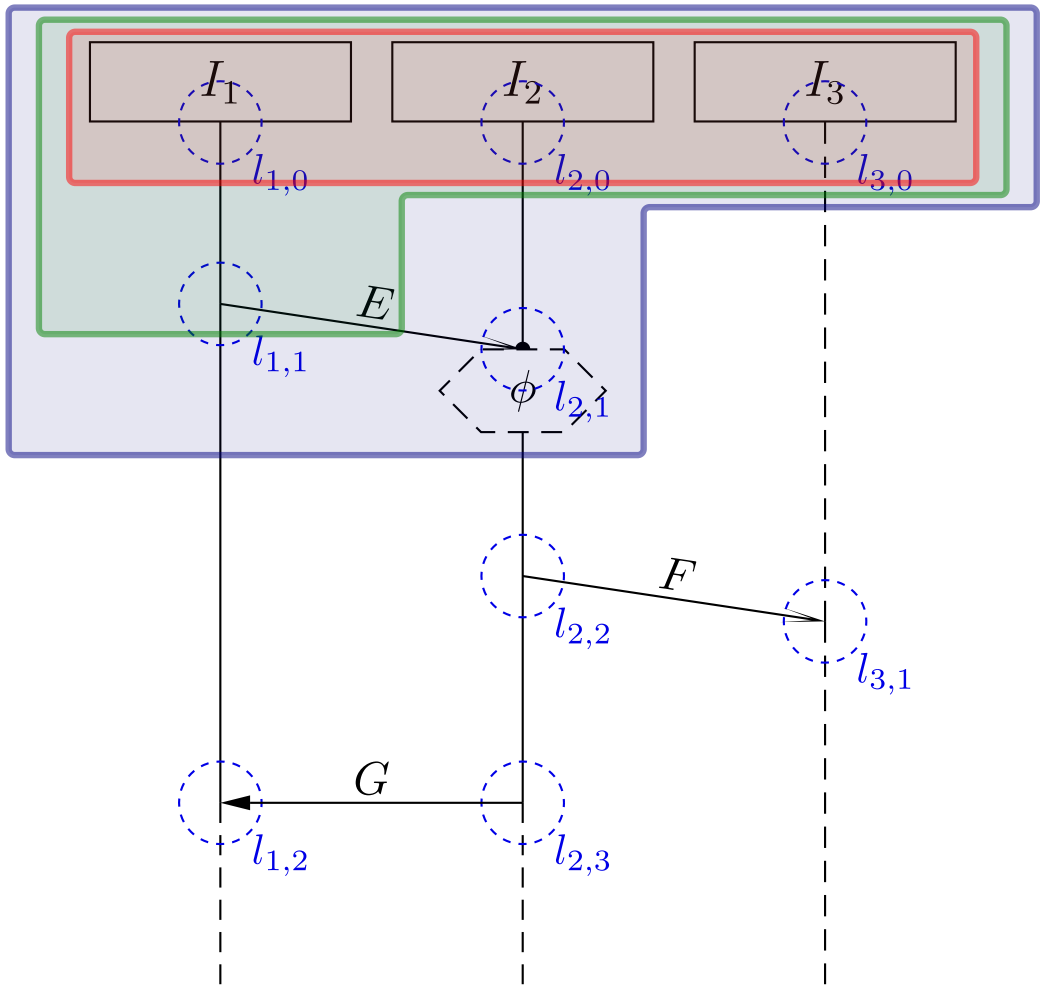
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



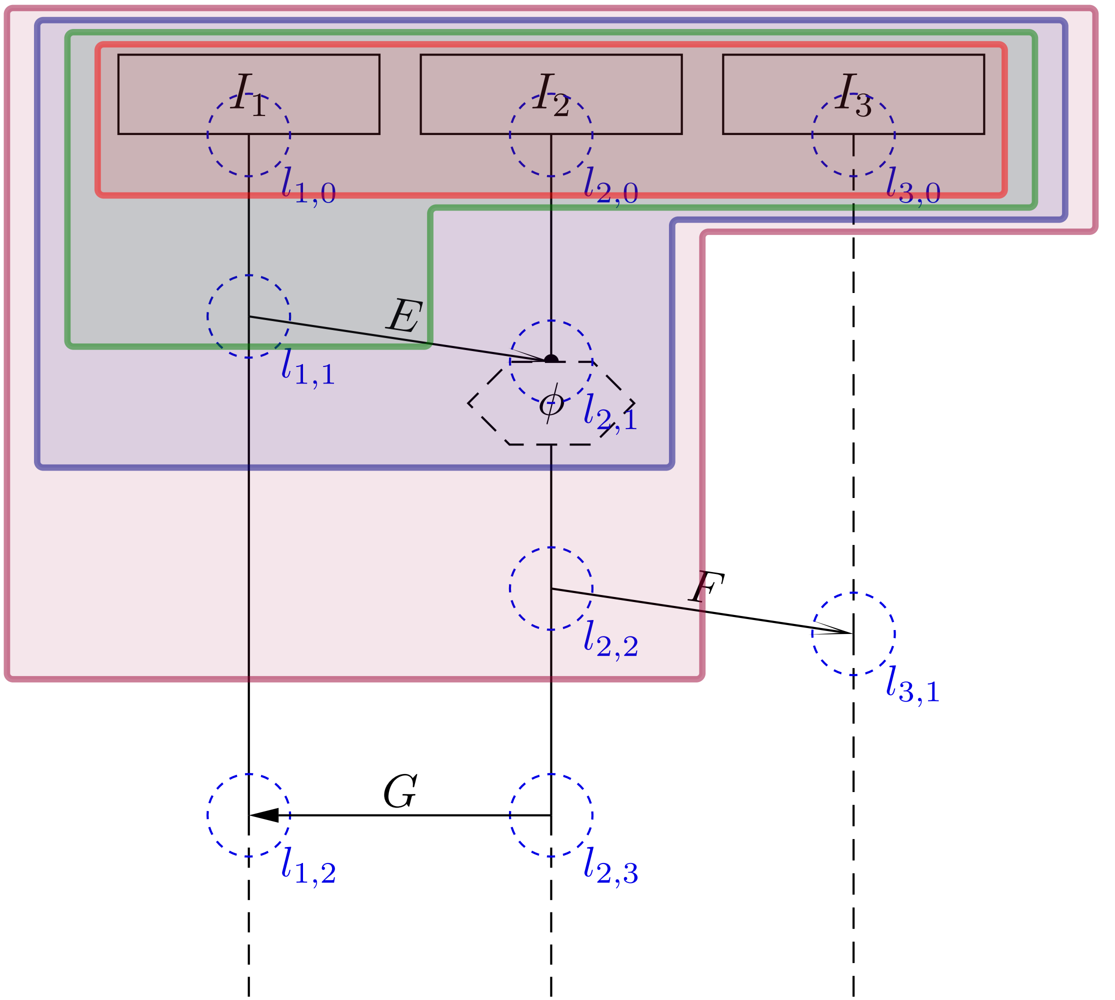
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



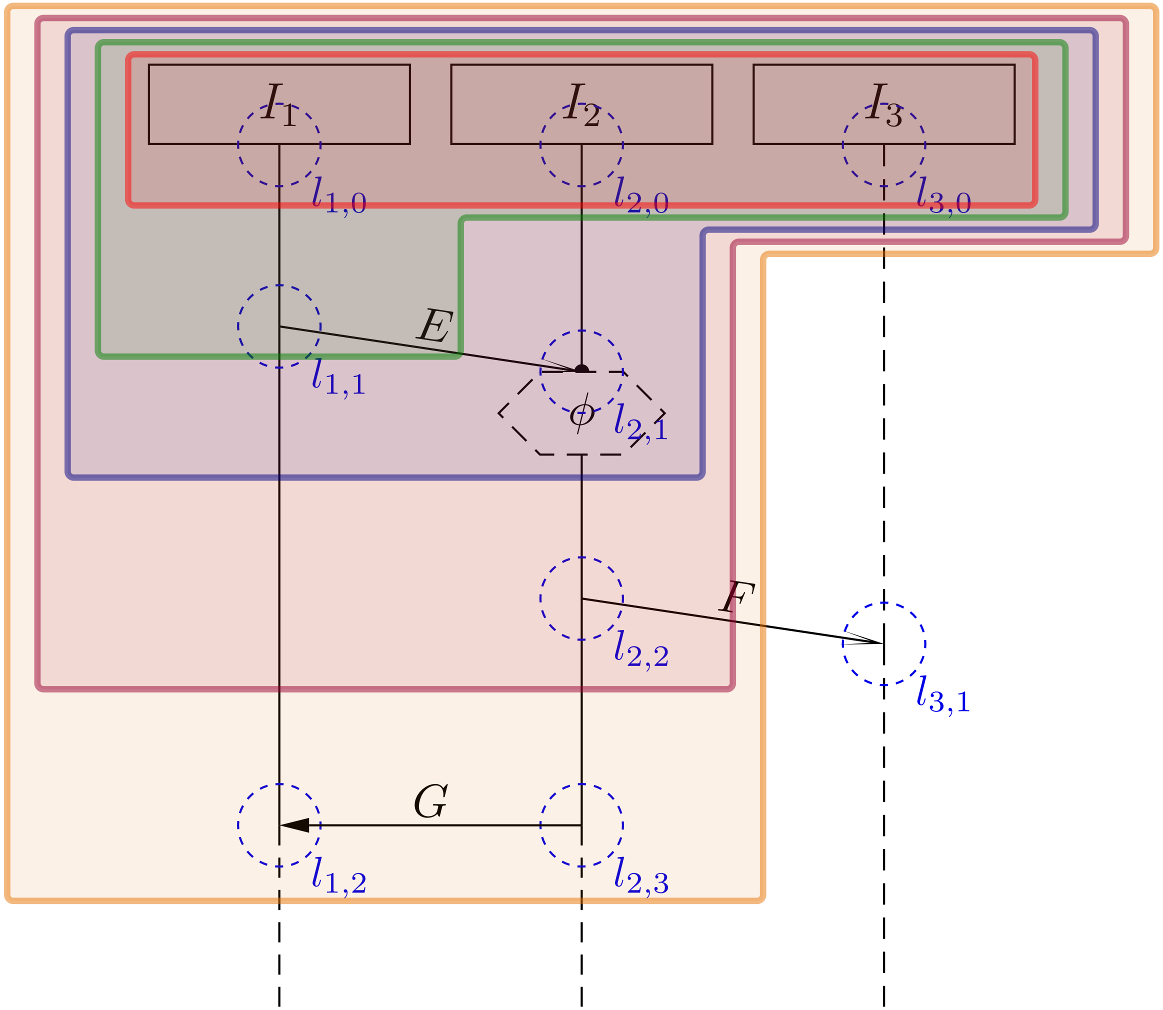
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



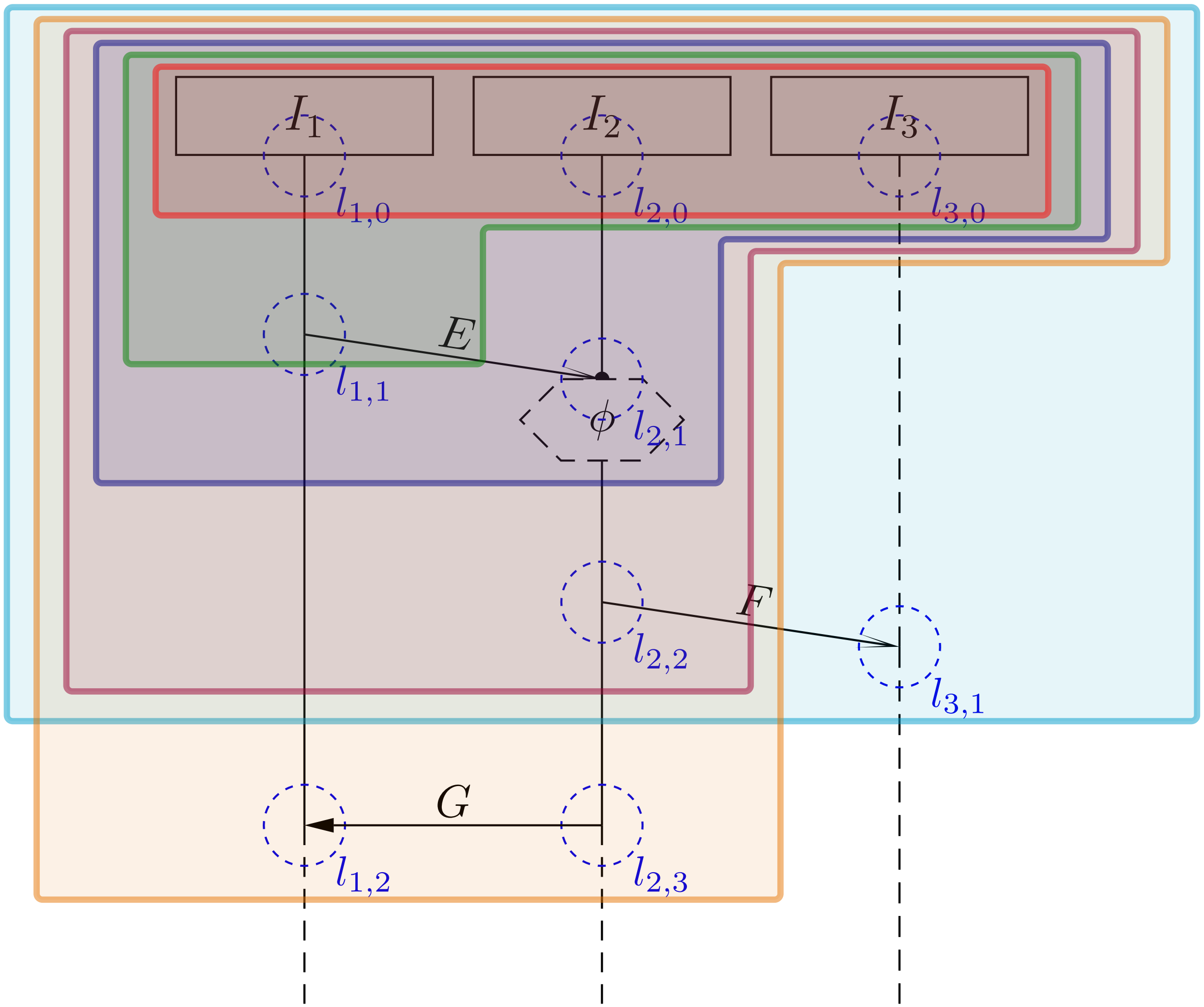
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



