Softwaretechnik / Software-Engineering

Lecture 9: Live Sequence Charts & RE Wrap-Up

2019-06-03

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Introduction

Definition: Software & SW Specification

Requirements Specification
- Desired Properties
- Kinds of Requirements
- Analysis Techniques

Documents
- Dictionary, Specification

Specification Languages
- Natural Language
- Decision Tables
- Syntax, Semantics
- Completeness, Consistency, ...

Scenarios
- User Stories, Use Cases
- Live Sequence Charts
- Syntax, Semantics

Wrap-Up

Vocabulary
Techniques
- informal
- semi-formal
- formal
The Plan: A Formal Semantics for a Visual Formalism

does the software satisfy the LSC?

read out relevant information

concrete syntax
(diagram)

abstract syntax

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

apply construction procedure

semantics
(Büchi automaton)

software

\(\sqsubseteq\)
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
  - Full LSC (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) Advanced LSC Topics
    - Full LSC with pre-chart
  - LSCs in Requirements Engineering
    - strengthening existential LSCs (scenarios)
      into universal LSCs (requirements)
  - LSCs in Quality Assurance
- Requirements Engineering Wrap-Up
  - Requirements Analysis in a Nutshell
  - Recall: Validation by Translation
LSC Semantics: TBA Construction
**Definition.** Let $((L, \preceq, \sim), I, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta)$ be an LSC body. A non-empty set $\emptyset \neq C \subseteq L$ is called a cut of the LSC body iff $C$

- is **downward closed**, i.e.
  \[ \forall l, l' \in L \bullet l' \in C \land l \preceq l' \implies l \in C, \]

- is **closed under simultaneity**, i.e.
  \[ \forall l, l' \in L \bullet l' \in C \land l \sim l' \implies l \in C, \text{ and} \]

- comprises at least **one location per instance line**, i.e.
  \[ \forall I \in I \bullet C \cap I \neq \emptyset. \]

The temperature function is extended to cuts as follows:

\[
\Theta(C) = \begin{cases} 
\text{hot} & \text{if } \exists l \in C \bullet (\not\exists l' \in C \bullet l < l') \land \Theta(l) = \text{hot} \\
\text{cold} & \text{otherwise}
\end{cases}
\]

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

Cut Examples
Cut Examples

∅ \neq C \subseteq \mathcal{L} — downward closed — simultaneity closed — at least one loc. per instance line
\emptyset \neq C \subseteq \mathcal{L} —\text{downward closed} —\text{simultaneity closed} —\text{at least one loc. per instance line}
A Successor Relation on Cuts

The partial order “≺” and the simultaneity relation “∼” of locations induce a **direct successor relation** on cuts of an LSC body as follows:

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

A set \( \emptyset \neq F \subseteq L \) of locations is called **fired-set** \( F \) of cut \( C \) if and only if

- \( C \cap F = \emptyset \) and \( C \cup F \) is a cut, i.e. \( F \) is closed under simultaneity,

- all locations in \( F \) are **direct \( \prec \)-successors** of the front of \( C \), i.e.

  \[
  \forall l \in F \exists l' \in C \cdot l' \prec l \land (\nexists l'' \in L \cdot l' \prec l'' \prec l),
  \]

- locations in \( F \) that lie on the same instance line are **pairwise unordered**, i.e.

  \[
  \forall l \neq l' \in F \cdot (\exists I \in I \cdot \{l, l'\} \subseteq I) \implies l \not\preceq l' \land l' \not\preceq l,
  \]

- for each asynchronous message reception in \( F \), the corresponding **sending is already in** \( C \),

  \[
  \forall (l, E, l') \in \text{Msg} \cdot l' \in F \implies l \in C.
  \]

The cut \( C' = C \cup F \) is called **direct successor of** \( C \) **via** \( F \), denoted by \( C \sim_{F} C' \).
$C \cap \mathcal{F} = \emptyset \iff C \cup \mathcal{F}$ is a cut — only direct \prec-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in
successor cut example

\[ C \cap \mathcal{F} = \emptyset \]

- only direct \( \prec \)-successors — same instance line on front pairwise unordered

- sending of asynchronous reception already in

\[ C \cup \mathcal{F} \]
Language of LSC Body: Example
The TBA $B(\mathcal{L})$ of LSC $\mathcal{L}$ over $C$ and $\mathcal{E}$ is $(C_B, Q, q_{ini}, \rightarrow, Q_F)$ with

- $C_B = C \cup \mathcal{E}_{I?}^\mathcal{T}$, where $\mathcal{E}_{I?}^\mathcal{T} = \{E_{i,j}^{\xi}, E_{i,j}^{\eta} \mid E \in \mathcal{E}, i, j \in I\}$,

- $Q$ is the set of cuts of $\mathcal{L}$, $q_{ini}$ is the instance heads cut,

- $\rightarrow$ consists of loops, progress transitions (from $\sim \mathcal{F}$), 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.
Recall: The TBA $B(L)$ of LSC $L$ is $(C, Q, q_{ini}, \rightarrow, Q_F)$ with

- $Q$ is the set of cuts of $L$, $q_{ini}$ is the instance heads cut,
- $C_B = C \cup \mathcal{E}_I$,
- $\rightarrow$ consists of loops, progress transitions (from $\sim \mathcal{F}$), and legal exits (cold cond./local inv.),
- $Q_F = \{ C \in Q \mid v(C) = \text{cold} \lor C = L \}$ is the set of cold cuts.

