Softwaretechnik / Software-Engineering

Lecture 8: Use Cases and Scenarios

2017-06-01

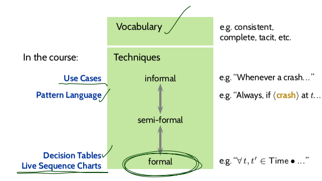
Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Requirements Engineering: Content

Structure of Topic Areas

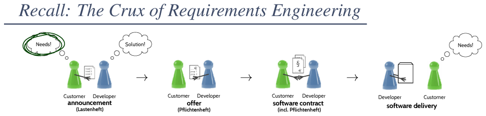
Example: Requirements Engineering



Content

Scenarios

Recall: The Crux of Requirements Engineering

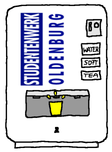


One quite effective approach:

try to **approximate** the requirements with positive and negative **scenarios**.

- Dear customer, please describe example usages of the desired system.
  Customer intuition: "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.
  Customer intuition: "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.
- Prominent early advocate: **OOSE** (Jacobson, 1992).

Example: Vending Machine



- **Positive scenario**: Buy a Softdrink
  - (i) Insert one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Get a softdrink.
- **Positive scenario**: Get Change
  - (i) Insert one 50 cent and one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Get a softdrink.
  - (iv) Get 50 cent change.
- **Negative scenario**: A Drink for Free
  - (i) Insert one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Do not insert any more money.
  - (iv) Get **two** softdrinks.

Notations for Scenarios

- The idea of **scenarios** (sometimes without **negative** or **anti-scenarios**) (re-)occurs in many process models or software development approaches.
- In the following, we will discuss two-and-a-half notations (in increasing formality):
  - **User Stories** (part of **Extreme Programming**)
  - **Use Cases** and Use Case Diagrams (**OOSE**)
  - **Sequence Diagrams** (here: **Live Sequence Charts** (Damm and Harel, 2001))

User Stories

User Stories (Beck, 1999)

*"A User Story is a **concise, written description** of a **piece of functionality** that will be **valuable to a user** (or owner) of the software."*

Per **user story**, use one **file card** with the user story, e.g. following the pattern:

As a [role] I want [something] so that [benefit].

and in addition:

- **unique identifier** (e.g. unique number),
- **priority** (from 1 (highest) to 10 (lowest)) **assigned by customer**,
- **effort**, **estimated by developers**,
- back side of file card: (acceptance) **test case(s)**, i.e., how to tell whether the user story has been realised.

Proposed card layout (front side):

priority	unique identifier, name	estimation
	As a [role] I want [something] so that [benefit].	
risk		real effort

Natural Language Patterns

Natural language requirements can be (tried to be) written as an instance of the **pattern** "$\langle A \rangle$ $\langle B \rangle$ $\langle C \rangle$ $\langle D \rangle$ $\langle E \rangle$ $\langle F \rangle$." (German grammar) where

A	clarifies when and under what conditions the activity takes place
B	is MUST (obligation), SHOULD (wish), or WILL (intention); also: MUST NOT (forbidden)
C	is either "the system" or the concrete name of a (sub-)system
D	one of three possibilities: • "does", description of a system activity, • "offers", description of a function offered by the system to somebody, • "is able if", usage of a function offered by a third party, under certain conditions
E	extensions, in particular an object
F	the actual process word (what happens)

(Rupp and die SOPHISTen, 2009)

Example:

After office hours ($= A$), the system ($= C$) should ($= B$) offer to the operator ($= D$) a backup ($= F$) of all new registrations to an external medium ($= E$).

33/37

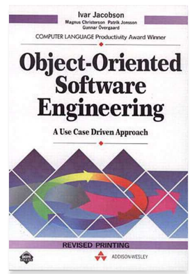
As a [role] I want [something] so that [benefit].
risk | real effort

User Stories: Discussion

- ✔ easy to create, small units
- ✔ close contact to customer
- ✔ objective / testable: by fixing test cases early
- ✘ may get difficult to keep overview over whole system to be developed → maybe best suited for changes / extensions (after first iteration).
- ✘ not designed to cover non-functional requirements and restrictions
- ✘ agile spirit: strong dependency on competent developers
- ✘ estimation of effort may be difficult

(Balzert, 2009)