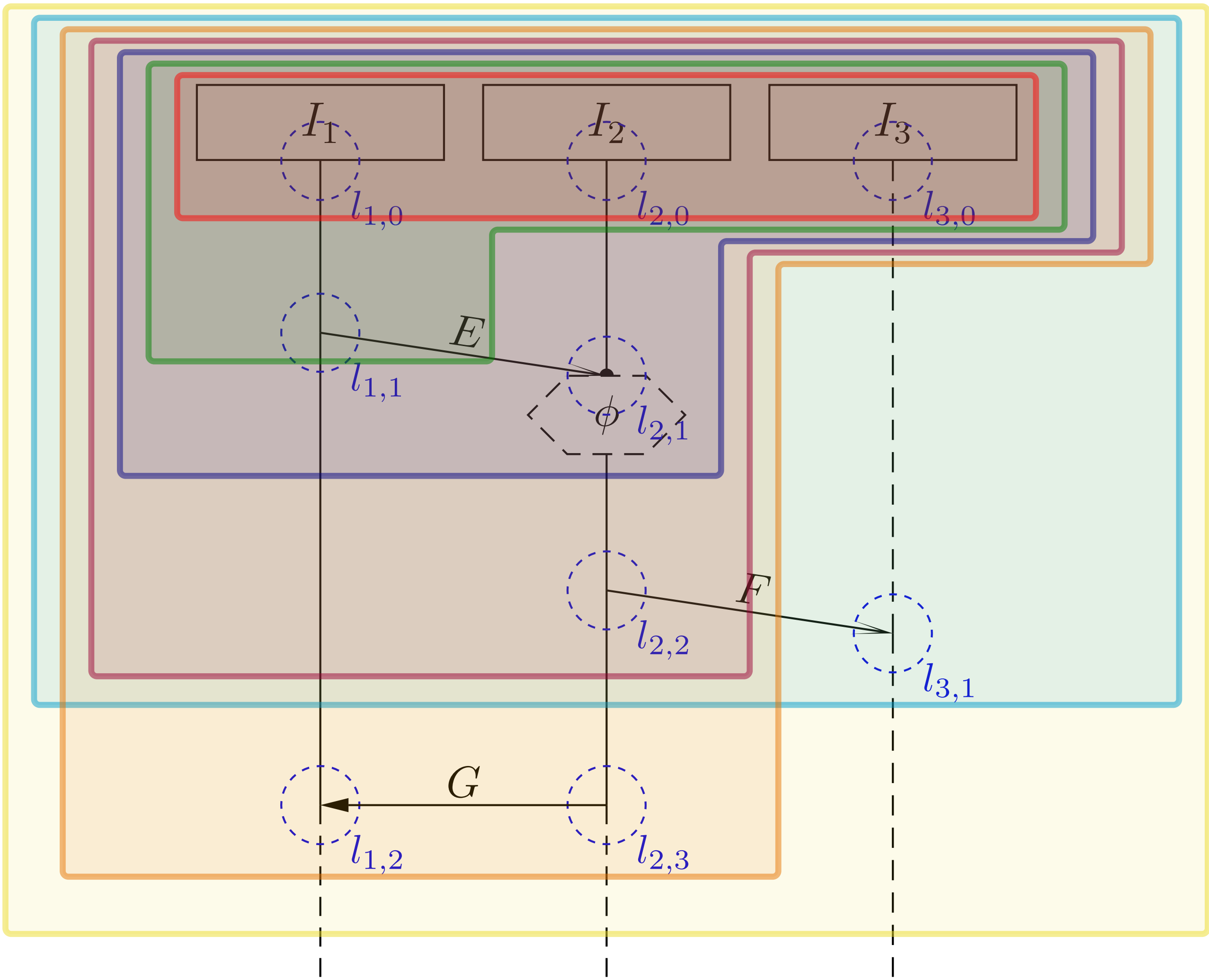
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



A Successor Relation on Cuts

The partial order "$\preceq$" and the simultaneity relation "$\sim$" of locations induce a **direct successor relation** on cuts of $\mathcal{L}$ as follows:

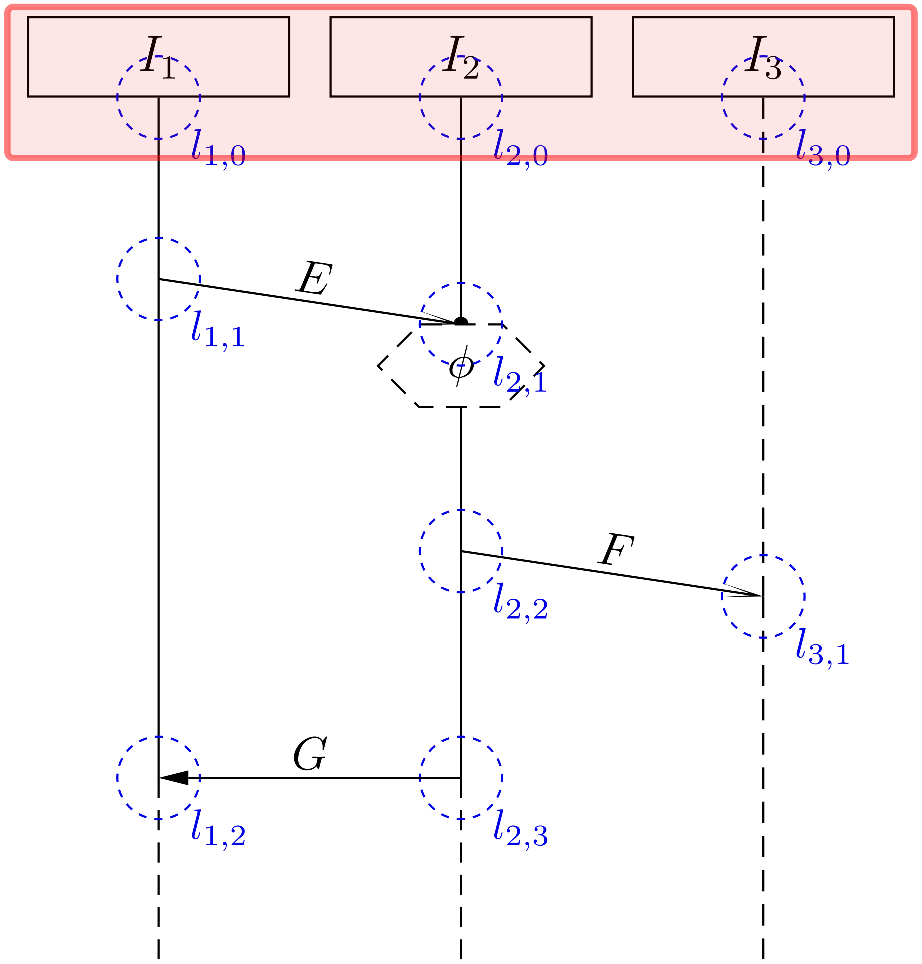
Definition.
Let $C \subseteq \mathcal{L}$ bet a cut of LSC body $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$.
A set $\emptyset \neq \mathcal{F} \subseteq \mathcal{L}$ is called **fired-set** $\mathcal{F}$ of C if and only if

- $C \cap \mathcal{F} = \emptyset$ and $C \cup \mathcal{F}$ is a cut, i.e. $\mathcal{F}$ is closed under simultaneity,
- all locations in $\mathcal{F}$ are direct $\prec$-successors of the front of C, i.e.
$$\forall\, l \in \mathcal{F}\ \exists\, l' \in C \bullet l' \prec l \wedge (\nexists\, l'' \in C \bullet l' \prec l''),$$
- locations in $\mathcal{F}$, that lie on the same instance line, are pairwise unordered, i.e.
$$\forall\, l \neq l' \in \mathcal{F} \bullet (\exists\, I \in \mathcal{I} \bullet \{l, l'\} \subseteq I) \implies l \not\preceq l' \wedge l' \not\preceq l,$$
- for each asynchronous message reception in $\mathcal{F}$, the corresponding sending is already in C,
$$\forall\, (l, E, l') \in \mathsf{Msg} \bullet l' \in \mathcal{F} \implies l \in C.$$

The cut $C' = C \cup \mathcal{F}$ is called **direct successor of C via $\mathcal{F}$**, denoted by $C \rightsquigarrow_{\mathcal{F}} C'$.

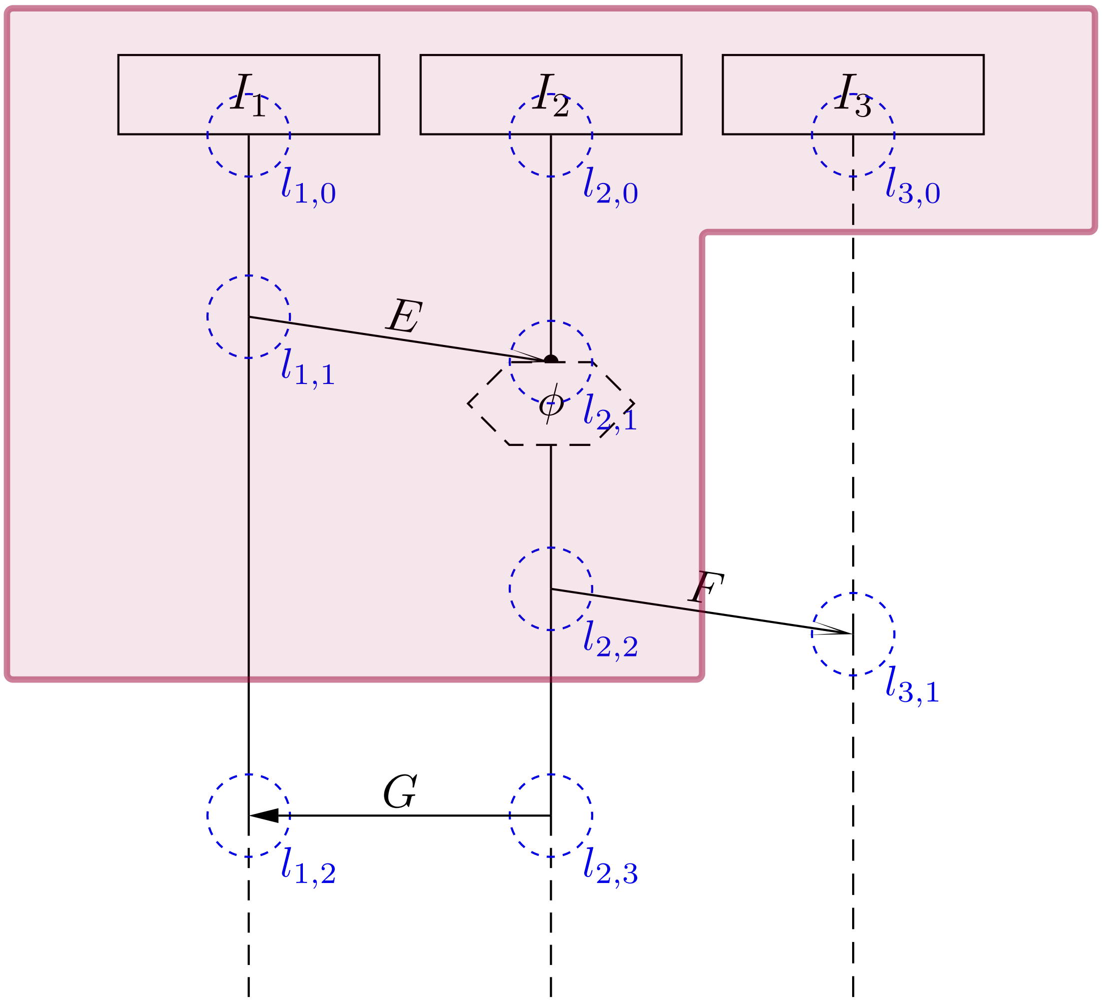
Successor Cut Example

$C \cap \mathcal{F} = \emptyset$ — $C \cup \mathcal{F}$ is a cut — only direct $\prec$-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in

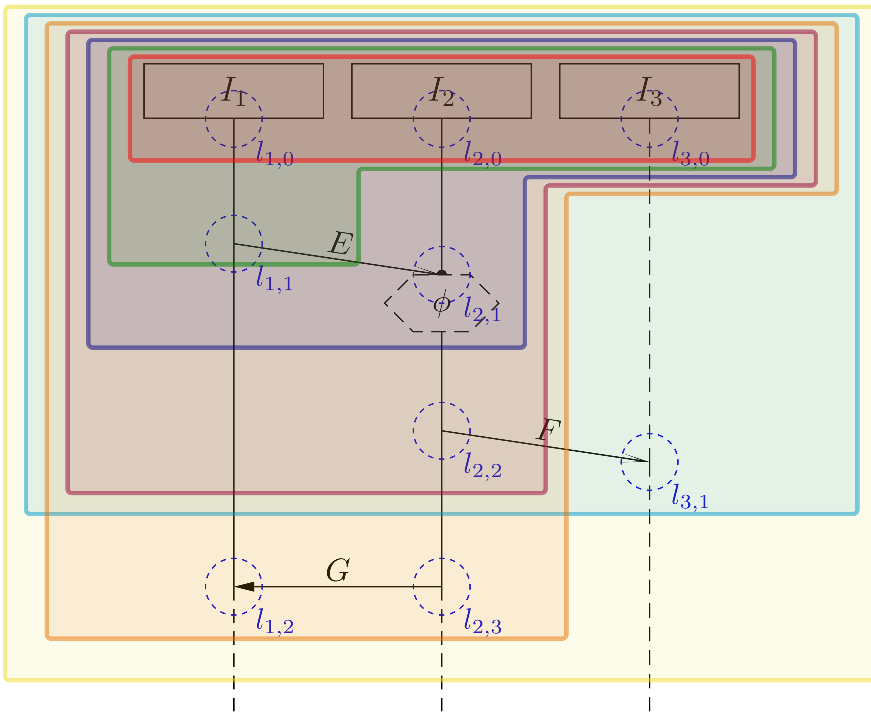


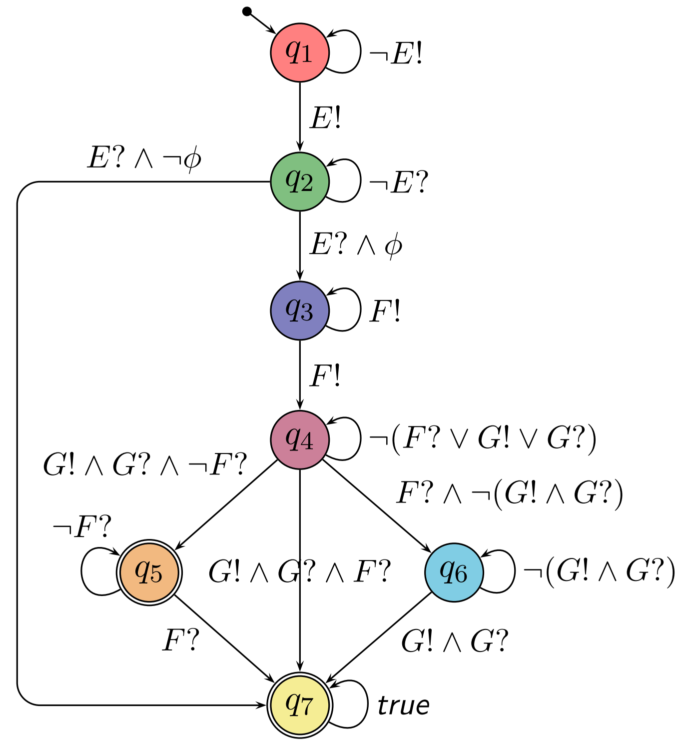
Successor Cut Example

$C \cap \mathcal{F} = \emptyset$ — $C \cup \mathcal{F}$ is a cut — only direct $\prec$-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in



Language of LSC Body: Example





The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ over C and $\mathcal{E}$ is $(\mathcal{C}, Q, q_{ini}, \rightarrow, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C} = C \cup \mathcal{E}_{!?}$, where $\mathcal{E}_{!?} = \{E!, E? \mid E \in \mathcal{E}\}$,
- $\rightarrow$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_{\mathcal{F}}$), and **legal exits** (cold cond./local inv.),
- $Q_F = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts and the maximal cut.

TBA Construction Principle

Recall: The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ is $(\mathcal{C}, Q, q_{ini}, \rightarrow, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C} = C \cup \{E!, E? \mid E \in \mathcal{E}\}$,
- $\rightarrow$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_{\mathcal{F}}$), and **legal exits** (cold cond./local inv.),
- $\mathcal{F} = \{C \in Q \mid \Theta(C) = \mathsf{cold} \lor C = \mathcal{L}\}$ is the set of cold cuts.

$$\rightarrow = \{(q, \qquad\qquad , q) \mid q \in Q\} \cup \{(q, \qquad\qquad\qquad , q') \mid q \rightsquigarrow_{\mathcal{F}} q'\} \cup \{(q, \qquad\qquad , \mathcal{L}) \mid q \in Q\}$$

– 09 – 2015-06-11 – Scutfire –

TBA Construction Principle

Recall: The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ is $(\mathcal{C}, Q, q_{ini}, \to, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C} = C \cup \{E!, E? \mid E \in \mathcal{E}\}$,
- $\to$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_\mathcal{F}$), and **legal exits** (cold cond./local inv.),
- $\mathcal{F} = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts.

So in the following, we "only" need to construct the transitions' labels:

$$\to = \{(q, \qquad , q) \mid q \in Q\} \cup \{(q, \qquad , q') \mid q \rightsquigarrow_\mathcal{F} q'\} \cup \{(q, \qquad , \mathcal{L}) \mid q \in Q\}$$

TBA Construction Principle

Recall: The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ is $(\mathcal{C}, Q, q_{ini}, \rightarrow, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C} = C \cup \{E!, E? \mid E \in \mathcal{E}\}$,
- $\rightarrow$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_{\mathcal{F}}$), and **legal exits** (cold cond./local inv.),
- $\mathcal{F} = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts.

