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

Topic Area Requirements Engineering: Content

- 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
The Plan: A Formal Semantics for a Visual Formalism

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.
  \[\forall l, l' \in L \cdot l' \in C \land l \preceq l' \implies l \in C,\]
- is closed under simultaneity, i.e.
  \[\forall l, l' \in L \cdot l' \in C \land l \sim l' \implies l \in C,\]
  and
- comprises at least one location per instance line, i.e.
  \[\forall l \in I \cdot \exists C \subseteq L \cdot I \ni l \neq \emptyset.\]

The temperature function is extended to cuts as follows:

\[\Theta(C) = \begin{cases} \text{hot} & \text{if } \exists l \in C \cdot (\exists l' \in C \cdot l \sim l') \land \Theta(l) = \text{hot} \\ \text{cold} & \text{otherwise} \end{cases}\]

that is, \(C\) is hot if and only if at least one of its maximal elements is hot.
Cut Examples

\[ \emptyset \neq C \subseteq L \] — downward closed — simultaneity closed — at least one loc. per instance line

Cut Examples

\[ \emptyset \neq C \subseteq L \] — downward closed — simultaneity closed — at least one loc. per instance line
Cut Examples

∅ \neq C \subseteq \mathcal{L} \text{ — downward closed — simultaneity closed — at least one loc. per instance line}
Cut Examples

\[ \emptyset \neq \mathcal{C} \subseteq \mathcal{L} \text{ — downward closed — simultaneity closed — at least one loc. per instance line} \]
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 \(C \subseteq L\) 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.
  \[\forall l \in F \exists l' \in C \land l' \prec l \land (\exists l'' \in L \land l'' < l'),\]
- locations in \(F\) that lie on the same instance line are pairwise unordered, i.e.
  \[\forall l \neq l' \in F \land (\exists l \in I \land \{l, l'\} \subseteq I) \implies l \not\preceq l' \land l' \not\preceq l,\]
- for each asynchronous message reception in \(F\), the corresponding sending is already in \(C\),
  \[\forall (l, E, l') \in \text{Msg} \land l' \in F \implies l \in C.\]

The cut \(C' = C \cup F\) is called **direct successor of** \(C\) via \(F\), denoted by \(C \xrightarrow{\phi} C'\).
Successor Cut Example

\[ C \cap F = \emptyset - C \cup F \] is a cut — only direct \( \prec \)-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in
The TBA $B(\mathcal{L})$ of LSC $\mathcal{L}$ over $\mathcal{C}$ and $\mathcal{E}$ is $(\mathcal{C}_B, Q, q_{ini}, \rightarrow, Q_F)$ with

1. $\mathcal{C}_B = \mathcal{C} \cup \mathcal{E}^T$, where $\mathcal{E}^T = \{ E_i^{i,j} : E_j^{i,j} \mid E \in \mathcal{E}, i, j \in \mathcal{I} \}$.

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

3. $\rightarrow$ consists of loops, progress transitions (from $\rightarrow_{\mathcal{F}}$), and legal exits (cold cond./local inv.),

4. $Q_F = \{ C \in Q \mid \Theta(C) = \text{cold} \lor C = \mathcal{L} \}$ is the set of cold cuts and the maximal cut.
Recall: The TBA \( \mathcal{B}(L) \) of LSC \( L \) is \((C, Q, q_{\text{ini}}, \rightarrow, Q_F)\) with

- \( Q \) is the set of cuts of \( L \), \( q_{\text{ini}} \) is the instance heads cut,
- \( C = C_{\text{init}} \cup E_{\text{local}} \),
- \( \rightarrow \) consists of loops, progress transitions (from \( \rightarrow \)), 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.

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

\[
\rightarrow = \{(q, \psi_{\text{loop}}(q)) \mid q \in Q\} \cup \{(q, \psi_{\text{prop}}(q, q'), q') \mid q \rightarrow_F q' \} \cup \{(q, \psi_{\text{exit}}(q)) \mid q \in Q\}
\]
Loop Condition

\[ \psi_{\text{loop}}(q) = \psi_{\text{Mag}}(q) \land \psi_{\text{LocInv}}(q) \land \psi_{\text{Cond}}(q) \]

- \( \psi_{\text{Mag}}(q) = \neg \bigvee_{1 \leq i \leq n} \psi \in \text{Mag}(q_i \setminus q) \land (\text{strict} \implies \bigwedge_{\psi \in E_I^L \cap \text{Mag}(L)} \neg \psi) \)

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

- \( \psi_{\text{Cond}}(q) = \bigwedge_{(l,\ell) \in \text{Cond}, \Theta(\ell) = \theta, L \setminus (q_i \setminus q) \neq \emptyset} \psi \)

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

Progress Condition

\[ \psi_{\text{pin}}(q_i, q) = \psi_{\text{Mag}}(q, q_i) \land \psi_{\text{LocInv}}(q, q_i) \land \psi_{\text{Range}}(q_i, q) \]

- \( \psi_{\text{Mag}}(q, q_i) = \bigwedge_{\psi \in \text{Mag}(q_i \setminus q)} \psi \land \bigwedge_{j \neq i} \bigwedge_{\psi \in \text{Mag}(q_j \setminus q)} \neg \psi \)

- \( \psi_{\text{Cond}}(q, q_i) = \bigwedge_{(l,\ell) \in \text{Cond}, \Theta(\ell) = \theta, L \setminus (q_i \setminus q) \neq \emptyset} \psi \)