Use Cases



Use Case: Definition

use case – A **sequence of interactions** between an actor (or actors) and a system triggered by a specific actor, **which produces a result** for an actor. (Jacobson, 1992)

More precisely:

- A use case has **participants**: the **system** and at least one **actor**.
- **Actor**: an actor represents what interacts with the system.
  - An actor is a **role**, which a **user** or an **external system** may assume when interacting with the system under design.
  - Actors are not part of the system, thus they are **not described in detail**.
  - Actions of actors are **non-deterministic** (possibly constrained by domain model).
- A use case is triggered by a **stimulus** as input by the **main actor**.
- A use case is **goal oriented**, i.e. the main actor wants to reach a particular goal.
- A use case describes **all interactions** between the system and the participating actors that are needed to achieve the goal (or fail to achieve the goal for reasons).
- A use case **ends** when the desired goal is achieved, or when it is clear that the desired goal cannot be achieved.

Use Case Example: ATM Authentication

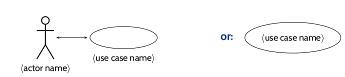
name	Authentication
goal	the client wants access to the ATM
pre-condition	the ATM is operational, the welcome screen is displayed, card and PIN of client are available
post-condition	client accepted, services of ATM are offered
post-cond. in exceptional case	access denied, card returned or withheld, welcome screen displayed
actors	client (main actor), bank system
open questions	none
normal case	1. client inserts card 2. ATM read card, sends data to bank system 3. bank system checks validity 4. ATM shows PIN screen 5. client enters PIN 6. ATM reads PIN, sends to bank system 7. bank system checks PIN 8. ATM accepts and shows main menu
exception case 2a	card not readable 2a.1 ATM displays "card not readable" 2a.2 ATM returns card 2a.3 ATM shows welcome screen

exc. case 2b	card readable, but not ATM card
exc. case 2c	no connection to bank system ✓
exc. case 3a	card not valid or disabled ✓
exc. case 5a	client cancels ✓
exc. case 5b	client doesn't react within 5 s ✓
exc. case 6a	no connection to bank system ✓
exc. case 7a	first or second PIN wrong
exc. case 7b	third PIN wrong

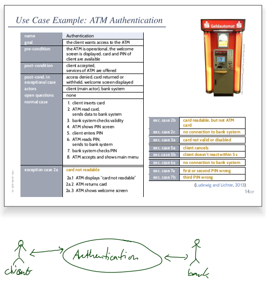
(Ludewig and Lichter, 2013)

Use Case Diagrams

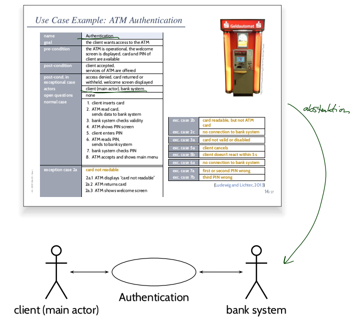
Use Case Diagrams: Basic Building Blocks



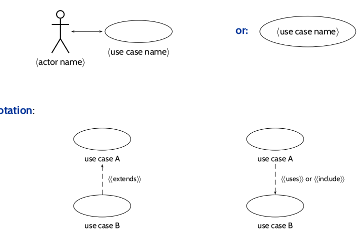
Example: Use Case Diagram of the ATM Use Case



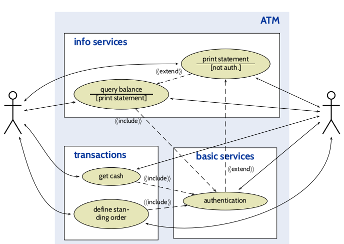
Example: Use Case Diagram of the ATM Use Case



Use Case Diagrams: More Building Blocks

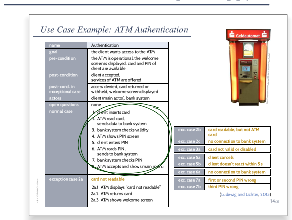


Use Case Diagram: Bigger Examples



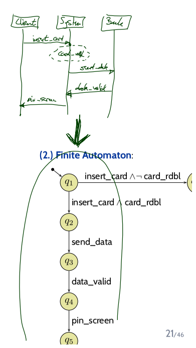
(Ludewig and Lichter, 2013)

Customer *and* Developer Happy?