So in the following, we "only" need to construct the transitions' labels:

$$\rightarrow = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \rightsquigarrow_{\mathcal{F}} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\}$$

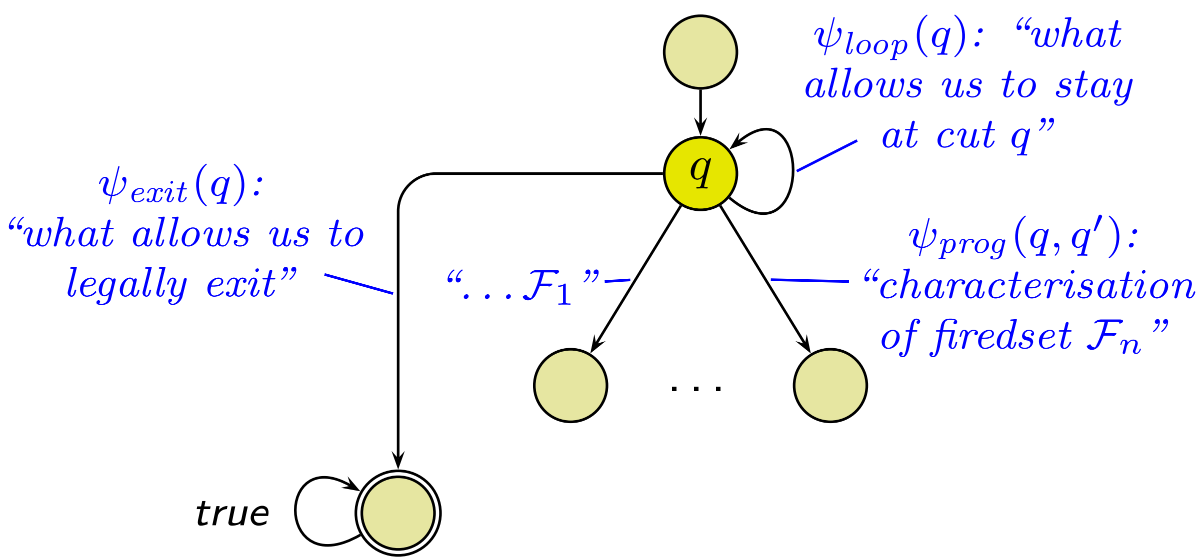
TBA Construction Principle

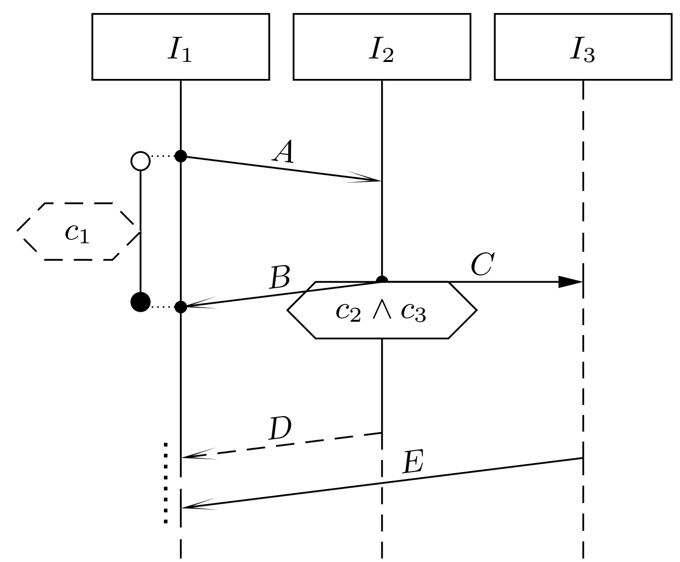
Recall: The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ is $(\mathcal{C}, Q, q_{ini}, \rightarrow, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C} = C \cup \{E!, E? \mid E \in \mathcal{E}\}$,
- $\rightarrow$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_{\mathcal{F}}$), and **legal exits** (cold cond./local inv.),
- $\mathcal{F} = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts.

So in the following, we "only" need to construct the transitions' labels:

$$\rightarrow = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \rightsquigarrow_{\mathcal{F}} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\}$$





TBA Construction Principle

So in the following, we "only" need to construct the transitions' labels:

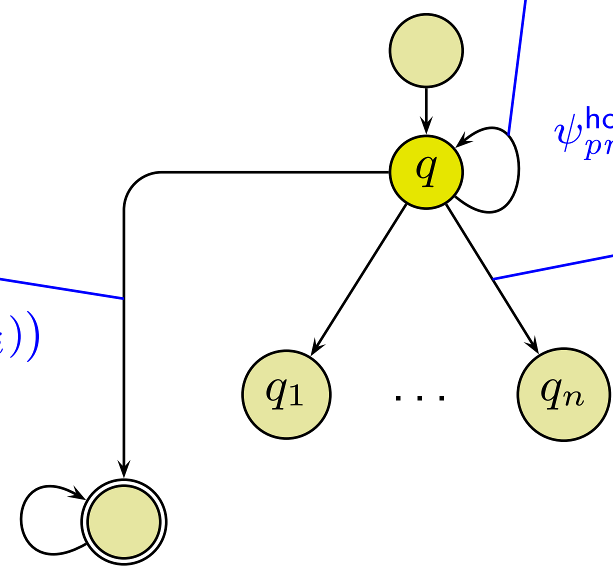
$$\rightarrow = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \rightsquigarrow_{\mathcal{F}} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\}$$

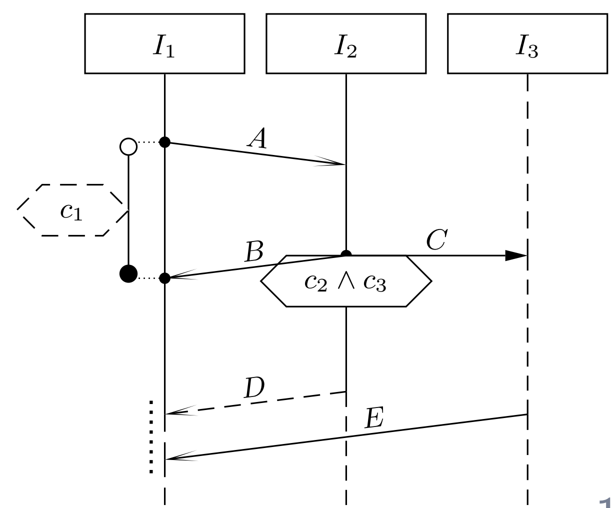
$$\psi_{loop}(q) = \underbrace{\psi^{\mathsf{Msg}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{hot}}(q)}_{=:\psi^{\mathsf{hot}}_{loop}(q)} \wedge \psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)$$

$$\psi_{exit}(q) = (\psi^{\mathsf{hot}}_{loop}(q) \wedge \neg\psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)) \vee \bigvee_{1 \leq i \leq n} \psi^{\mathsf{hot}}_{prog}(q, q_i) \wedge (\neg\psi^{\mathsf{LocInv},\bullet}_{\mathsf{cold}}(q, q_i) \vee \neg\psi^{\mathsf{Cond}}_{\mathsf{cold}}(q, q_i))$$

$$\psi_{prog}(q, q_n) = \psi^{\mathsf{hot}}_{prog}(q, q_n) \wedge \psi^{\mathsf{Cond}}_{\mathsf{cold}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{cold}}(q, q_n)$$

$$\psi^{\mathsf{hot}}_{prog}(q, q_n) = \psi^{\mathsf{Msg}}(q, q_n) \wedge \psi^{\mathsf{Cond}}_{\mathsf{hot}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{hot}}(q, q_n)$$





Loop Condition

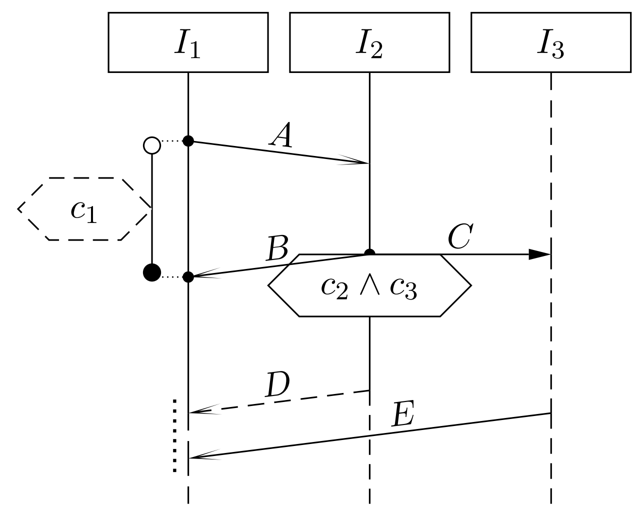
$$\psi_{loop}(q) = \psi^{\mathsf{Msg}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{hot}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)$$

- $\psi^{\mathsf{Msg}}(q) = \neg \bigvee_{1 \leq i \leq n} \psi^{\mathsf{Msg}}(q, q_i) \wedge \big(strict \implies \bigwedge_{\psi \in \mathcal{E}_{!?} \cap \mathsf{Msg}(\mathcal{L})} \neg\psi\big)$
- $\psi^{\mathsf{LocInv}}_{\theta}(q) = \bigwedge_{\ell=(l,\iota,\phi,l',\iota') \in \mathsf{LocInv},\ \Theta(\ell)=\theta,\ \ell \text{ active at } q} \phi$

  A location l is called **front location** of cut C if and only if $\nexists\, l' \in \mathcal{L} \bullet l \prec l'$.

  Local invariant $(l_0, \iota_0, \phi, l_1, \iota_1)$ is **active** at cut (!) q if and only if $l_0 \preceq l \preceq l_1$ for some front location l of cut (!) q.

- $\mathsf{Msg}(\mathcal{F}) = \{E! \mid (l, E, l') \in \mathsf{Msg},\ l \in \mathcal{F}\} \cup \{E? \mid (l, E, l') \in \mathsf{Msg},\ l' \in \mathcal{F}\}$
- $\mathsf{Msg}(\mathcal{F}_1, \dots, \mathcal{F}_n) = \bigcup_{1 \leq i \leq n} \mathsf{Msg}(\mathcal{F}_i)$



– 09 – 2015-06-11 – Slscsem –

Progress Condition

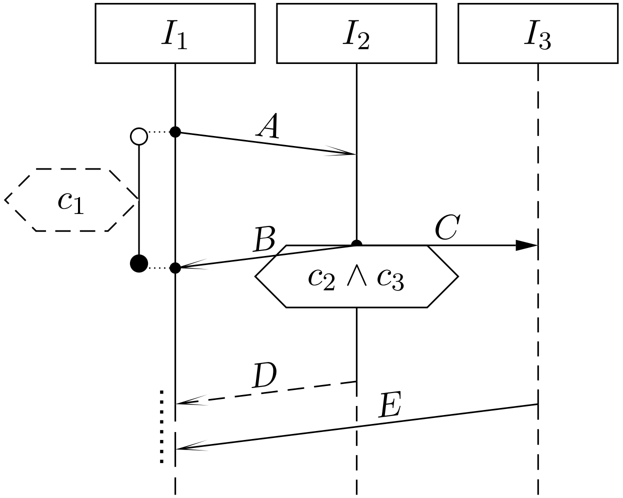
$$\psi^{\mathsf{hot}}_{prog}(q, q_i) = \psi^{\mathsf{Msg}}(q, q_n) \wedge \psi^{\mathsf{Cond}}_{\mathsf{hot}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{hot}}(q_n)$$

- $\psi^{\mathsf{Msg}}(q, q_i) = \bigwedge_{\psi \in \mathsf{Msg}(q_i \setminus q)} \psi \wedge \bigwedge_{j \neq i} \bigwedge_{\psi \in (\mathsf{Msg}(q_j \setminus q) \setminus \mathsf{Msg}(q_i \setminus q))} \neg\psi$
  $\wedge \left(strict \implies \bigwedge_{\psi \in (\mathcal{E}_{!?} \cap \mathsf{Msg}(\mathcal{L})) \setminus \mathsf{Msg}(\mathcal{F}_i)} \neg\psi\right)$
- $\psi^{\mathsf{Cond}}_{\theta}(q, q_i) = \bigwedge_{\gamma = (L, \phi) \in \mathsf{Cond},\ \Theta(\gamma) = \theta,\ L \cap (q_i \setminus q) \neq \emptyset} \phi$
- $\psi^{\mathsf{LocInv},\bullet}_{\theta}(q, q_i) = \bigwedge_{\lambda = (l, \iota, \phi, l', \iota') \in \mathsf{LocInv},\ \Theta(\lambda) = \theta,\ \lambda\ \bullet\text{-active at } q_i} \phi$

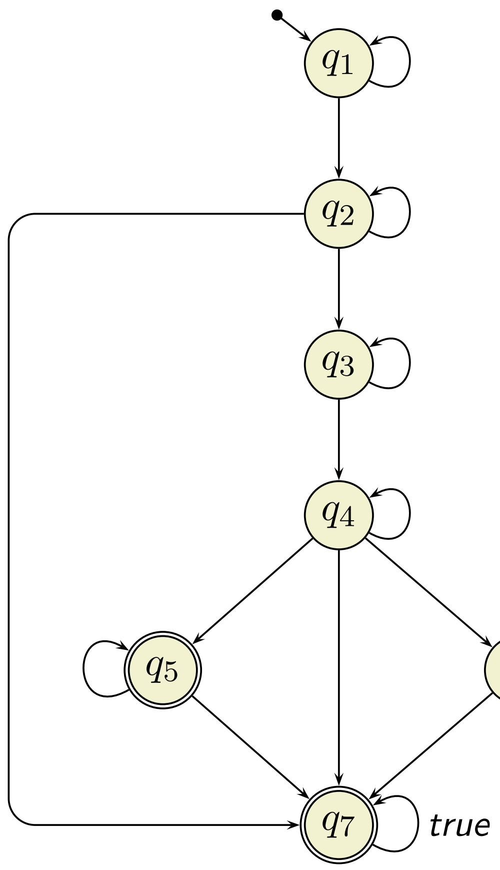
  Local invariant $(l_0, \iota_0, \phi, l_1, \iota_1)$ is **•-active** at q if and only if

  - $l_0 \prec l \prec l_1$, or
  - $l = l_0 \wedge \iota_0 = \bullet$, or
  - $l = l_1 \wedge \iota_1 = \bullet$

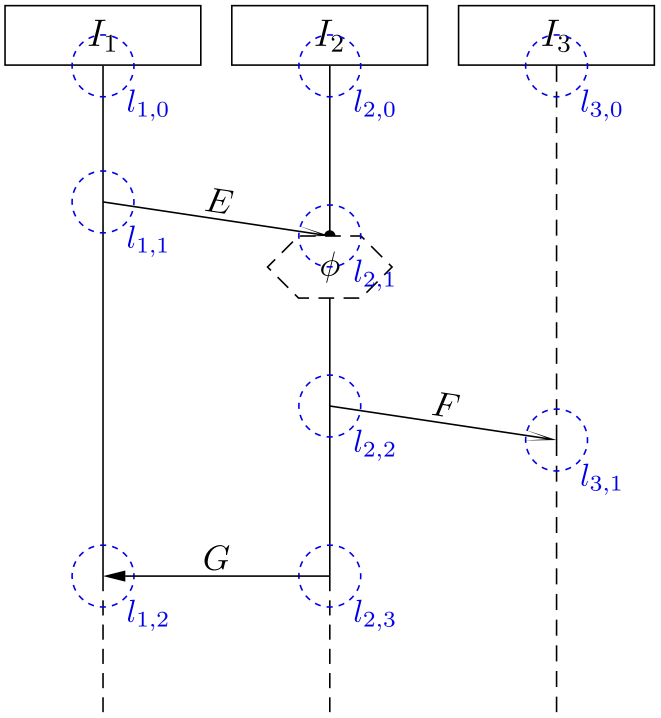
  for some front location l of cut (!) q.