So in the following, we “only” need to construct the transitions’ labels:

$$\rightarrow = \{(q, \psi_{loop}(q), q) \mid q \in Q \} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \sim \mathcal{F} q' \} \cup \{(q, \psi_{exit}(q), L) \mid q \in Q \}$$
“Only” construct the transitions’ labels:

\[ \rightarrow = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \xrightarrow{e} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\} \]

\[ \psi_{loop}(q) = \psi_{Msg}(q) \wedge \psi_{LocInv}^{hot}(q) \wedge \psi_{LocInv}^{cold}(q) \]

\[ \psi_{exit}(q) = (\psi_{loop}^{hot}(q) \wedge \neg \psi_{LocInv}^{cold}(q)) \wedge \bigvee_{1 \leq i \leq n}(\psi_{prog}^{hot}(q, q_i) \wedge \neg \psi_{LocInv}^{cold}(q, q_i)) \]

\[ \psi_{prog}(q, q_n) = \psi_{Msg}(q, q_n) \wedge \psi_{Cond}^{hot}(q, q_n) \wedge \psi_{LocInv,\bullet}^{hot}(q, q_n) \wedge \psi_{Cond}^{cold}(q, q_n) \wedge \psi_{LocInv,\bullet}^{cold}(q, q_n) \]

\[ \psi_{hot}^{loop}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \neg \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \psi_{LocInv}^{hot}(q) \wedge \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \neg \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \psi_{LocInv}^{hot}(q) \wedge \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \neg \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \wedge \psi_{LocInv}^{hot}(q) \wedge \psi_{LocInv}^{cold}(q) \]

\[ \psi_{hot}^{loop}(q) \]
**Loop Condition**

\[ \psi_{\text{loop}}(q) = \psi_{\text{Msg}}(q) \land \psi_{\text{LocInv hot}}(q) \land \psi_{\text{LocInv cold}}(q) \]

- \( \psi_{\text{Msg}}(q) = \neg \bigvee_{1 \leq i \leq n} \psi \in \text{Msg}(q_i \setminus q) \psi \land \left( \text{strict } \implies \bigwedge_{\psi \in E_1} \neg \psi \right) \)

- \( \psi_{\text{LocInv}}(q) = \bigwedge_{(l, \ell, \phi, l', \ell') \in \text{LocInv}, \Theta(\ell) = \theta} \ell \text{ active at } q \phi \)

A location \( l \) is called **front location** of cut \( C \) if and only if \( \nexists l' \in C \cdot l < l' \).

Local invariant \( (l_0, \nu_0, \phi, l_1, \nu_1) \) is **active** at cut (!) \( q \) if and only if \( l_0 \leq l < l_1 \) for some front location \( l \) of cut \( q \) or \( l = l_1 \land \nu_1 = \bullet \).

- \( \text{Msg}(\mathcal{F}) = \{ E_1^{I(l), I(l')} | (l, E, l') \in \text{Msg}, l \in \mathcal{F} \} \cup \{ E_2^{I(l), I(l')} | (l, E, l') \in \text{Msg}, l' \in \mathcal{F} \} \)

- \( \text{Msg}(\mathcal{F}_1, \ldots, \mathcal{F}_n) = \bigcup_{1 \leq i \leq n} \text{Msg}(\mathcal{F}_i) \)
**Progress Condition**

\[ \psi_{\text{prog}}^{\text{hot}}(q, q_i) = \psi_{\text{Msg}}^{\text{hot}}(q, q_n) \land \psi_{\text{hot}}^{\text{Cond}}(q, q_n) \land \psi_{\text{hot}}^{\text{LocInv}, \bullet}(q_n) \]

- \( \psi_{\text{Msg}}^{\text{hot}}(q, q_i) = \bigwedge_{\psi \in \text{Msg}(q_i \setminus q)} \psi \land \bigwedge_{j \neq i} \bigwedge_{\psi \in (\text{Msg}(q_j \setminus q) \setminus \text{Msg}(q_i \setminus q))} \neg \psi \land (\text{strict} \implies \bigwedge_{\psi \in (E_i \cap \text{Msg}(L)) \setminus \text{Msg}(F_i)} \neg \psi) =: \psi_{\text{strict}}^{\text{hot}}(q, q_i) \)

- \( \psi_{\text{Cond}}^{\text{hot}}(q, q_i) = \bigwedge_{\gamma = (L, \phi) \in \text{Cond}, \Theta(\gamma) = \theta, L \cap (q_i \setminus q) \neq \emptyset} \phi \)