(1.) Observables:

- event **insert_card**
- condition **card_rdbl**
- event **send_data**
- event **data_valid**
- event **pin_screen**



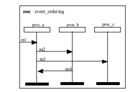
Content

- **User Stories**
- **Use Cases**
  - Use Case Diagrams
- **Sequence Diagrams**
  - A Brief History
  - **Live Sequence Charts**
    - Syntax:
      - Elements, Locations,
    - Towards Semantics:
      - Cuts
      - Firedsets

Sequence Diagrams

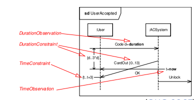
A Brief History of Sequence Diagrams

- **Message Sequence Charts**,
  ITU standardized in different versions (ITU Z.120, 1st edition: 1993); often accused of lacking a formal semantics.



(ITU-T, 2011)

- **Sequence Diagrams** of UML 1.x
  (one of three main authors: I. Jacobson)
- **SDs of UML 2.x** address **some** issues, yet the standard exhibits unclarities and even contradictions
  (Harel and Maoz, 2007; Störrle, 2003)

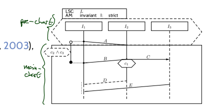


(OMG, 2007)

- For the lecture, we consider

  Live Sequence Charts (LSCs)

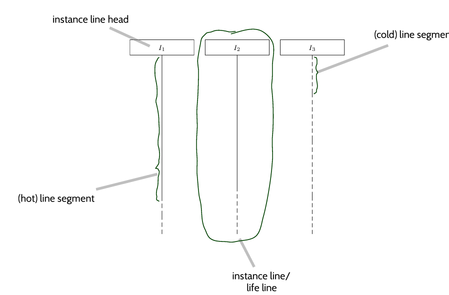
  (Damm and Harel, 2001; Klose, 2003; Harel and Marelly, 2003),
  LSCs have a common fragment with UML 2.x SDs:
  (Harel and Maoz, 2007).



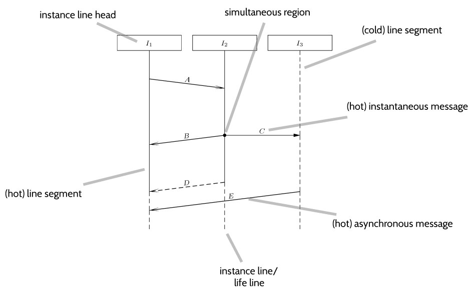
Live Sequence Charts: Syntax (Body)

LSC Body Building Blocks

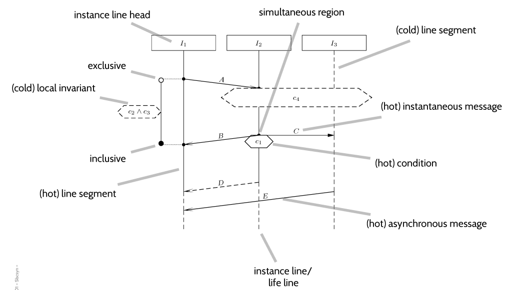
LSC Body Building Blocks



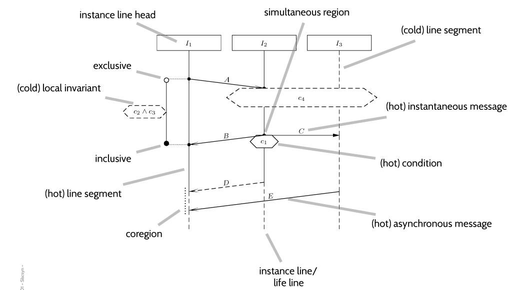
LSC Body Building Blocks



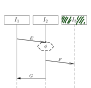
LSC Body Building Blocks



LSC Body Building Blocks

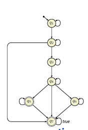


The Plan: A Formal Semantics for a Visual Formalism



concrete syntax
(diagram)

$((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$
abstract syntax



semantics
(Büchi automaton)

LSC Body: Abstract Syntax

Definition. [*LSC Body*]
Let $\mathcal{E}$ be a set of events and $\mathcal{C}$ a set of atomic propositions, $\mathcal{E} \cap \mathcal{C} = \emptyset$.
An LSC body over $\mathcal{E}$ and $\mathcal{C}$ is a tuple