Example



q_1
q_2
q_3
q_4
q_5
q_6
q_7
true



I_1
I_2
I_3
$l_{1,0}$
$l_{2,0}$
$l_{3,0}$
E
$l_{1,1}$
ϕ
$l_{2,1}$
F
$l_{2,2}$
$l_{3,1}$
G
$l_{1,2}$
$l_{2,3}$

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_{\mathscr{L}})$ consist of

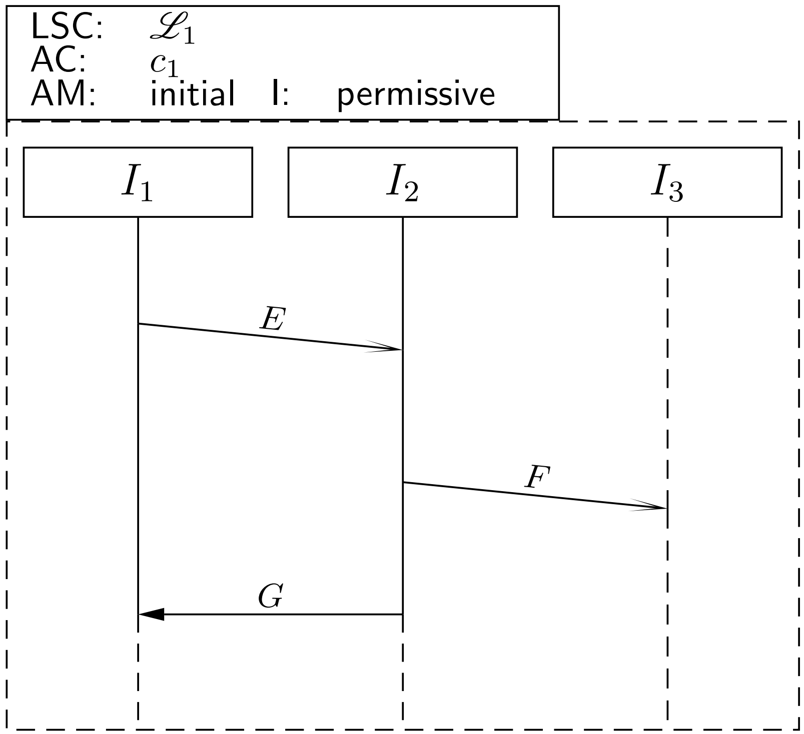
- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** $strict$ (otherwise called **permissive**)
- **activation mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathscr{L}} = \text{cold}$) or **universal** ($\Theta_{\mathscr{L}} = \text{hot}$).

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_{\mathscr{L}})$ consist of

- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** $strict$ (otherwise called **permissive**)
- **activation mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathscr{L}} = \text{cold}$) or **universal** ($\Theta_{\mathscr{L}} = \text{hot}$).

Concrete syntax:



Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_{\mathscr{L}})$ consist of

- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode** $am \in \{\mathsf{initial}, \mathsf{invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathscr{L}} = \mathsf{cold}$) or **universal** ($\Theta_{\mathscr{L}} = \mathsf{hot}$).

A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_{\mathscr{L}}$	$am = \mathsf{initial}$	$am = \mathsf{invariant}$
cold		
hot		

where $ac = ac_0 \wedge \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0) \wedge \psi^{\mathsf{Msg}}(\emptyset, C_0)$; C_0 is the minimal (or **instance heads**) cut.

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_{\mathscr{L}})$ consist of

- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode** *am* $\in \{\mathsf{initial}, \mathsf{invariant}\}$,
- **chart mode existential** $(\Theta_{\mathscr{L}} = \mathsf{cold})$ or **universal** $(\Theta_{\mathscr{L}} = \mathsf{hot})$.

A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_{\mathscr{L}}$	$am = \mathsf{initial}$	$am = \mathsf{invariant}$
cold	$\exists\, w \in W \bullet w^0 \models ac \wedge$ $w^0 \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/1 \in \mathit{Lang}(\mathcal{B}(\mathscr{L}))$	
hot		

where $ac = ac_0 \wedge \psi_{\mathsf{cold}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge \psi^{\mathsf{Msg}}(\emptyset, C_0)$; C_0 is the minimal (or **instance heads**) cut.

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_\mathscr{L})$ consist of

- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **chart mode existential** ($\Theta_\mathscr{L} = \text{cold}$) or **universal** ($\Theta_\mathscr{L} = \text{hot}$).

A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_\mathscr{L}$	$am = \text{initial}$	$am = \text{invariant}$
cold	$\exists\, w \in W \bullet w^0 \models ac \,\wedge$ $w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists\, w \in W\ \exists\, k \in \mathbb{N}_0 \bullet w^k \models ac \,\wedge$ $w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k{+}1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot		

where $ac = ac_0 \wedge \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0) \wedge \psi^{\text{Msg}}(\emptyset, C_0)$; C_0 is the minimal (or **instance heads**) cut.

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_\mathscr{L})$ consist of

- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** strict (otherwise called **permissive**)
- **activation mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **chart mode** **existential** ($\Theta_\mathscr{L} = \text{cold}$) or **universal** ($\Theta_\mathscr{L} = \text{hot}$).

A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_\mathscr{L}$	$am = \text{initial}$	$am = \text{invariant}$
cold	$\exists\, w \in W \bullet w^0 \models ac \wedge$ $w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in \mathit{Lang}(\mathcal{B}(\mathscr{L}))$	$\exists\, w \in W\ \exists\, k \in \mathbb{N}_0 \bullet w^k \models ac \wedge$ $w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k{+}1 \in \mathit{Lang}(\mathcal{B}(\mathscr{L}))$
hot	$\forall\, w \in W \bullet w^0 \models ac \implies$ $w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in \mathit{Lang}(\mathcal{B}(\mathscr{L}))$	

where $ac = ac_0 \wedge \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0) \wedge \psi^{\text{Msg}}(\emptyset, C_0)$; C_0 is the minimal (or **instance heads**) cut.

Finally: The LSC Semantics

A **full LSC** $\mathscr{L} = (((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta), ac_0, am, \Theta_{\mathscr{L}})$ consist of

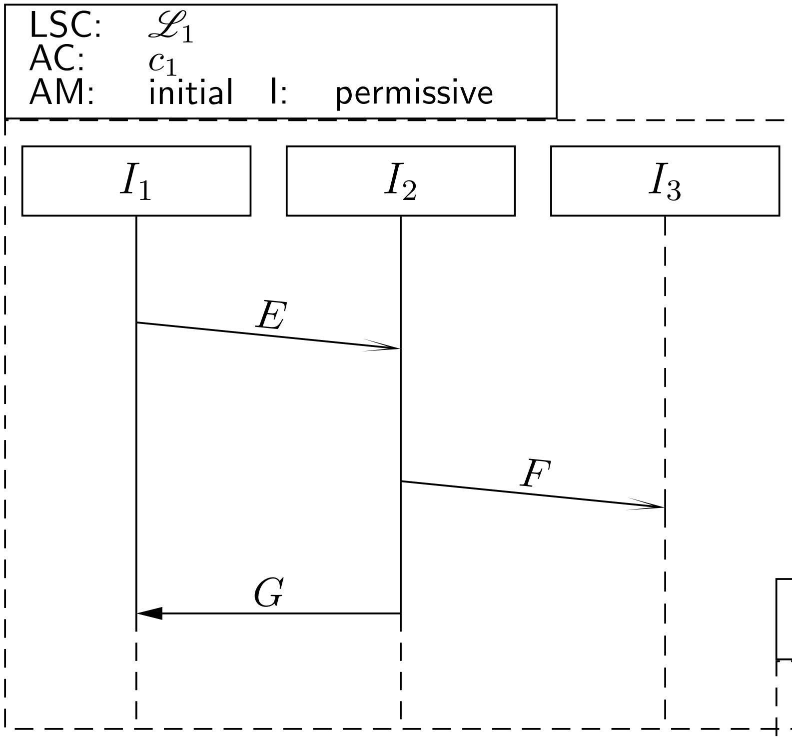
- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode** $am \in \{\mathsf{initial}, \mathsf{invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathscr{L}} = \mathsf{cold}$) or **universal** ($\Theta_{\mathscr{L}} = \mathsf{hot}$).

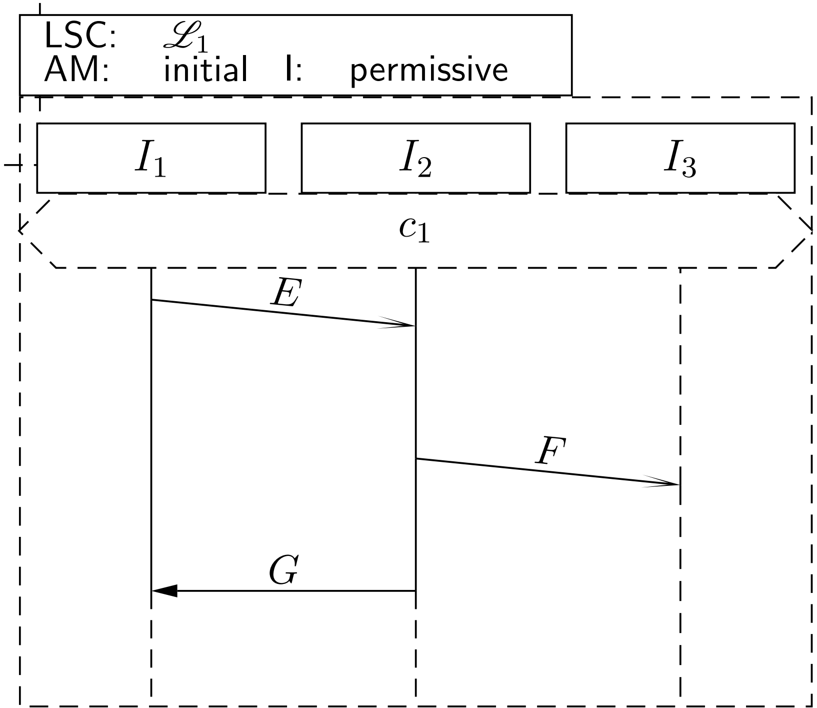
A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_{\mathscr{L}}$	$am = \mathsf{initial}$	$am = \mathsf{invariant}$
cold	$\exists\, w \in W \bullet w^0 \models ac \wedge w^0 \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists\, w \in W\ \exists\, k \in \mathbb{N}_0 \bullet w^k \models ac \wedge w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/k{+}1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall\, w \in W \bullet w^0 \models ac \implies w^0 \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall\, w \in W\ \forall\, k \in \mathbb{N}_0 \bullet w^k \models ac \implies w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/k{+}1 \in Lang(\mathcal{B}(\mathscr{L}))$

where $ac = ac_0 \wedge \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0) \wedge \psi^{\mathsf{Msg}}(\emptyset, C_0)$; C_0 is the minimal (or **instance heads**) cut.

Activation Condition





LSCs vs. Software

LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.

S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, i.e. the states are valuations of the conditions in C,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the events are of the form $E!$, $E?$.

LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.

S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, i.e. the states are valuations of the conditions in C,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the events are of the form $E!, E?$.

Construct letters by joining σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \ldots$$

LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.

S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, i.e. the states are valuations of the conditions in C,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the events are of the form $E!, E?$.

Construct letters by joining σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \ldots$$

We say S **satisfies** LSC $\mathscr{L}$ (e.g. universal, invariant), denoted by $S \models \mathscr{L}$, if and only if

$$\forall \pi \in [\![S]\!]\ \forall k \in \mathbb{N}_0 \bullet w(\pi)^k \models ac \implies w(\pi)^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \land w(\pi)/k+1 \in \mathit{Lang}(\mathcal{B}(\mathscr{L}))$$

LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.

S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, i.e. the states are valuations of the conditions in C,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the events are of the form $E!, E?$.

Construct letters by joining σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \ldots$$

We say S **satisfies** LSC $\mathscr{L}$ (e.g. universal, invariant), denoted by $S \models \mathscr{L}$, if and only if

$$\forall \pi \in [\![S]\!]\ \forall k \in \mathbb{N}_0 \bullet w(\pi)^k \models ac \implies w(\pi)^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w(\pi)/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$$

$\Theta_{\mathscr{L}}$	$am = \text{initial}$	$am = \text{invariant}$
cold	$\exists w \in W \bullet w^0 \models ac \wedge$ $w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists w \in W\ \exists k \in \mathbb{N}_0 \bullet w^k \models ac \wedge$ $w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall w \in W \bullet w^0 \models ac \implies$ $w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall w \in W\ \forall k \in \mathbb{N}_0 \bullet w^k \models ac \implies$ $w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.

S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, i.e. the states are valuations of the conditions in C,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the events are of the form $E!$, $E?$.

Construct letters by joining σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \ldots$$

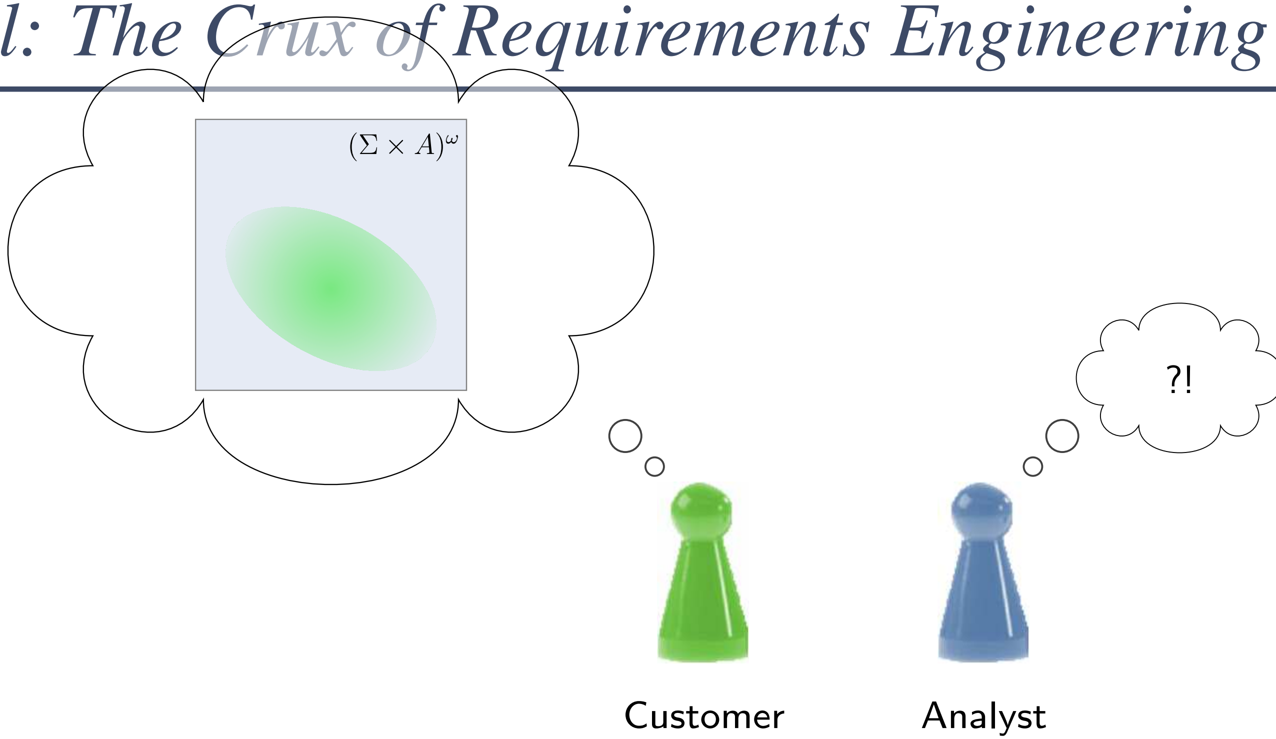
We say S **satisfies** LSC $\mathscr{L}$ (e.g. universal, invariant), denoted by $S \models \mathscr{L}$, if and only if

$$\forall\, \pi \in [\![S]\!]\ \forall\, k \in \mathbb{N}_0 \bullet w(\pi)^k \models ac \implies w(\pi)^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w(\pi)/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$$

$\Theta_{\mathscr{L}}$	am = initial	am = invariant
cold	$\exists\, w \in W \bullet w^0 \models ac \wedge$ $w^0 \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists\, w \in W\ \exists\, k \in \mathbb{N}_0 \bullet w^k \models ac \wedge$ $w^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall\, w \in W \bullet w^0 \models ac \implies$ $w^0 \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall\, w \in W\ \forall\, k \in \mathbb{N}_0 \bullet w^k \models ac \implies$ $w^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

Software S satisfies **a set of** LSCs $\mathscr{L}_1, \ldots, \mathscr{L}_n$ if and only if $S \models \mathscr{L}_i$ for all $1 \leq i \leq n$.