- \( \psi_{\text{LocInv}, \bullet}^{\text{hot}}(q, q_i) = \bigwedge_{\lambda = (l, \iota, \phi, l', \iota') \in \text{LocInv}, \Theta(\lambda) = \theta, \lambda \bullet\text{-active at } q_i} \phi \)

Local invariant \((l_0, \iota_0, \phi, l_1, \iota_1)\) is \(\bullet\text{-active}\) at \(q\) if and only if

- \(l_0 < l < l_1, \text{ or}\)
- \(l = l_0 \land \iota_0 = \bullet, \text{ or}\)
- \(l = l_1 \land \iota_1 = \bullet\)

for some front location \(l\) of cut (!) \(q\).
Example (without strictness condition)
Example (without strictness condition)
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
  - Full LSC (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) Advanced LSC Topics
    - Full LSC with pre-chart
- LSCs in Requirements Engineering
  - strengthening existential LSCs (scenarios) into universal LSCs (requirements)
- LSCs in Quality Assurance

- Requirements Engineering Wrap-Up
  - Requirements Analysis in a Nutshell
  - Recall: Validation by Translation
Excursion: Symbolic Büchi Automata
From Finite Automata to Symbolic Büchi Automata

\[ A: \quad \Sigma = \{0, 1\} \]

\[ B: \quad \Sigma = \{0, 1\} \]

\[ B': \quad \Sigma = \{0, 1\} \]

\[ A_{sym}: \quad \Sigma = \{a, b, c, d\} \rightarrow \mathbb{B} \]

\[ B_{sym}: \quad \Sigma = \{a, b, c, d\} \rightarrow \mathbb{B} \]

\[ \omega = 0101010101\ldots \]

\[ L(B) = 0(10)^* \]

\[ L(B') = ? \]

\[ L(A_{sym}) = \left\{ a, b, c, d \right\} \rightarrow \mathbb{B} \]

\[ (a \land b, b \land c, c \land d, d \land a) = \left\{ \frac{a}{b}, \frac{c}{d} \right\} \cup \left\{ \frac{a}{b}, \frac{a}{d}, \frac{b}{d} \right\} \cup \left\{ \frac{a}{b}, \frac{b}{d}, \frac{c}{d} \right\} \cup \left\{ \frac{a}{b}, \frac{b}{d}, \frac{b}{c} \right\} \cup \left\{ \frac{a}{b}, \frac{b}{d}, \frac{c}{d}, \frac{b}{c} \right\} \]
Symbolic Büchi Automata

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

\[ B = (C_B, Q, q_{ini}, \to, Q_F) \]

where

- \( C_B \) is a set of atomic propositions,
- \( Q \) is a finite set of states,
- \( q_{ini} \in Q \) is the initial state,
- \( \to \subseteq Q \times \Phi(C_B) \times Q \) is the finite transition relation.
  Each transitions \((q, \psi, q') \in \to\) from state \(q\) to state \(q'\)
  is labelled with a propositional formula \(\psi \in \Phi(C_B)\).
- \( Q_F \subseteq Q \) is the set of fair (or accepting) states.

Example:

\[ B_{sym}: \Sigma = \{a, b, c, d\} \rightarrow B \]
**Definition.** Let $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ be a TBA and

$$w = \sigma_1, \sigma_2, \sigma_3, \cdots \in (C_B \rightarrow \mathbb{B})^\omega$$

an infinite word, each letter is a valuation of $C_B$.

An infinite sequence

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

of states is called **run** of $B$ over $w$ if and only if

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

**Example:**

$$w = \{a \mapsto true, b \mapsto true, c \mapsto false, d \mapsto false\}, \{c\}, \{a, b\}, \{(d), \{a, b\}\}^\omega$$

{a, b} for short
The Language of a TBA

Definition.
We say TBA $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ accepts the word

$$w = (\sigma_i)_{i \in \mathbb{N}_0} \in (C_B \rightarrow B)^\omega$$

if and only if $B$ has a run

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

over $w$ such that fair (or accepting) states are visited infinitely often by $\rho$, i.e.,

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

We call the set $\text{Lang}(B) \subseteq (C_B \rightarrow B)^\omega$ of words that are accepted by $B$ the language of $B$.

Example:

$B_{sym} : \quad \Sigma = \{a, b, c, d\} \rightarrow B$

$$B_{sym} : \quad \Sigma = \{a, b, c, d\} \rightarrow B$$

1. $q_1 \xleftarrow{a \land b} q_2$
2. $q_1 \xrightarrow{c \lor d} q_2$
LSCs vs. Software
**Software, formally**

**Definition.** Software is a finite description $S$ of a (possibly infinite) set $[[S]]$ of (finite or infinite) computation paths of the form

$$
s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots
$$

where

- $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called state (or configuration), and
- $\alpha_i \in A$, $i \in \mathbb{N}_0$, is called action (or event).