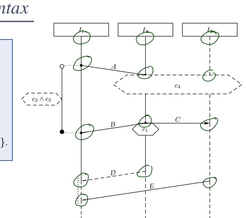
$$((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$$

where

- $\mathcal{L}$ is a finite, non-empty of **locations** with
  - a **partial order** $\preceq \subseteq \mathcal{L} \times \mathcal{L}$,
  - a symmetric **simultaneity relation** $\sim \subseteq \mathcal{L} \times \mathcal{L}$ disjoint with $\preceq$, i.e. $\preceq \cap \sim = \emptyset$,
- $\mathcal{I} = \{I_1, \dots, I_n\}$ is a partitioning of $\mathcal{L}$; elements of $\mathcal{I}$ are called **instance line**,
- $\mathsf{Msg} \subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L}$ is a set of **messages** with $(l, E, l') \in \mathsf{Msg}$ only if $(l, l') \in \prec \cup \sim$; message (l, E, l') is called **instantaneous** iff $l \sim l'$ and **asynchronous** otherwise,
- $\mathsf{Cond} \subseteq (2^{\mathcal{L}} \setminus \emptyset) \times \Phi(\mathcal{C})$ is a set of **conditions** with $(L, \phi) \in \mathsf{Cond}$ only if $l \sim l'$ for all $l \neq l' \in L$,
- $\mathsf{LocInv} \subseteq \mathcal{L} \times \{\circ, \bullet\} \times \Phi(\mathcal{C}) \times \mathcal{L} \times \{\circ, \bullet\}$ is a set of **local invariants** with $(l, \iota, \phi, l', \iota') \in \mathsf{LocInv}$ only if $l \prec l'$, $\circ$: exclusive, $\bullet$: inclusive,
- $\Theta : \mathcal{L} \cup \mathsf{Msg} \cup \mathsf{Cond} \cup \mathsf{LocInv} \to \{\mathrm{hot}, \mathrm{cold}\}$ assigns to each location and each element a **temperature**.

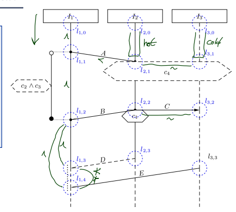
From Concrete to Abstract Syntax

- locations $\mathcal{L}$,
- $\preceq \subseteq \mathcal{L} \times \mathcal{L}$, $\quad \sim \subseteq \mathcal{L} \times \mathcal{L}$
- $\mathcal{I} = \{I_1, \dots, I_n\}$,
- $\mathsf{Msg} \subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L}$,
- $\mathsf{Cond} \subseteq (2^{\mathcal{L}} \setminus \emptyset) \times \Phi(\mathcal{C})$
- $\mathsf{LocInv} \subseteq \mathcal{L} \times \{\circ, \bullet\} \times \Phi(\mathcal{C}) \times \mathcal{L} \times \{\circ, \bullet\}$,
- $\Theta : \mathcal{L} \cup \mathsf{Msg} \cup \mathsf{Cond} \cup \mathsf{LocInv} \to \{\mathrm{hot}, \mathrm{cold}\}$.



From Concrete to Abstract Syntax

- locations $\mathcal{L}$,
- $\preceq \subseteq \mathcal{L} \times \mathcal{L}$, $\sim \subseteq \mathcal{L} \times \mathcal{L}$
- $\mathcal{I} = \{I_1, \dots, I_n\}$,
- Msg $\subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L}$,
- Cond $\subseteq (2^{\mathcal{C}} \setminus \emptyset) \times \Phi(\mathcal{C})$
- LocInv $\subseteq \mathcal{L} \times \{\circ, \bullet\} \times \Phi(\mathcal{C}) \times \mathcal{L} \times \{\circ, \bullet\}$,
- $\Theta : \mathcal{L} \cup$ Msg $\cup$ Cond $\cup$ LocInv $\to$ {hot, cold}.