Recall: The Crux of Requirements Engineering



requirements analysis

One quite effective approach:
try to **approximate** the requirements with positive and negative **scenarios**.

- Dear customer, please describe example usages of the desired system.

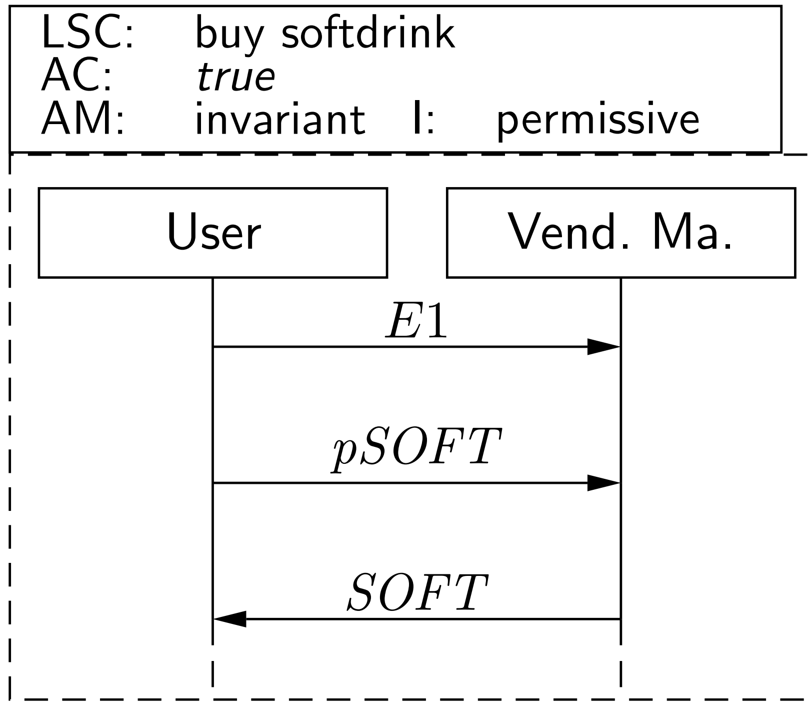
  "If the system is not at all able to do this, then it's not what I want."

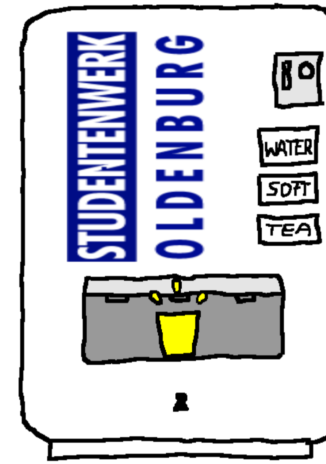
- Dear customer, please describe behaviour that the desired system must not show.

  "If the system does this, then it's not what I want."

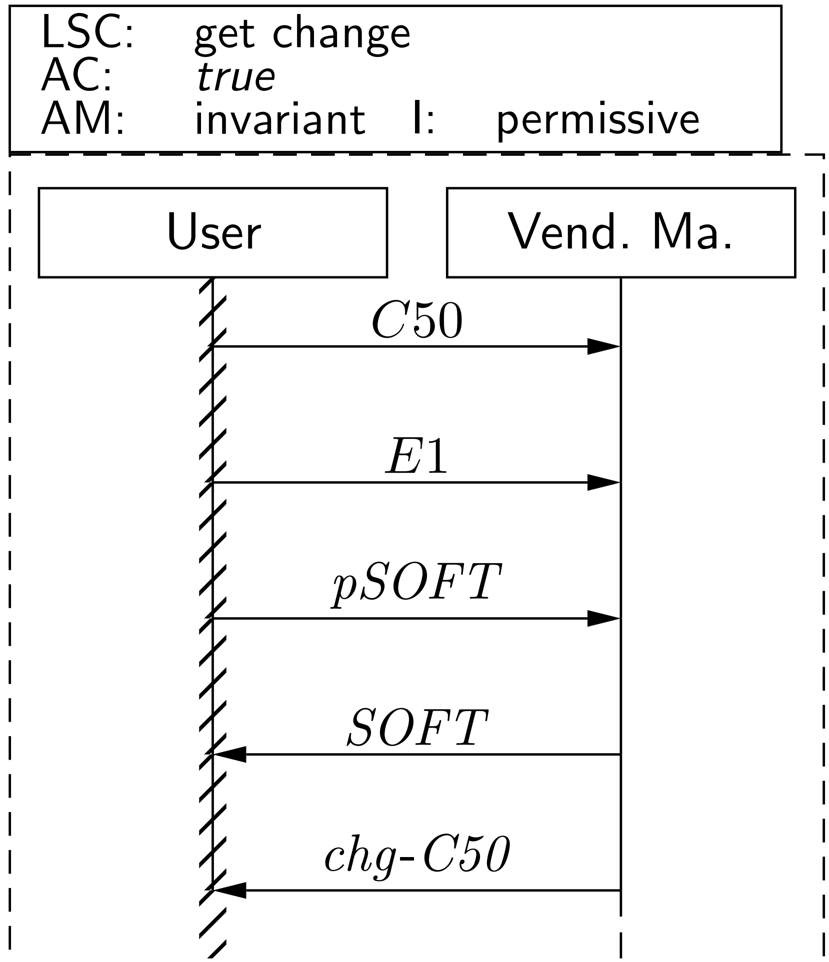
- From there on, refine and generalise:
  what about exceptional cases? what about corner-cases? etc.

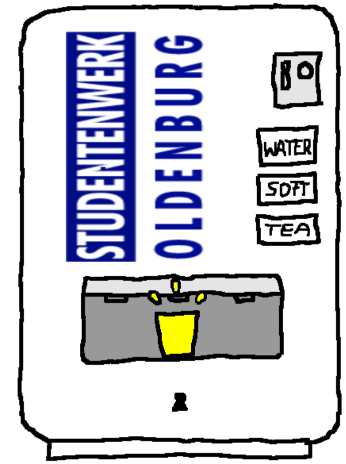
Example: Buy A Softdrink



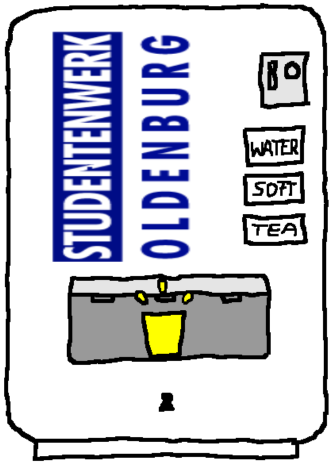


Example: Get Change

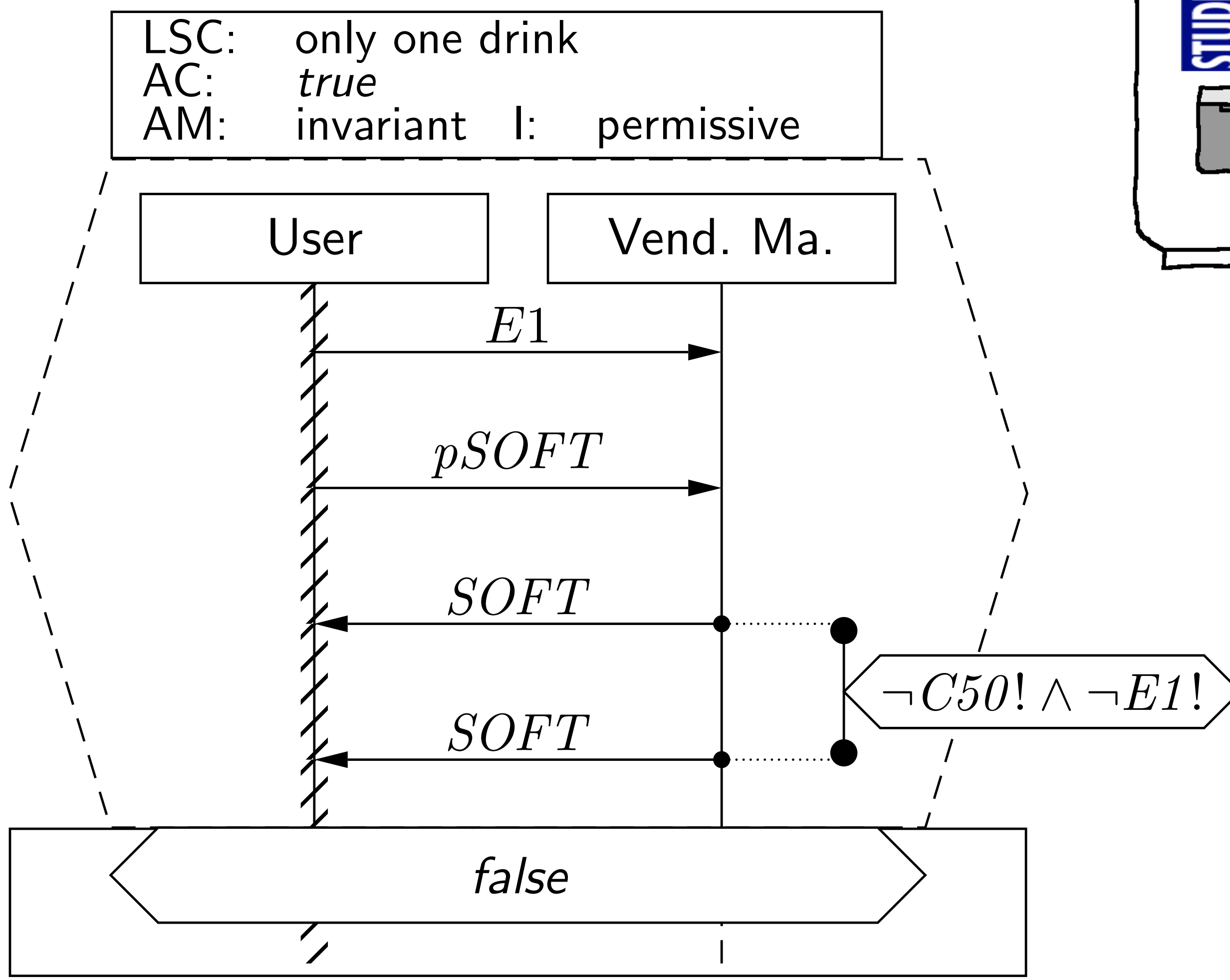


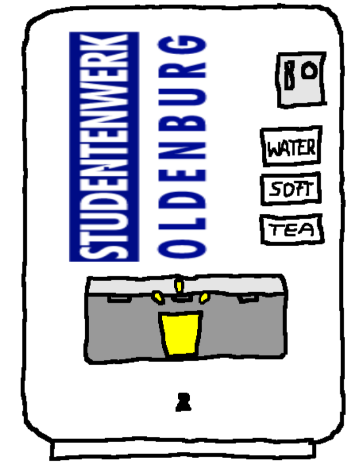


Example: Don't Give Two Drinks

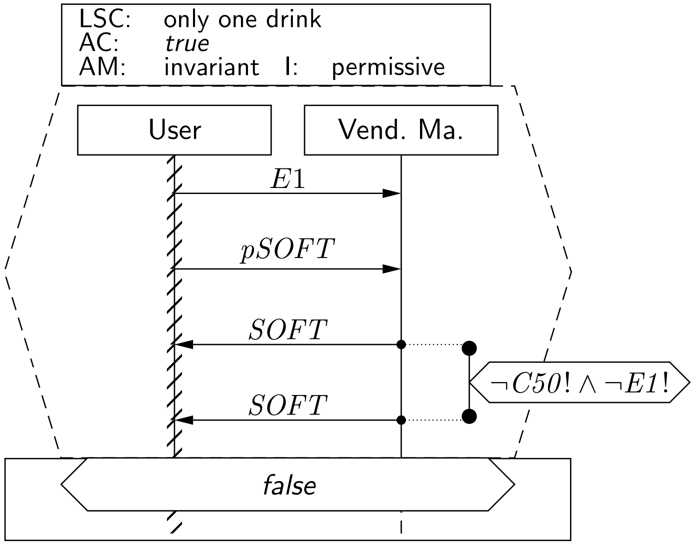


Example: Don't Give Two Drinks





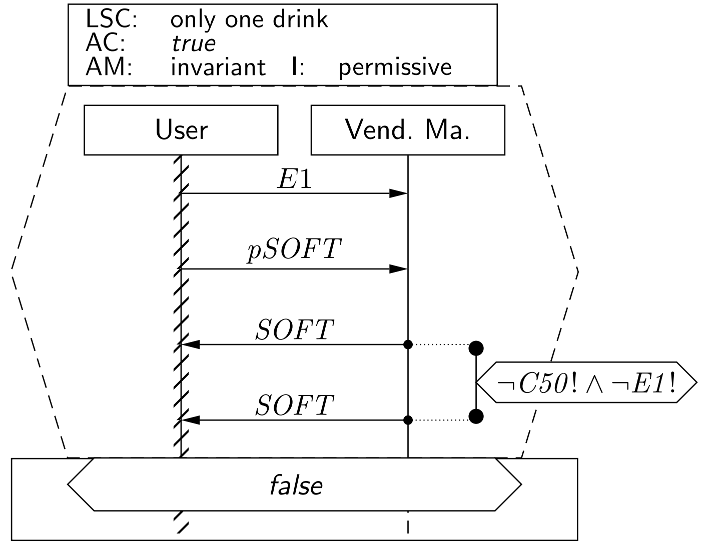
Pre-Charts



A **full LSC** $\mathscr{L} = (PC, MC, ac_0, am, \Theta_{\mathscr{L}})$ **actually** consist of

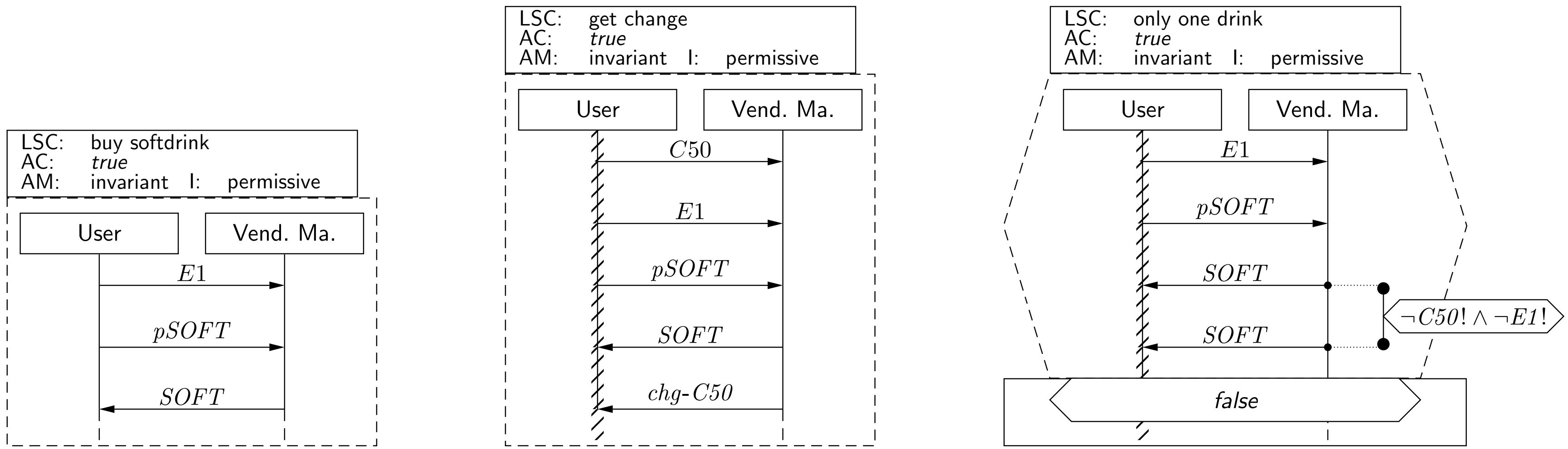
- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathsf{Msg}_P, \mathsf{Cond}_P, \mathsf{LocInv}_P, \Theta_P)$ (possibly empty),
- **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, \mathsf{Msg}_M, \mathsf{Cond}_M, \mathsf{LocInv}_M, \Theta_M)$ (non-empty),
- **activation condition** *ac* $\in \Phi(C)$, **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathscr{L}} = \text{cold}$) or **universal** ($\Theta_{\mathscr{L}} = \text{hot}$).