The (possibly partial) function $[[\cdot]] : S \mapsto [[S]]$ is called interpretation of $S$. 
**Definition.** A **software specification** is a finite description $\mathcal{S}$ of a (possibly infinite) set $[\mathcal{S}]$ of softwares, i.e.

$$[\mathcal{S}] = \{(S_1, [\cdot]_1), (S_2, [\cdot]_2), \ldots\}.$$  

The (possibly partial) function $[\cdot] : \mathcal{S} \mapsto [\mathcal{S}]$ is called **interpretation** of $\mathcal{S}$.

**Definition.** Software $(S, [\cdot])$ **satisfies** software specification $\mathcal{S}$, denoted by $S \models \mathcal{S}$, if and only if

$$(S, [\cdot]) \in [\mathcal{S}].$$
Software Satisfies Software Specification: Example

Software Specification

\( \mathcal{S} : \)

<table>
<thead>
<tr>
<th>( T ): room ventilation</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b ) button pressed?</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( off ) ventilation off?</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \ast )</td>
</tr>
<tr>
<td>( on ) ventilation on?</td>
<td>( \ast )</td>
<td>( \times )</td>
<td>( \ast )</td>
</tr>
<tr>
<td>( go ) start ventilation</td>
<td>( \ast )</td>
<td>( \times )</td>
<td>( \ast )</td>
</tr>
<tr>
<td>( stop ) stop ventilation</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \ast )</td>
</tr>
</tbody>
</table>

Define: \((S, [\cdot]) \in [\mathcal{S}]\) if and only if for all

\[
\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in [S]
\]

and for all \(i \in \mathbb{N}_0\),

\[
\exists r \in T \cdot \sigma_i \models \mathcal{F}(r).
\]

Software

- Assume we have a program \( S \) for the room ventilation controller.
- Assume we can observe at well-defined points in time the conditions \( b, off, on, go, stop \) when the software runs.
- Then the behaviour \([S]\) of \( S \) can be viewed as computation paths of the form

\[
\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{\tau} \sigma_2 \cdots
\]

where each \( \sigma_i \) is a valuation of \( b, off, on, go, stop \), i.e. \( \sigma_i : \{b, off, on, go, stop\} \rightarrow \mathbb{B} \).

- For example:

\[
(\ off) \xrightarrow{\tau} \left( \begin{array}{c} b \\ off \\ go \end{array} \right) \xrightarrow{\tau} (\ on) \xrightarrow{\tau} \left( \begin{array}{c} b \\ on \\ stop \end{array} \right) \xrightarrow{\tau} (\ off) \ldots
\]
Software Satisfies Software Specification: Another Example

Software Specification

\( \mathcal{J} : \)

- LSC: buy water
- AC: true
- AM: invariant: \( l \) strict

User \( \rightarrow \) CoinValidator \( \rightarrow \) ChoicePanel \( \rightarrow \) Dispenser

Define: \( (S, [\cdot]) \in [\mathcal{J}] \) if and only if

- tja... (in a minute)

Software

- Assume we can observe at well-defined points in time the observables relevant for the LSC (conditions and messages) when the software \( S \) runs.

- Then the behaviour \([S]\) of \( S \) can be viewed as computation paths over the LSC’s observables.

- For example:

\[ \sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E_{U,V}} \sigma_2 \xrightarrow{pSOFT_{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT_{V,U}} \ldots \]

- And then we can relate \( S \) to \( \mathcal{J} \).
The Plan: A Formal Semantics for a Visual Formalism

- concrete syntax (diagram)
- abstract syntax
- does the software satisfy the LSC?
- read out relevant information
- apply construction procedure

((L, ≤, ∼), I, Msg, Cond, LocInv, Θ)

semantics (Büchi automaton)

software
A software $S$ is called \textbf{compatible} with LSC $\mathcal{L}$ over $C$ and $E$ is if and only if

- $\Sigma = (C \rightarrow \mathbb{B}), C \subseteq C$, i.e. the \textbf{states} comprise valuations of the conditions in $C$,
- $A = (B \rightarrow \mathbb{B}), E^{T} \subseteq B$, i.e. the \textbf{events} comprise valuations of $E^{i,j}$, $E^{i,j}$.

A computation path $\pi = \sigma_{0} \xrightarrow{\alpha_{1}} \sigma_{1} \xrightarrow{\alpha_{2}} \sigma_{2} \cdots \in [S]$ of software $S$ \textbf{induces} the word $w(\pi) = (\sigma_{0} \cup \alpha_{1}), (\sigma_{1} \cup \alpha_{2}), (\sigma_{2} \cup \alpha_{3}), \ldots$, we use $W_{S}$ to denote the set of words induced by $[S]$, i.e.