- $\mathcal{L} = \{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, \; l_{1,2}, l_{1,4}, \; l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}, \; l_{3,0}, l_{3,1}, l_{3,2}, l_{3,3}\}$

LSC Body: Abstract Syntax

Definition. [*LSC Body*]
Let $\mathcal{E}$ be a set of events and $\mathcal{C}$ a set of atomic propositions, $\mathcal{E} \cap \mathcal{C} = \emptyset$.
An LSC body over $\mathcal{E}$ and $\mathcal{C}$ is a tuple

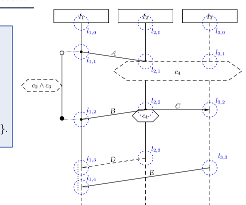
$$((\mathcal{L}, \preceq, \sim), \mathcal{I}, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta)$$

where

- $\mathcal{L}$ is a finite, non-empty of locations with
  - a partial order $\preceq \subseteq \mathcal{L} \times \mathcal{L}$,
  - a symmetric simultaneity relation $\sim \subseteq \mathcal{L} \times \mathcal{L}$ disjoint with $\preceq$, i.e. $\preceq \cap \sim = \emptyset$,
- $\mathcal{I} = \{I_1, \dots, I_n\}$ is a partitioning of $\mathcal{L}$; elements of $\mathcal{I}$ are called instance line,
- Msg $\subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L}$ is a set of messages with $(l, E, l') \in$ Msg only if $(l, l') \in \prec \cup \sim$; message (l, E, l') is called instantaneous iff $l \sim l'$ and asynchronous otherwise,
- Cond $\subseteq (2^{\mathcal{C}} \setminus \emptyset) \times \Phi(\mathcal{C})$ is a set of conditions with $(L, \phi) \in$ Cond only if $l \sim l'$ for all $l \neq l' \in L$,
- LocInv $\subseteq \mathcal{L} \times \{\circ, \bullet\} \times \Phi(\mathcal{C}) \times \mathcal{L} \times \{\circ, \bullet\}$ is a set of local invariants with $(l, \iota, \phi, l', \iota') \in$ LocInv only if $l \prec l'$, $\circ$: exclusive, $\bullet$: inclusive,
- $\Theta : \mathcal{L} \cup$ Msg $\cup$ Cond $\cup$ LocInv $\to$ {hot, cold} assigns to each location and each element a temperature.

From Concrete to Abstract Syntax

- locations $\mathcal{L}$,
- $\preceq \subseteq \mathcal{L} \times \mathcal{L}$, $\sim \subseteq \mathcal{L} \times \mathcal{L}$
- $\mathcal{I} = \{I_1, \dots, I_n\}$,
- Msg $\subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L}$,
- Cond $\subseteq (2^{\mathcal{C}} \setminus \emptyset) \times \Phi(\mathcal{C})$
- LocInv $\subseteq \mathcal{L} \times \{\circ, \bullet\} \times \Phi(\mathcal{C}) \times \mathcal{L} \times \{\circ, \bullet\}$,
- $\Theta : \mathcal{L} \cup$ Msg $\cup$ Cond $\cup$ LocInv $\to$ {hot, cold}.

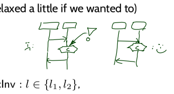


- $\mathcal{L} = \{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, \; l_{1,2}, l_{1,4}, \; l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}, \; l_{3,0}, l_{3,1}, l_{3,2}, l_{3,3}\}$
- $l_{1,0} \prec l_{1,1} \prec l_{1,2} \prec l_{1,3}, \quad l_{1,2} \prec l_{1,4}, \quad l_{2,0} \prec l_{2,1} \prec l_{2,2} \prec l_{2,3}, \quad l_{3,0} \prec l_{3,1} \prec l_{3,2} \prec l_{3,3},$
  $l_{1,1} \prec l_{2,1}, \quad l_{2,2} \prec l_{1,2}, \quad l_{2,3} \prec l_{1,3}, \quad l_{3,2} \prec l_{1,4}, \quad l_{2,1} \sim l_{3,1}, \quad l_{2,2} \sim l_{3,2},$
- $\mathcal{I} = \{\{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, l_{1,4}\}, \{l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}\}, \{l_{3,0}, l_{3,1}, l_{3,2}\}\}$, $l_{3,3}$
- Msg $= \{(l_{1,1}, A, l_{2,1}), (l_{2,2}, B, l_{1,2}), (l_{2,2}, C, l_{3,2}), (l_{2,3}, D, l_{1,3}), (l_{3,3}, E, l_{1,4})\}$
- Cond $= \{(\{l_{2,1}, l_{3,1}\}, c_4), (\{l_{2,2}\}, c_2 \wedge c_3)\}$,
- LocInv $= \{(l_{1,1}, \circ, c_1, l_{1,2}, \bullet)\}$