Pre-Charts Semantics



$\Theta_{\mathscr{L}}$	$am = \text{initial}$	$am = \text{invariant}$
cold	$\exists\, w \in W\ \exists\, m \in \mathbb{N}_0 \bullet w^0 \models ac$ $\wedge\, w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^P)$ $\wedge\, w/1, \dots, w/m \in \mathit{Lang}(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^M)$ $\wedge\, w/m+1 \in \mathit{Lang}(\mathcal{B}(MC))$	$\exists\, w \in W\ \exists\, k < m \in \mathbb{N}_0 \bullet w^k \models ac$ $\wedge\, w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^P)$ $\wedge\, w/k+1, \dots, w/m \in \mathit{Lang}(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^M)$ $\wedge\, w/m+1 \in \mathit{Lang}(\mathcal{B}(MC))$
hot	$\forall\, w \in W \bullet w^0 \models ac$ $\wedge\, w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^P)$ $\wedge\, w/1, \dots, w/m \in \mathit{Lang}(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0^M)$ $\implies w^{m+1} \models \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0^M)$ $\wedge\, w/m+1 \in \mathit{Lang}(\mathcal{B}(MC))$	$\forall\, w \in W\ \forall\, k \leq m \in \mathbb{N}_0 \bullet w^k \models ac$ $\wedge\, w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0^P)$ $\wedge\, w/k+1, \dots, w/m \in \mathit{Lang}(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0^M)$ $\implies w^{m+1} \models \psi_{\text{cold}}^{\text{Cond}}(\emptyset, C_0^M)$ $\wedge\, w/m+1 \in \mathit{Lang}(\mathcal{B}(MC))$

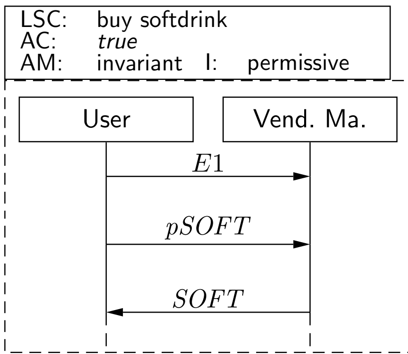
Note: Scenarios and Acceptance Test

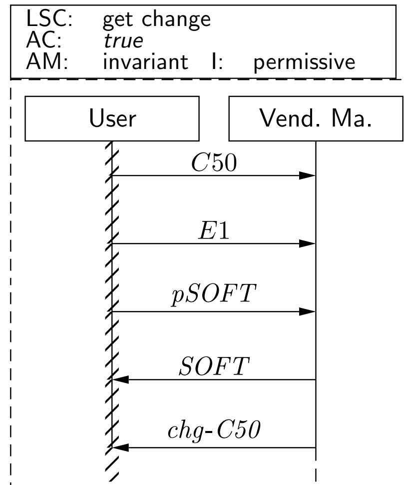


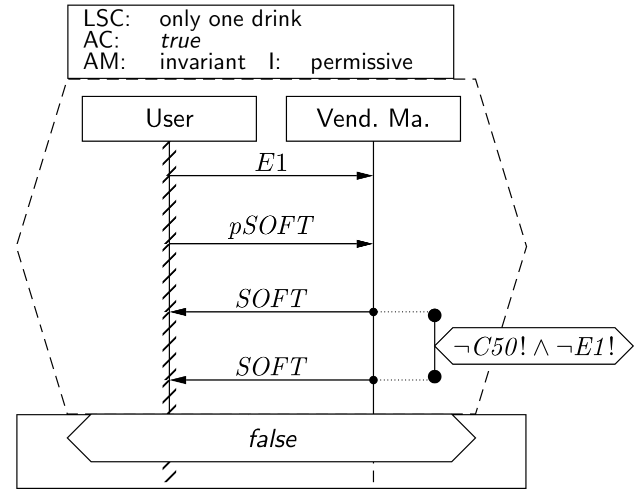
- **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!

  (∗: as well as (positive) scenarios in general, like use-cases)

Note: Scenarios and Acceptance Test





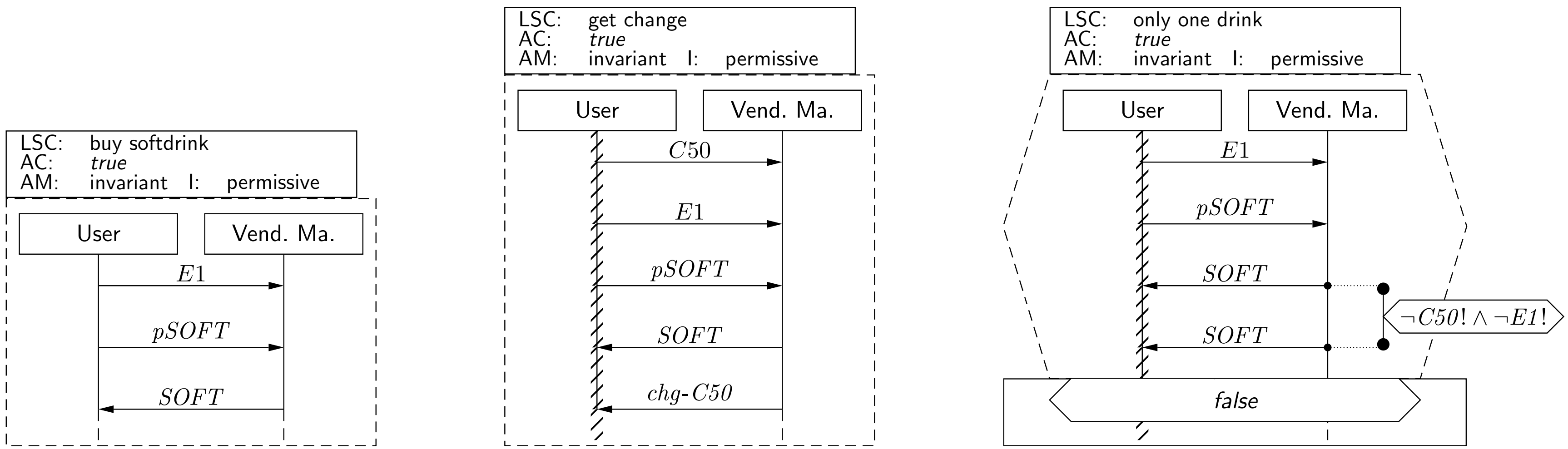


- **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!

  (∗: as well as (positive) scenarios in general, like use-cases)

- **Universal** LSCs (and negative/anti-scenarios) in general need **exhaustive analysis**!

Note: Scenarios and Acceptance Test



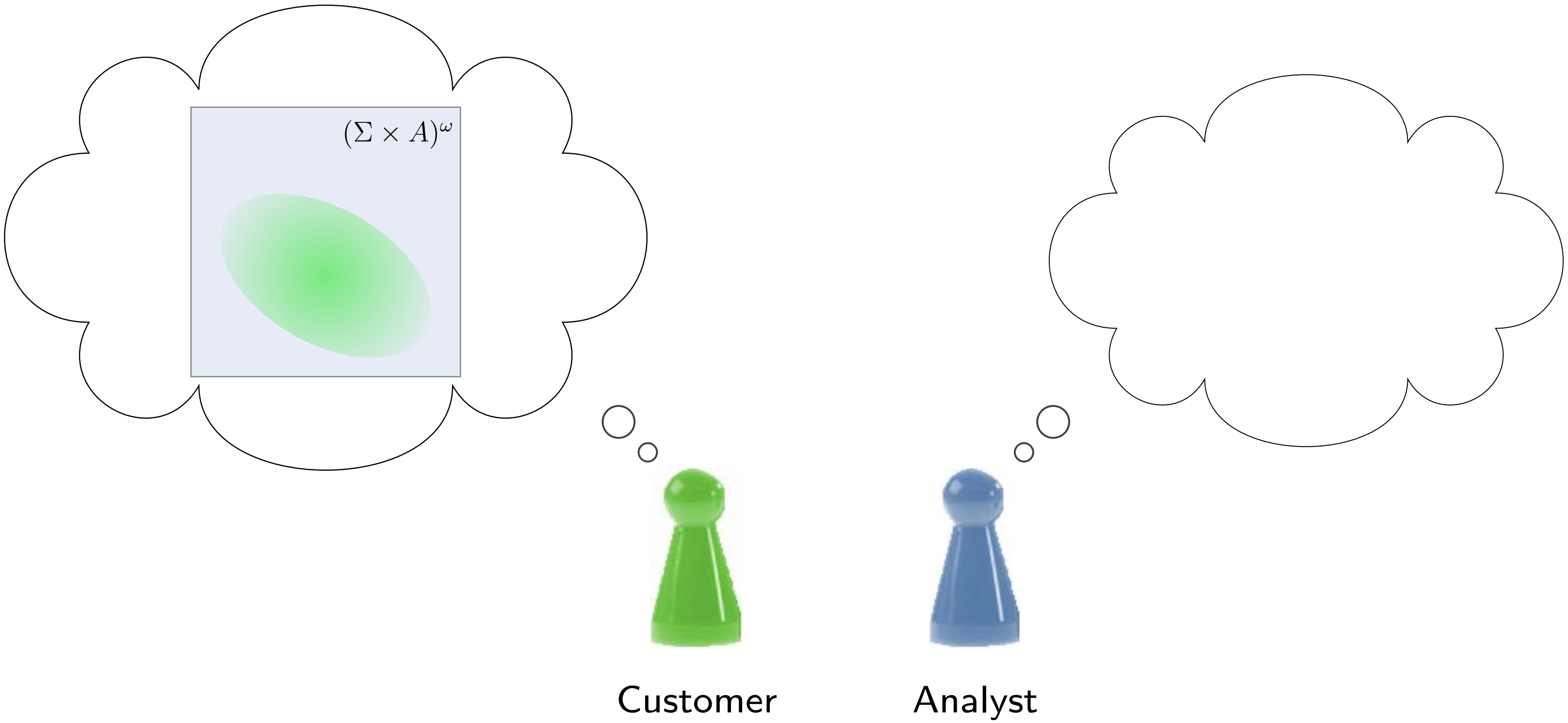
- **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!

  (∗: as well as (positive) scenarios in general, like use-cases)

- **Universal** LSCs (and negative/anti-scenarios) in general need **exhaustive analysis**!

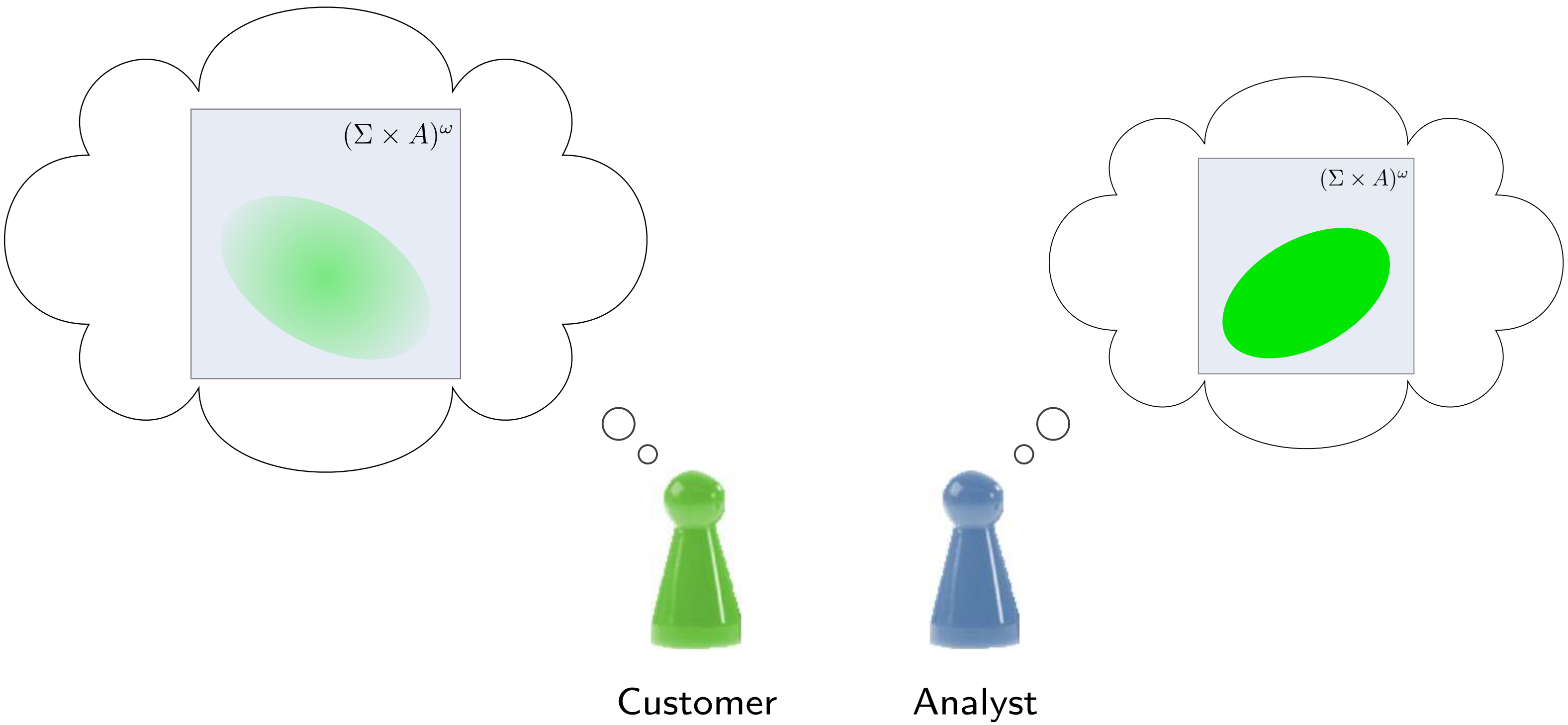
  (Because they require that the software **never ever** exhibits the unwanted behaviour.)