$$W_{S} = \{w(\pi) \mid \pi \in [S]\}.$$
LSCs vs. Software (or Systems)

\[ \begin{align*}
\Gamma = & \sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E_1 U,V} \sigma_2 \xrightarrow{pSOFT U,V} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{SOFT V,U} \sigma_6 \xrightarrow{\tau} \ldots \in [S] \\
\omega(\pi) = & \{ \}, \{ E_1 U,V, E_1 U,V \}, \{ pSOFT U,V, pSOFT U,V \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \ldots \\
& \in \mathcal{L}(A_C) \\
\end{align*} \]

\[ w = \{ \}, \{ E_1 U,V, E_1 U,V \}, \{ pSOFT U,V, pSOFT U,V \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \{ \}, \ldots \in \text{Lang}(B(\mathcal{L})) \]

\[ \begin{array}{c|c}
\text{User} & \text{Vend. Mach.} \\
\hline
E_1 & \text{pSOFT} \\
\hline
\text{pSOFT} & \text{SOFT} \\
\end{array} \]

\[ \begin{array}{c}
E_1: \text{ insert 1€ coin} \\
pSOFT: \text{ press ‘SOFT’ button} \\
SOFT: \text{ dispense soft drink} \\
\end{array} \]

TBA over \( C_B = C \cup \mathcal{E}^\pi \), where \( C = \emptyset \) and \( \mathcal{E}^\pi = \{ E_1 U,V, E_1 U,V, pSOFT U,V, pSOFT U,V, \ldots \} \).
LSCs vs. Software (or Systems)

\[ \sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E_1^{U,V}} \sigma_2 \xrightarrow{pSOFT^{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT^{V,U}} \ldots \in [S] \]

\[ w(\pi) = \sigma_0, (\sigma_1 \cup \{ E_1^{U,V}, E_1^{U,V} \}), (\sigma_2 \cup \{ pSOFT_1^{U,V}, pSOFT_1^{U,V} \}), \sigma_3, \sigma_4, \sigma_5, (\sigma_6 \cup \{ SOFT_1^{V,U}, SOFT_1^{V,U} \}), \ldots \]

\[ w = \{ \}, \{ E_1^{U,V}, E_1^{U,V} \}, \{ pSOFT_1^{U,V}, pSOFT_1^{U,V} \}, \{ \}, \{ \}, \{ \}, \{ SOFT_1^{V,U}, SOFT_1^{V,U} \}, \{ \}, \ldots \in \text{Lang}(B(\mathcal{L})) \]

**User**

- **E1:** insert 1 € coin
- **pSOFT:** press ‘SOFT’ button
- **SOFT:** dispense soft drink

**Vend. Mach.**

- **E1:**
- **pSOFT:**
- **SOFT:**

TBA over \( C_B = C \cup \mathcal{E}_{?!} \), where \( C = \emptyset \) and

\( \mathcal{E}_{?!} = \{ E_1^{U,V}, E_1^{?}, pSOFT_1^{U,V}, pSOFT_1^{?}, SOFT_1^{V,U}, SOFT_1^{?}, \ldots \} \).
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
  - Full LSC (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) Advanced LSC Topics
    - Full LSC with pre-chart
- LSCs in Requirements Engineering
  - strengthening existential LSCs (scenarios) into universal LSCs (requirements)
- LSCs in Quality Assurance
- Requirements Engineering Wrap-Up
  - Requirements Analysis in a Nutshell
  - Recall: Validation by Translation
Activation Condition and Mode
A full LSC $\mathcal{L} = (MC, ac_0, am, \Theta_{\mathcal{L}})$ consists of

- (non-empty) **main-chart** $MC = ((L_M, \preceq_M, \sim_M), I_M, \text{Msg}_M, \text{Cond}_M, \text{LocInv}_M, \Theta_M)$,
- **activation condition** $ac_0 \in \Phi(C)$,
- **strictness flag** $\text{strict}$ (if false, $\mathcal{L}$ is **permissive**)
- **activation mode** $am \in \{\text{initial, invariant}\}$,
- **chart mode** **existential** ($\Theta_{\mathcal{L}} = \text{cold}$) or **universal** ($\Theta_{\mathcal{L}} = \text{hot}$).
Software Satisfies LSC

Let $S$ be a software which is compatible with LSC $\mathcal{L}$ (without pre-chart).

We say software $S$ satisfies LSC $\mathcal{L}$, denoted by $S \models \mathcal{L}$, if and only if

\[
\begin{array}{c|c|c}
\Theta_{\mathcal{L}} & am = initial & am = invariant \\
\hline
cold & \exists w \in W_S \bullet w^0 \models ac \land \lnot \psi_{exit}(C_0) \\
& \land w^0 \models \psi_{prog}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \exists w \in W_S \exists k \in \mathbb{N}_0 \bullet w^k \models ac \land \lnot \psi_{exit}(C_0) \\
& \land w^k \models \psi_{prog}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L})) \\
\hline
hot & \forall w \in W_S \bullet w^0 \models ac \land \lnot \psi_{exit}(C_0) \\
& \implies w^0 \models \psi_{prog}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \forall w \in W_S \forall k \in \mathbb{N}_0 \bullet w^k \models ac \land \lnot \psi_{exit}(C_0) \\
& \implies w^k \models \psi_{cond_{\text{hot}}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L})) \\
\end{array}
\]

where and $C_0$ is the minimal (or instance heads) cut of the main-chart.