Well-Formedness

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

- For each location $l \in \mathcal{L}$, **if** l is the location of
  - a **condition**, i.e. $\exists (L, \phi) \in$ Cond $: l \in L$, or
  - a **local invariant**, i.e. $\exists (l_1, \iota_1, \phi, l_2, \iota_2) \in$ LocInv $: l \in \{l_1, l_2\}$,

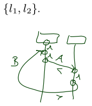


then there is a location l' **simultaneous** to l, i.e. $l \sim l'$, which is the location of

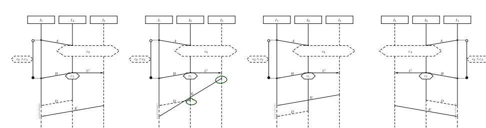
- an **instance head**, i.e. l' is minimal wrt. $\preceq$, or
- a **message**, i.e.

$$\exists (l_1, E, l_2) \in \text{Msg} : l \in \{l_1, l_2\}.$$

Note: if messages in a chart are **cyclic**,
then there doesn't exist a partial order
(so such diagrams **don't even have** an abstract syntax).



Concrete vs. Abstract Syntax

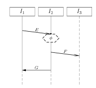


- $\mathcal{L} = \{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, \; l_{1,2}, l_{1,4}, \; l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}, \; l_{3,0}, l_{3,1}, l_{3,2}, l_{3,3}\}$
- $l_{1,0} \prec l_{1,1} \prec l_{1,2} \prec l_{1,3}, \quad l_{1,2} \prec l_{1,4}, \quad l_{2,0} \prec l_{2,1} \prec l_{2,2} \prec l_{2,3}, \quad l_{3,0} \prec l_{3,1} \prec l_{3,2} \prec l_{3,3},$
  $l_{1,1} \prec l_{2,1}, \quad l_{2,2} \prec l_{1,2}, \quad l_{2,3} \prec l_{1,3}, \quad l_{3,2} \prec l_{1,4}, \quad l_{2,1} \sim l_{3,1}, \quad l_{2,2} \sim l_{3,2},$
- $\mathcal{I} = \{\{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, l_{1,4}\}, \{l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}\}, \{l_{3,0}, l_{3,1}, l_{3,2}\}\}$,
- Msg $= \{(l_{1,1}, A, l_{2,1}), (l_{2,2}, B, l_{1,2}), (l_{2,2}, C, l_{3,2}), (l_{2,3}, D, l_{1,3}), (l_{3,3}, E, l_{1,4})\}$
- Cond $= \{(\{l_{2,1}, l_{3,1}\}, c_4), (\{l_{2,2}\}, c_2 \wedge c_3)\}$,
- LocInv $= \{(l_{1,1}, \circ, c_1, l_{1,2}, \bullet)\}$

Content

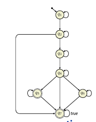
LSC Semantics: Towards Automaton Construction

The Plan: A Formal Semantics for a Visual Formalism



concrete syntax
(diagram)

$((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$
abstract syntax



semantics
(Büchi automaton)

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

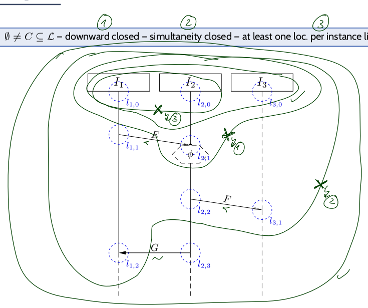
The temperature function is extended to cuts as follows:

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

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

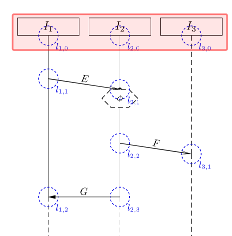
Cut Examples

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



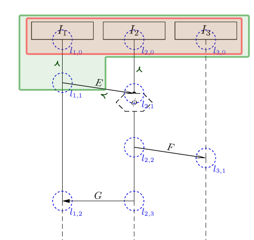
Cut Examples

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



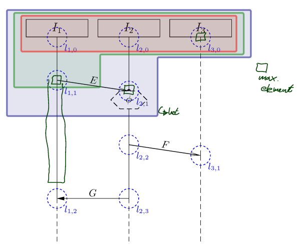
Cut Examples

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



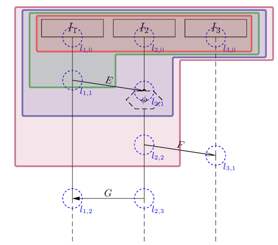
Cut Examples

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



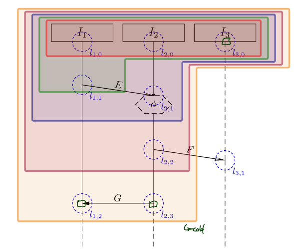
Cut Examples

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



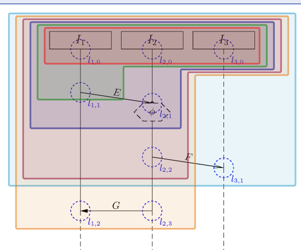
Cut Examples

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



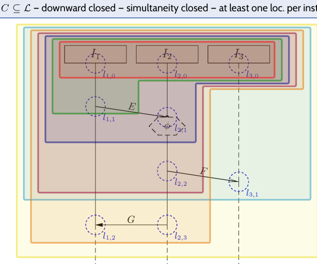
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ – 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 an LSC body as follows:

Definition.
Let $C \subseteq \mathcal{L}$ bet a cut of LSC body $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$.

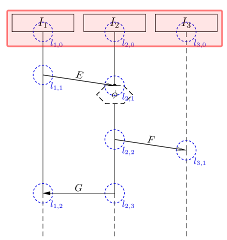
A set $\emptyset \neq \mathcal{F} \subseteq \mathcal{L}$ of locations is called **fired-set** $\mathcal{F}$ of cut C if and only if

- $C \cap \mathcal{F} = \emptyset$ 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.
  $$\forall\, l \in \mathcal{F}\, \exists\, l' \in C \bullet l' \prec l \wedge (\nexists\, l'' \in C \bullet l' \prec l'' \prec l),$$
- 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 $C \rightsquigarrow_{\mathcal{F}} C'$.

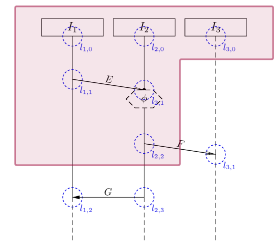
Successor Cut Example

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

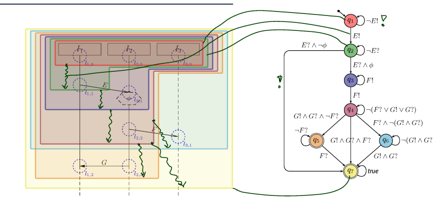


Successor Cut Example

$C \cap \mathcal{F} = \emptyset$ – $C \cup \mathcal{F}$ 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 $\mathcal{C}$ and $\mathcal{E}$ is $(\mathcal{C}_\mathcal{B}, Q, q_{ini}, \rightarrow, Q_F)$ with

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

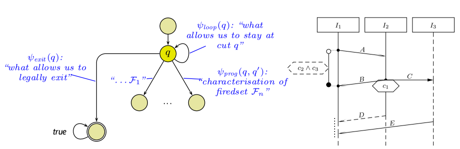
TBA Construction Principle

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

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

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



Tell Them What You've Told Them. . .

- **User Stories**: simple example of scenarios
  - **strong point**: naming tests is necessary,
  - **weak point**: hard to keep overview; global restrictions.
- **Use-Cases**:
  - interactions between system and actors,
  - be sure to elaborate exceptions and corner cases,
  - in particular effective with customers lacking technical background.
- **Use-Case Diagrams**:
  - visualise which participants are relevant for which use-case,
  - are rather **useless** without the underlying use-case.
- **Sequence Diagrams**:
  - a **visual formalism** for interactions, i.e.,
    - precisely defined syntax,
    - precisely defined semantics ($\rightarrow$ next lecture).
  - Can be used to precisely describe the interactions of a **use-case**.

References

References

Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum, 3rd edition.

Beck, K. (1999). *Extreme Programming Explained – Embrace Change*. Addison-Wesley.

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

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, 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.

Jacobson, I. (1992). *Object Oriented Software Engineering – A Use Case Driven Approach*. ACM Press.

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

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

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

Störrle, H. (2003). Assert, negate and refinement in UML-2 interactions. Technical Report TUM-I0323, Technische Universität München.