Strenghening Scenarios Into Requirements



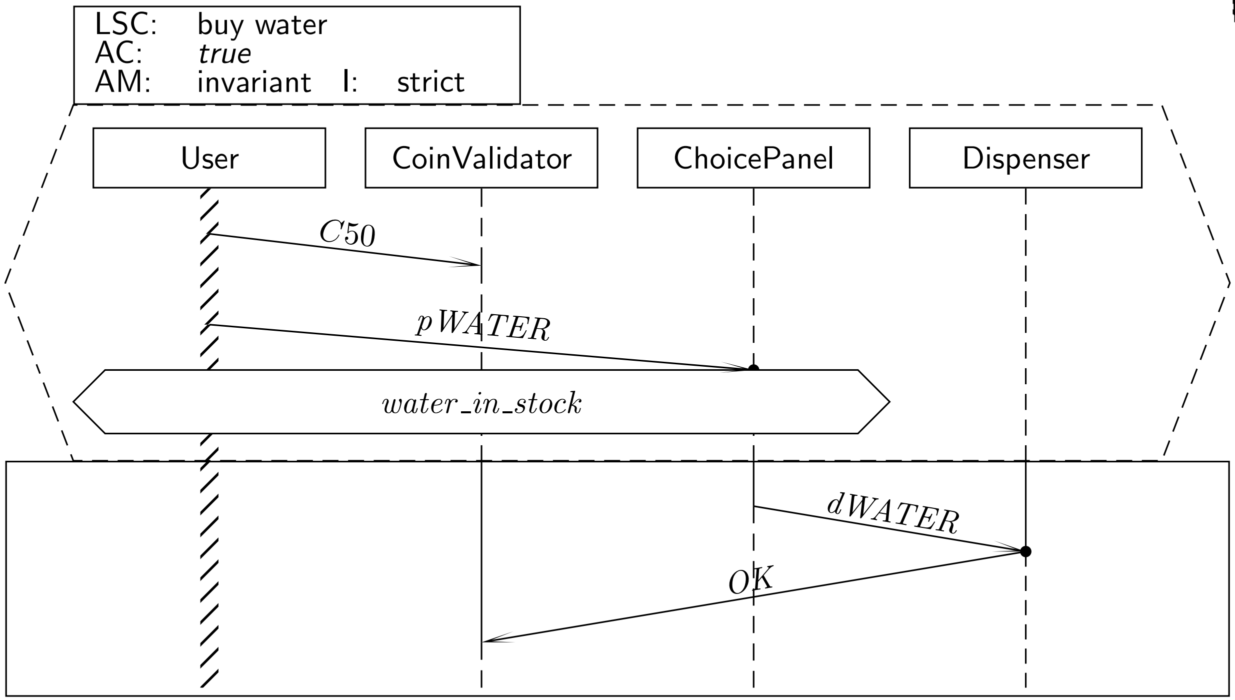
requirements analysis

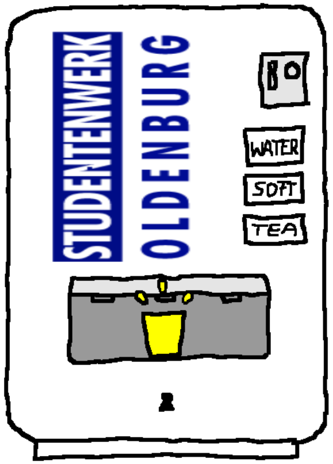
Strenghening Scenarios Into Requirements



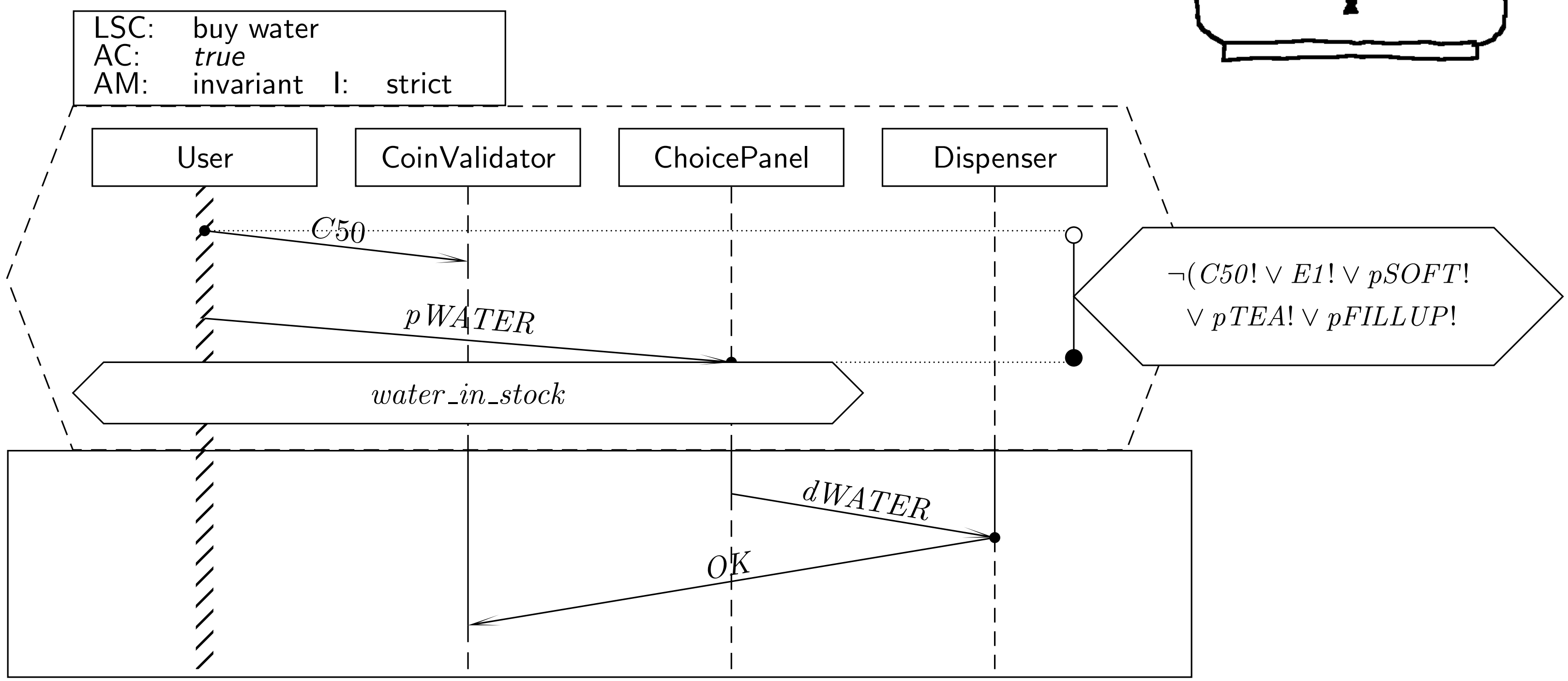
requirements analysis

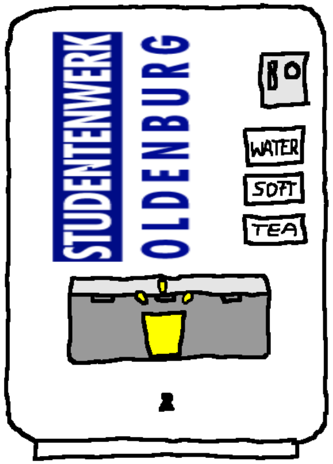
Universal LSC: Example



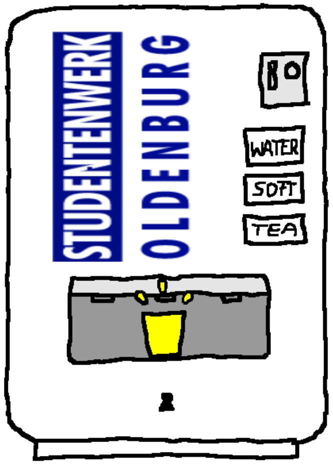


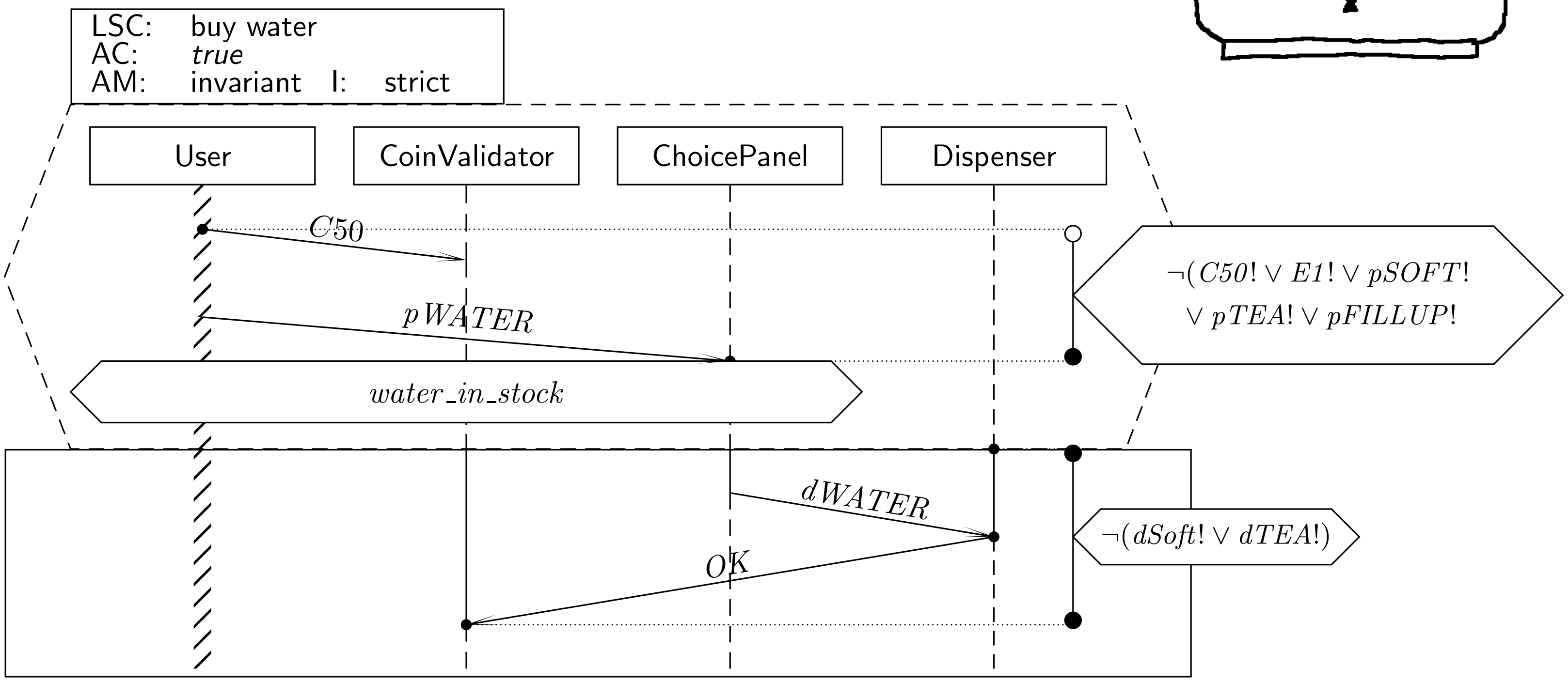
Universal LSC: Example



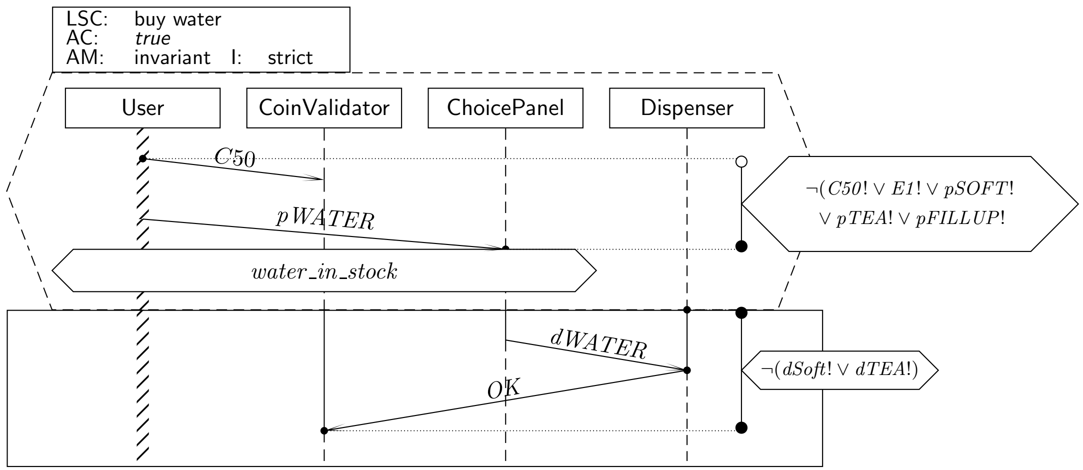


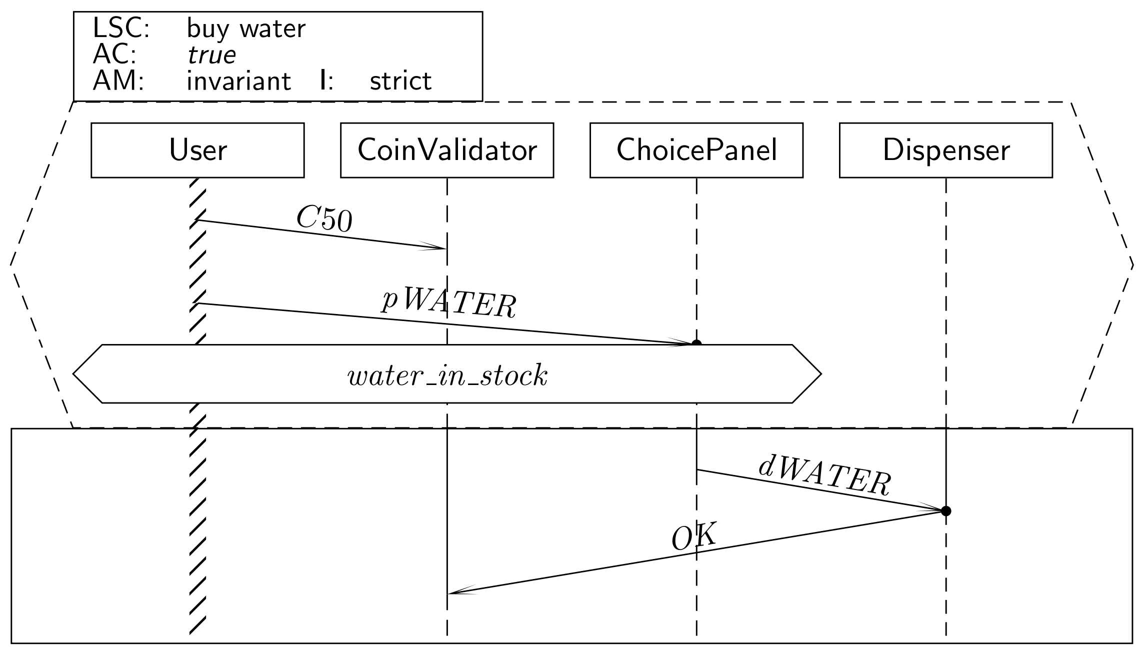
Universal LSC: Example



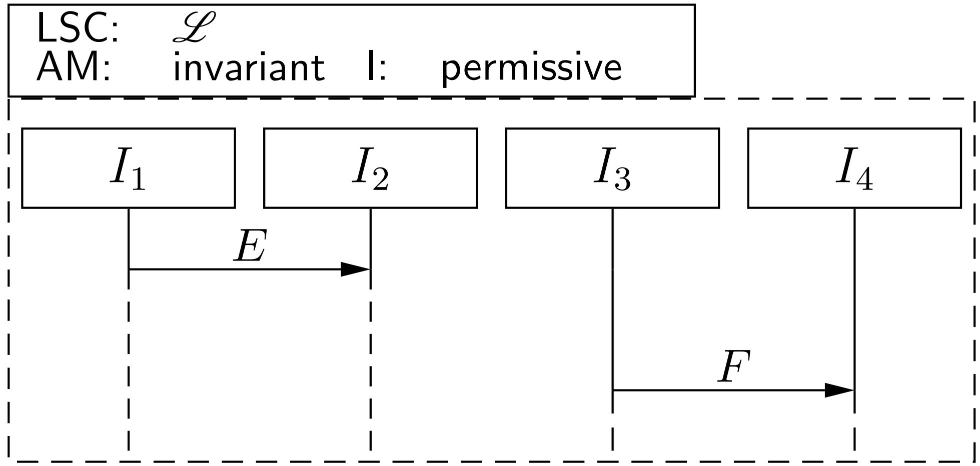


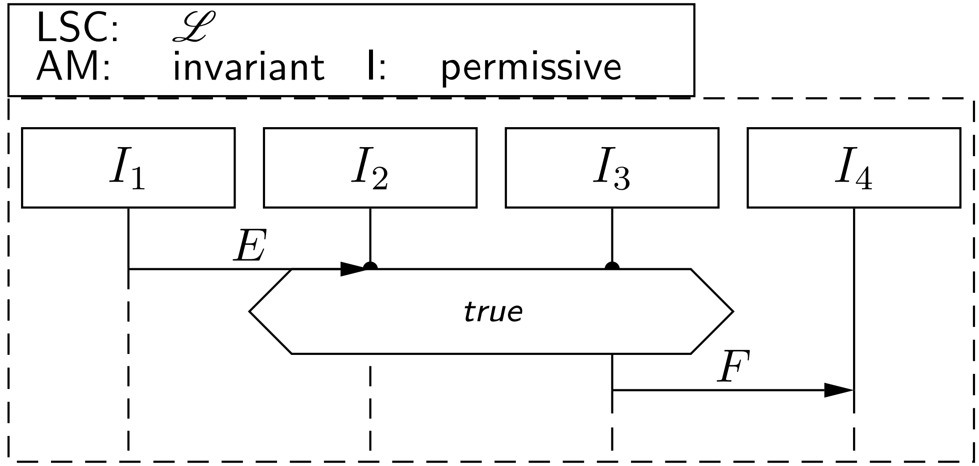
Shortcut: Forbidden Elements





Modelling Idiom: Enforcing Order





Requirements on Requirements Specifications

A **requirements specification** should be

- **correct**
  — it correctly represents the wishes/needs of the customer,
- **complete**
  — all requirements (existing in somebody's head, or a document, or . . .) should be present,
- **relevant**
  — things which are not relevant to the project should not be constrained,
- **consistent**, **free of contradictions**
  — each requirement is compatible with all other requirements; otherwise the requirements are **not realisable**,
- **neutral**, **abstract**
  — a requirements specification does not constrain the realisation more than necessary,
- **traceable**, **comprehensible**
  — the sources of requirements are documented, requirements are uniquely identifiable,
- **testable**, **objective**
  — the final product can **objectively** be checked for satisfying a requirement.