\[
\begin{array}{c|c|c}
\text{LSC: } & \text{AC: } & \text{AM: } \\
\text{true} & \text{invariant} & \text{I: permissive} \\
\hline
\text{User} & \text{Vend. Mach.} & \\
\hline
E1 & pSOFT & \\
\hline
\text{SOFT} & \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{LSC: } & \text{AC: } & \text{AM: } \\
\text{false} & \text{initial} & \text{I: permissive} \\
\hline
\text{User} & \text{Vend. Ma.} & \\
\hline
t & ckwATER & \\
\hline
ckSOFT & \\
\hline
ckTEA & \\
\end{array}
\]
Let $S$ be a software which is **compatible** with LSC $\mathcal{L}$ (without pre-chart).

We say software $S$ **satisfies** LSC $\mathcal{L}$, denoted by $S \models \mathcal{L}$, if and only if

<table>
<thead>
<tr>
<th>$\Theta$</th>
<th>$am = \text{initial}$</th>
<th>$am = \text{invariant}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cold</strong></td>
<td>$\exists w \in W_S \bullet w^0 \models ac \land \neg \psi_{exit}(C_0)$ $\land w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\exists w \in W_S \exists k \in \mathbb{N}<em>0 \bullet w^k \models ac \land \neg \psi</em>{exit}(C_0)$ $\land w^k \models \psi_{\text{prog}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))$</td>
</tr>
<tr>
<td><strong>hot</strong></td>
<td>$\forall w \in W_S \bullet w^0 \models ac \land \neg \psi_{exit}(C_0)$ $\implies w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\forall w \in W_S \forall k \in \mathbb{N}<em>0 \bullet w^k \models ac \land \neg \psi</em>{exit}(C_0)$ $\implies w^k \models \psi_{\text{hot}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))$</td>
</tr>
</tbody>
</table>

where and $C_0$ is the minimal (or **instance heads**) cut of the main-chart.

Software $S$ satisfies a **set of** LSCs $\mathcal{L}_1, \ldots, \mathcal{L}_n$ if and only if $S \models \mathcal{L}_i$ for all $1 \leq i \leq n$. 
LSCs At Work
Example: Vending Machine

- **Positive scenario: Buy a Softdrink**
  We (only) accept the software if it is possible to buy a softdrink.
  (i) Insert one 1 euro coin.
  (ii) Press the 'softdrink' button.
  (iii) Get a softdrink.

- **Positive scenario: Get Change**
  We (only) accept the software if it is possible to 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.

- **Requirement: Perform Self-Test on Power-on**
  We (only) accept the software if it always performs a self-test on power-on.
  (i) Check water dispenser.
  (ii) Check softdrink dispenser.
  (iii) Check tea dispenser.
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
  - Full LSC (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) Advanced LSC Topics
    - Full LSC with pre-chart
- LSCs in Requirements Engineering
  - strengthening existential LSCs (scenarios)
    into universal LSCs (requirements)
- LSCs in Quality Assurance
- Requirements Engineering Wrap-Up
  - Requirements Analysis in a Nutshell
  - Recall: Validation by Translation
(Slightly) Advanced LSC Topics
Full LSC Syntax (with pre-chart)

A full LSC $\mathcal{L} = (PC, MC, ac_0, am, \Theta_\mathcal{L})$ consists of

- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \text{Msg}_P, \text{Cond}_P, \text{LocInv}_P, \Theta_P)$ (possibly empty),

- (non-empty) **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, \text{Msg}_M, \text{Cond}_M, \text{LocInv}_M, \Theta_M),$

- activation condition $ac_0 \in \Phi(C),$

- strictness flag $\text{strict}$ (if *false*, $\mathcal{L}$ is permissive)

- activation mode $am \in \{\text{initial, invariant}\},$

- chart mode existential ($\Theta_\mathcal{L} = \text{cold}$) or universal ($\Theta_\mathcal{L} = \text{hot}$).


**LSC Semantics with Pre-chart**

<table>
<thead>
<tr>
<th>$\Theta_L$</th>
<th>$am = \text{initial}$</th>
<th>$am = \text{invariant}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta_L = \text{cold}$</td>
<td>$\exists w \in W \exists m \in \mathbb{N}_0 \bullet$</td>
<td>$\exists w \in W \exists k &lt; m \in \mathbb{N}_0 \bullet$</td>
</tr>
<tr>
<td></td>
<td>$\land w^0 \models ac \land \lnot \psi_{exit}(C^P_0) \land \psi_{prog}(\emptyset, C^P_0)$</td>
<td>$\land w^k \models ac \land \lnot \psi_{exit}(C^P_0) \land \psi_{prog}(\emptyset, C^P_0)$</td>
</tr>
<tr>
<td></td>
<td>$\land w/1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
<td>$\land w/k + 1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
</tr>
<tr>
<td></td>
<td>$\land w^{m+1} \models \lnot \psi_{exit}(C^M_0)$</td>
<td>$\land w^{m+1} \models \lnot \psi_{exit}(C^M_0)$</td>
</tr>
<tr>
<td></td>
<td>$\land w^{m+1} \models \psi_{prog}(\emptyset, C^M_0)$</td>
<td>$\land w^{m+1} \models \psi_{prog}(\emptyset, C^M_0)$</td>
</tr>
<tr>
<td></td>
<td>$\land w/m + 2 \in \text{Lang}(B(MC))$</td>
<td>$\land w/m + 2 \in \text{Lang}(B(MC))$</td>
</tr>
</tbody>
</table>

| $\Theta_L = \text{hot}$ | $\forall w \in W \forall m \in \mathbb{N}_0 \bullet$ | $\forall w \in W \forall k \leq m \in \mathbb{N}_0 \bullet$ |
|  | $\land w^0 \models ac \land \lnot \psi_{exit}(C^P_0) \land \psi_{prog}(\emptyset, C^P_0)$ | $\land w^k \models ac \land \lnot \psi_{exit}(C^P_0) \land \psi_{prog}(\emptyset, C^P_0)$ |
|  | $\land w/1, \ldots, w/m \in \text{Lang}(B(PC))$ | $\land w/k + 1, \ldots, w/m \in \text{Lang}(B(PC))$ |
|  | $\land w^{m+1} \models \lnot \psi_{exit}(C^M_0)$ | $\land w^{m+1} \models \lnot \psi_{exit}(C^M_0)$ |
|  | $\land w^{m+1} \models \psi_{prog}(\emptyset, C^M_0)$ | $\land w^{m+1} \models \psi_{prog}(\emptyset, C^M_0)$ |
|  | $\land w/m + 2 \in \text{Lang}(B(MC))$ | $\land w/m + 2 \in \text{Lang}(B(MC))$ |

where $C^P_0$ and $C^M_0$ are the minimal (or instance heads) cuts of pre- and main-chart.
Pre-Charts At Work
Example: Vending Machine

- **Requirement**: Buy Water
  
  We (only) accept the software if,
  
  (i) *Whenever* we insert 0.50 €,
  (ii) and press the ‘water’ button (and no other button),
  (iii) and there is water in stock,
  (iv) *then* we get water (and nothing else).

- **Negative scenario**: A Drink for Free
  
  We don’t accept the software if it is possible to get 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.
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
  - Full LSC (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) Advanced LSC Topics
    - Full LSC with pre-chart

- LSCs in Requirements Engineering
  - strengthening existential LSCs (scenarios) into universal LSCs (requirements)
  - LSCs in Quality Assurance

- Requirements Engineering Wrap-Up
  - Requirements Analysis in a Nutshell
  - Recall: Validation by Translation
LSCs in Requirements Analysis
One quite effective approach:

(i) **Approximate** the software requirements: ask for positive / negative **existential scenarios**.

- **Ask** the customer to describe **example usages** of the desired system.  
  In the sense of: “If the system is not at all able to do this, then it’s not what I want.”  
  (→ positive use-cases, existential LSC)

- **Ask** the customer to describe behaviour that **must not happen** in the desired system.  
  In the sense of: “If the system does this, then it’s not what I want.”  
  (→ negative use-cases, LSC with pre-chart and hot-false)

(ii) **Refine** result into **universal scenarios** (and validate them with customer).

- **Investigate** **preconditions, side-conditions, exceptional cases** and **corner-cases**.  
  (→ extend use-cases, refine LSCs with conditions or local invariants)

- **Generalise** into universal requirements, e.g., **universal LSCs**.

- **Validate** with customer using new positive / negative scenarios.
Strengthening Scenarios Into Requirements

Customer Developer
announcement (Lastenheft) → Customer Developer offer (Pflichtenheft) → Customer Developer software contract (incl. Pflichtenheft) → Developer software delivery

Customer Developer software delivery

Needs!

Solution!

needs 1
needs 2
needs 3...

Solution!

Customer Developer
announcement (Lastenheft)

Customer Developer offer (Pflichtenheft)

Customer Developer software contract (incl. Pflichtenheft)

Developer software delivery

prop. 1
prop. 2...

Customer Developer
software contract (incl. Pflichtenheft)

Customer Developer software delivery

spec 1
spec 2
aspec 2b...

Customer Developer
software contract (incl. Pflichtenheft)
Strengthening Scenarios Into Requirements

- **Ask customer for (pos./neg.) scenarios**, note down as existential LSCs:

- **Strengthen into requirements**, note down as universal LSCs:

- **Re-Discuss** with customer using example words of the LSCs’ language.
LSCs vs. Quality Assurance
Software $S$ satisfies existential LSC $\mathcal{L}$ if there exists $\pi \in \llbracket S \rrbracket$ such that $\mathcal{L}$ accepts $w(\pi)$. Prove $S \models \mathcal{L}$ by demonstrating $\pi$.