Requirements on LSC Specifications

- **correctness** is relative to "in the head of the customer" $\rightarrow$ still difficult;
- **complete**: we can at least define a kind of **relative completeness** in the sense of "did we cover all (exceptional) cases?";
- **relevant** also not analyseable **within** LSCs;
- **consistency** can formally be analysed!
- **neutral**/**abstract** is relative to the realisation $\rightarrow$ still difficult;
  But LSCs tend to support abstract specifications; specifying technical details is tedious.
- **traceable**/**comprehensible** are meta-properties, need to be established separately;
- a formal requirements specification, e.g. using LSCs, is immediately **objective**/**testable**.

For Decision Tables, we formally defined **additional quality criteria**:

- **uselessness**/**vacuity**,
- **determinism** may be desired,
- **consistency** wrt. domain model.

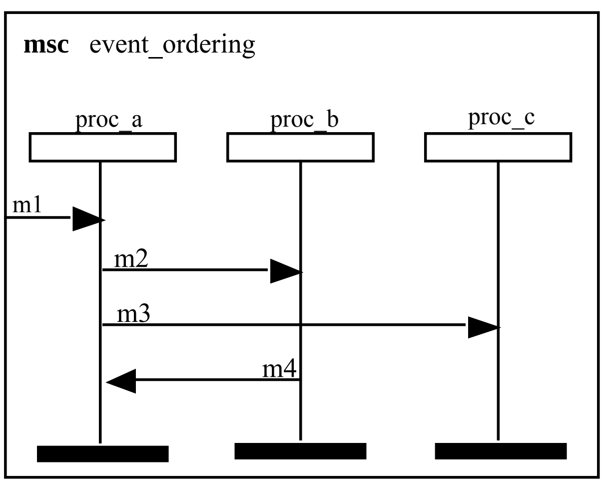
What about LSCs?

LSCs vs. MSCs

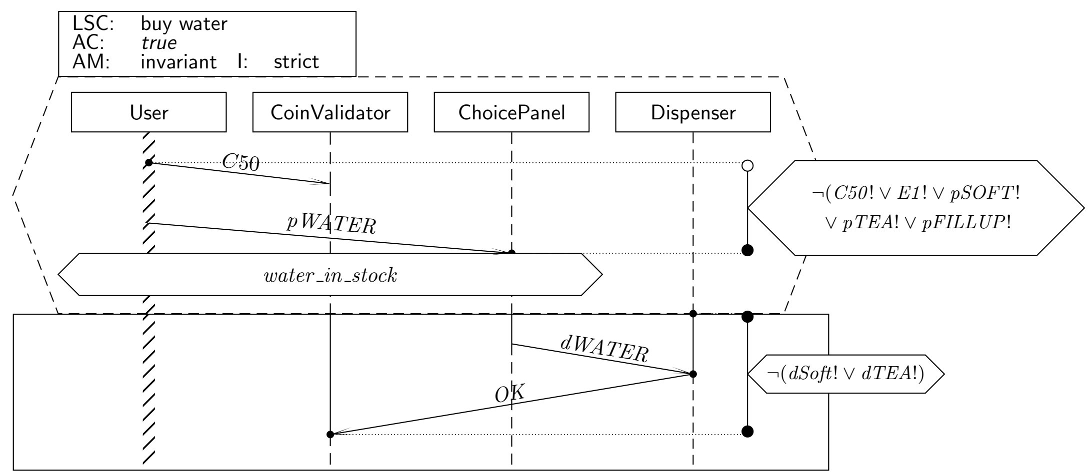
LSCs vs. MSCs

Recall: Most severe **drawbacks** of, e.g., MSCs:

- unclear **interpretation**: example scenario or invariant?
- unclear **activation**: what triggers the requirement?
- unclear **progress** requirement: must all messages be observed?
- **conditions** merely comments
- no means (in language) to express **forbidden scenarios**



(ITU-T, 2011)



Pushing It Even Further

(Harel and Marelly, 2003)

Requirements Engineering Wrap-Up

Recall: Software Specification Example

Alphabet:

- M – dispense cash only,
- C – return card only,
- $\begin{matrix}M\\C\end{matrix}$ – dispense cash and return card.

- **Customer 1** “don’t care”

$$\left(M.C \middle| C.M \middle| \begin{matrix}M\\C\end{matrix}\right)$$

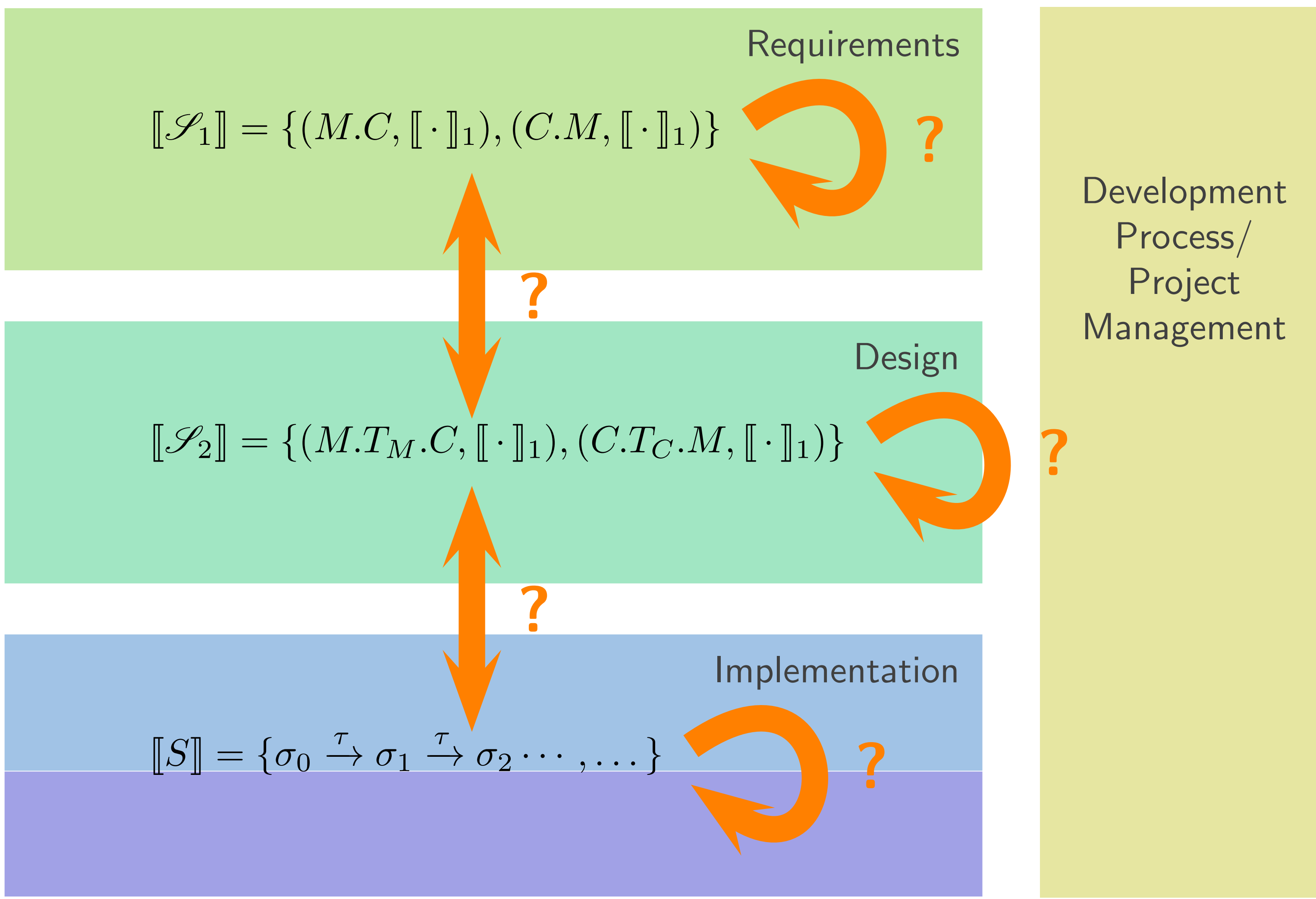
- **Customer 2** “you choose, but be consistent”

$$(M.C) \text{ or } (C.M)$$

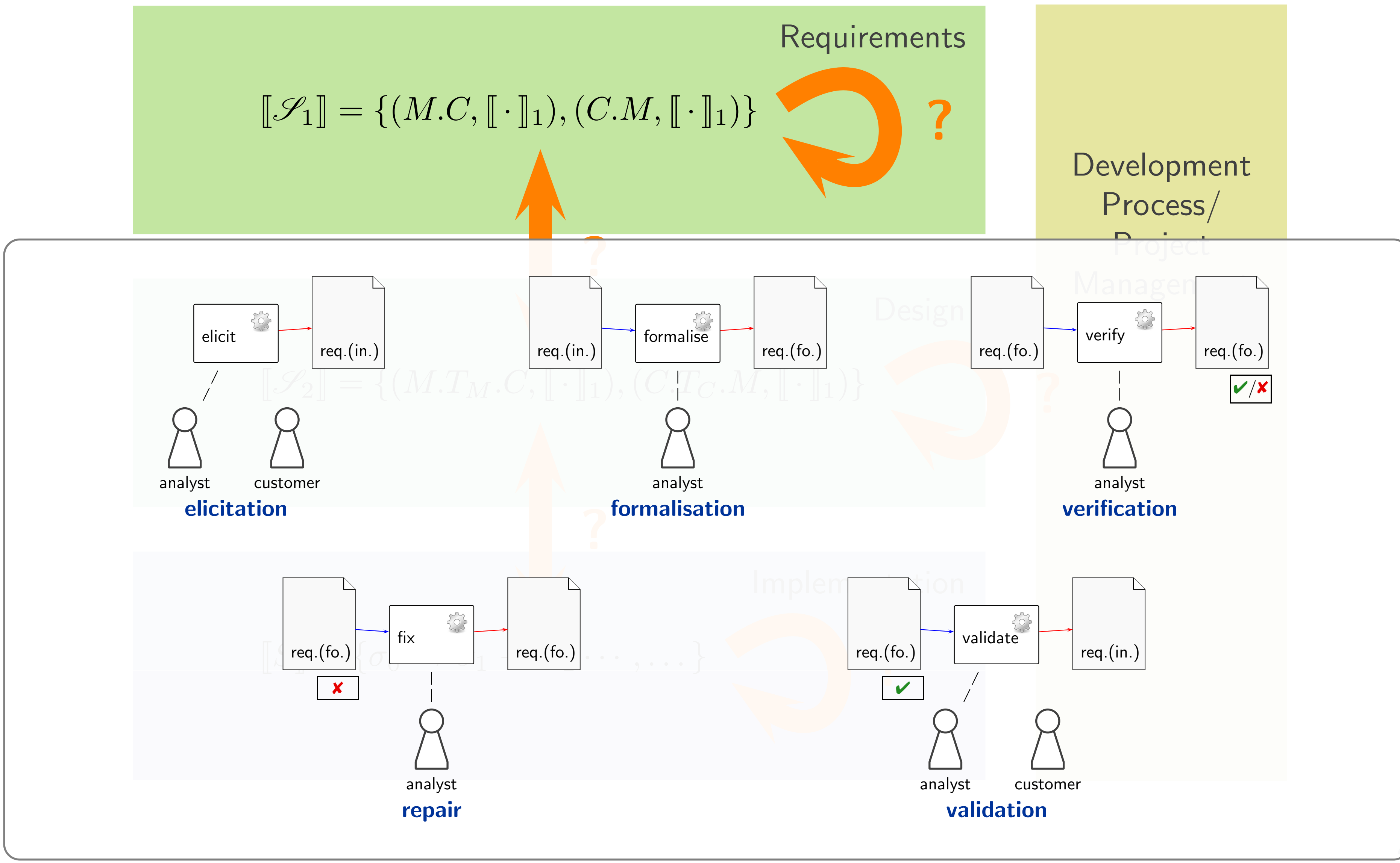
- **Customer 3** “consider human errors”

$$(C.M)$$

Recall: Formal Software Development



Recall: Formal Software Development



Final Remarks

One sometimes distinguishes:

- **Systems Engineering** (develop software for an embedded controller)

  Requirements typically stated in terms of **system observables** ("press WATER button"), needs to be mapped to terms of the software!

- **Software Engineering** (develop software which interacts with other software)

  Requirements stated in terms of the software.

We touched a bit of both, aimed at a general discussion.

- **Once again** (can it be mentioned too often?):

  Distinguish **domain elements** and **software elements**
  and (try to) keep them apart to avoid confusion.

A Classification of Software

Lehmann (Lehman, 1980; Lehman and Ramil, 2001) distinguishes three classes of software (my interpretation, my examples):

- **S-programs**: solve mathematical, abstract problems; can exactly (in particular formally) be specified; tend to be small; can be developed once and for all.

  Examples: sorting, compiler (!), compute π or $\sqrt{\cdot}$, cryptography, textbook examples, ...

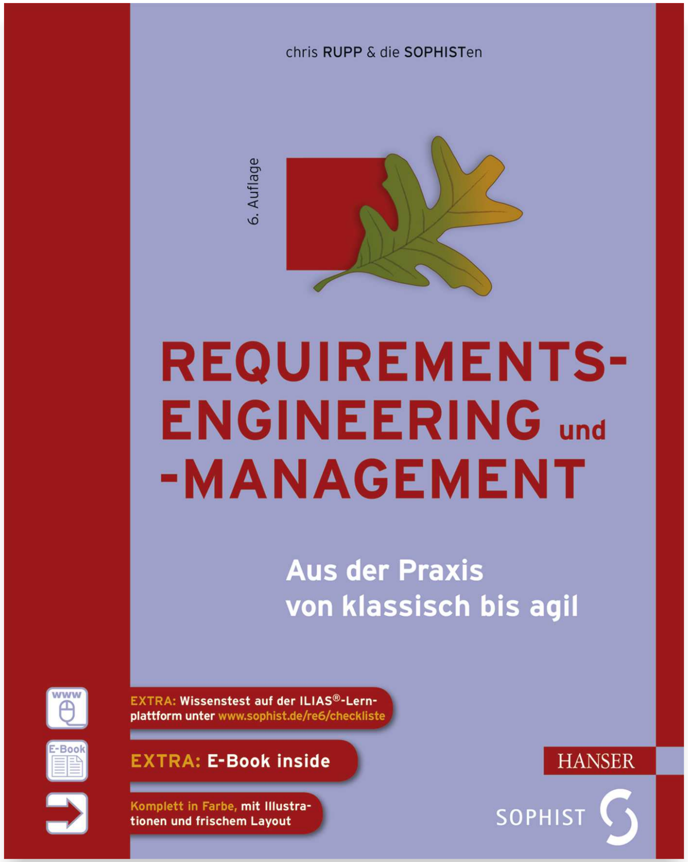
- **P-programs**: solve problems in the real world, e.g. read sensors and drive actors, may be in feedback loop; specification needs **domain model** (cf. Bjørner (2006), "A tryptich software development paradigm"); formal specification (today) possible, in terms of domain model, yet tends to be expensive

  Examples: cruise control, autopilot, traffic lights controller, plant automatisation, ...

- **E-programs**: embedded in socio-technical systems; in particular involve humans; specification often not clear, not even known; can grow huge; delivering the software induces new needs

  Examples: basically everything else; word processor, web-shop, game, smart-phone apps, ...

Literature Recommendation



(Rupp and die SOPHISTen, 2014)

References

References

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

ITU-T (2011). *ITU-T Recommendation Z.120: Message Sequence Chart (MSC)*, 5 edition.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rupp, C. and die SOPHISTen (2014). *Requirements-Engineering und -Management*. Hanser, 6th edition.