Note: Existential LSCs* may hint at test-cases for the acceptance test! (*: as well as (positive) scenarios in general, like use-cases)
Software \( S \) satisfies existential LSC \( \mathcal{L} \) if there exists \( \pi \in \llbracket S \rrbracket \) such that \( \mathcal{L} \) accepts \( w(\pi) \). Prove \( S \models \mathcal{L} \) by demonstrating \( \pi \).

Note: Existential LSCs* may hint at test-cases for the acceptance test! (*: as well as (positive) scenarios in general, like use-cases)
How to Prove that a Software Satisfies an LSC?

- Software $S$ satisfies **existential** LSC $\mathcal{L}$ if there exists $\pi \in \llbracket S \rrbracket$ such that $\mathcal{L}$ accepts $w(\pi)$. Prove $S \models \mathcal{L}$ by demonstrating $\pi$.

- Note: **Existential** LSCs* may hint at test-cases for the acceptance test! ($\ast$: as well as (positive) scenarios in general, like use-cases)

- **Universal** LSCs (and negative/anti-scenarios!) in general need an exhaustive analysis! (Because they require that the software never ever exhibits the unwanted behaviour.)

Prove $S \not\models \mathcal{L}$ by demonstrating one $\pi$ such that $w(\pi)$ is not accepted by $\mathcal{L}$.
Pushing Things Even Further

(Harel and Marelly, 2003)
Tell Them What You’ve Told Them...

- **Live Sequence Charts** (if well-formed)
  - have an abstract syntax: instance lines, messages, conditions, local invariants; mode: hot or cold.

- From an abstract syntax, mechanically construct its **TBA**.

- An **LSC** is **satisfied** by a software $S$ if and only if
  - **existential** (cold):
    - there is a word induced by a computation path of $S$
    - which is **accepted** by the LSC’s pre/main-chart TBA.
  - **universal** (hot):
    - all words induced by the computation paths of $S$
    - are **accepted** by the LSC’s pre/main-chart TBA.

- **Pre-charts** allow us to
  - specify **anti-scenarios** (“this must not happen”),
  - contrain **activation**.

- **Method**:
  - discuss (anti-)scenarios with customer,
  - generalise into universal LSCs and re-validate.
Requirements Engineering Wrap-Up
Introduction

Definition: **Software & SW Specification**

**Requirements Specification**
- Desired Properties
- Kinds of Requirements
- Analysis Techniques

**Documents**
- Dictionary, Specification

**Specification Languages**
- Natural Language
- Decision Tables
- Syntax, Semantics
- Completeness, Consistency, ...

**Scenarios**
- User Stories, Use Cases
- Live Sequence Charts
- Syntax, Semantics

**Wrap-Up**
Risks Implied by Bad Requirements Specifications

**design and implementation,**
- without specification, programmers may just “ask around” when in doubt, possibly yielding different interpretations → **difficult integration**
- without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → **systematic testing hardly possible**

**negotiation** (with customer, marketing department, or …)

**documentation,** e.g., the **user’s manual,**
- without specification, the user’s manual author can only describe what the system does, not what it should do (“every observation is a feature”)
- later **re-implementations.**
  - the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → **additional effort**

**preparation of tests,**
- without specification, re-use needs to be based on re-reading the code → **risk of unexpected changes**

**acceptance** by customer, resolving later objections or regress claims,
- without specification, it is unclear at delivery time whether behaviour is an error (developer needs to fix) or correct (customer needs to accept and pay) → **nasty disputes, additional effort**
Customers may not know what they want.

- That's in general not their “fault”!
- Care for tacit requirements.
- Care for non-functional requirements / constraints.

For requirements elicitation, consider starting with

- scenarios (“positive use case”) and anti-scenarios (“negative use case”)

and elaborate corner cases.

Thus, use cases can be very useful — use case diagrams not so much.

- Maintain a dictionary and high-quality descriptions.
- Care for objectiveness / testability early on.
  Ask for each requirements: what is the acceptance test?

Use formal notations
- to fully understand requirements (precision),
- for requirements analysis (completeness, etc.),
- to communicate with your developers.

If in doubt, complement (formal) diagrams with text (as safety precaution, e.g., in lawsuits).
Two broad directions:

- **Option 1**: teach formalism (usually not economic).
- **Option 2**: serve as translator / mediator.

1. Domain experts **tell** system scenario $S$ (maybe keep back, whether allowed / forbidden),
2. FM expert **translates** system scenario to valuation $\sigma$,
3. FM expert **evaluates** DT on $\sigma$,
4. FM expert **translates** outcome to “allowed / forbidden by DT”;
5. Compare expected outcome and real outcome.

**Recommendation:** (Course's Manifesto?)

- use formal methods for the **most important/intricate** requirements (formalising all requirements is in most cases **not possible**).
- use the **most appropriate** formalism for a given task,
- use formalisms that you know (really) well.
(Strong) Literature Recommendation

(Rupp and die SOPHISTen, 2014)
References