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

Local invariant \((l_0, l_0, \phi, l_1, i_1)\) is **active** at \(q\) if and only if

- \( l_0 \prec l \prec l_1, \text{ or} \)
- \( l = l_0 \land i_0 = \bullet, \text{ or} \)
- \( l = l_1 \land i_1 = \bullet \)

for some front location \(l\) of cut \((l)\) \(q\).
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
Symbolic Büchi Automata

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

\[ B = (C_D, Q, q_{init}, \rightarrow, Q_F) \]

where

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

Example:

\[ B_{sym}: Q = \{ (a, b, c, d) \rightarrow B \} \]
**Run of TBA**

**Definition.** Let $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ be a TBA and

$$w = \sigma_1, \sigma_2, \sigma_3, \ldots \in (C_B \rightarrow B)^\omega$$

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

An infinite sequence

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

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 $(q_i, \psi, q_{i+1}) \in \rightarrow$ s.t. $\sigma_i \models \psi_i$.

**Example:**

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

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**

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

over $w$ such that fair (or accepting) states are **visited infinitely often** by $\varrho$, 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:**

$$w = \{a \land b \rightarrow \text{true}, \overline{c \lor d} \rightarrow \text{true}\}, \{c\}, \{a, b\}, (\{d\}, \{a, b\})^\omega$$
Definition. **Software** is a finite description $S$ of a (possibly infinite) set $\{S\}$ of (finite or infinite) computation paths of the form

$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$$

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 $S$. 

---

**LSCs vs. Software**
Software Specification, formally

Definition. A software specification $\mathcal{S}$ is a finite description of a (possibly infinite) set $[\mathcal{S}]$ of softwares, i.e.

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

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

$\mathcal{S}$:

<table>
<thead>
<tr>
<th>$\cdot$</th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>off</td>
<td>$\times$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>on</td>
<td>$\times$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>go</td>
<td>$\times$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>stop</td>
<td>$\times$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

Define: $(S, [\cdot]) \in [\mathcal{S}]$ if and only if for all $\sigma_0 \xrightarrow{e_1} \sigma_1 \xrightarrow{e_2} \sigma_2 \cdots \in [S]$ and for all $i \in \mathbb{N}_0$.

$\exists \tau \in T \bullet \sigma_i \models \mathcal{F}(\tau)$.

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 $\sigma_0 \xrightarrow{e_1} \sigma_1 \xrightarrow{e_2} \sigma_2 \cdots$ 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:

$$\text{off} \xrightarrow{(off)} \text{on} \xrightarrow{(on)} \text{stop} \xrightarrow{(stop)} \text{off} \cdots$$
Software Satisfies Software Specification: Another Example

*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** \([\llbracket S \rrbracket]\) of \( S \) can be viewed as computation paths over the LSC’s observables.

*For example:*

- And then we can relate \( S \) to \( \mathcal{F} \).

---

The Plan: A Formal Semantics for a Visual Formalism

- Does the software satisfy the LSC?
- Apply construction procedure
- \( (\mathcal{L}, \preceq, \sim), \mathcal{I}, \text{Msg, Cond, LocInv, } \Theta ) \)
- Concrete syntax (diagram)
- Abstract syntax
- Semantics (Buchi automaton)
A software \( S \) is called \textit{compatible} with \textit{LSCs} \( L \) over \( C \) and \( E \) if and only if
\begin{itemize}
  \item \( \Sigma = (C \rightarrow B), C \subseteq C \), i.e. the \textit{states} comprise valuations of the conditions in \( C \),
  \item \( A = (B \rightarrow B), E^I \subseteq B \), i.e. the \textit{events} comprise valuations of \( E^I_i \), \( E^J_j \).
\end{itemize}

A computation path \( \pi = \sigma_0 \alpha_1 \sigma_2 \cdots \in [S] \) of software \( S \) \textit{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.
\[ W_S = \{ w(\pi) \mid \pi \in [S] \}. \]

\textbf{LSCs vs. Software (or Systems)}

\[
\begin{align*}
\ell & = \sigma_0 \sigma_1 \sigma_2 pSOFT \sigma_3 \sigma_4 \sigma_5 SOFT \cdots \in [S] \\
w(\ell) & = \{ \{ \}, \{ E_1^{UV}, E_1^{LV} \}, \{ pSOFT^{UV}, pSOFT^{LV} \}, \{ \}, \{ \}, \{ SOFT^{V,U}, SOFT^{V,U} \}, \{ \}, \ldots \} \in \text{Lang}(B(\mathcal{L}))
\end{align*}
\]

\( w = \{ \}, \{ E_1^{UV}, E_1^{LV} \}, \{ pSOFT^{UV}, pSOFT^{LV} \}, \{ \}, \{ \}, \{ SOFT^{V,U}, SOFT^{V,U} \}, \{ \}, \ldots \in \text{Lang}(B(\mathcal{L})) \)

\( \ell \)

\begin{itemize}
  \item \text{User}
  \item \text{Vend. Mach.}
\end{itemize}

\begin{itemize}
  \item \( E_1 \): \text{insert 1¢ coin}
  \item \( pSOFT \): \text{press 'SOFT' button}
  \item \( SOFT \): \text{dispense soft drink}
\end{itemize}
**LSCs vs. Software (or Systems)**

\[
\begin{align*}
\sigma_0 \xrightarrow{\tau} \sigma_1 & \xrightarrow{E_1^{U,V}} \sigma_2 \\
& \xrightarrow{p_{SOFT}^{U,V}} \sigma_3 \\
& \xrightarrow{\tau} \sigma_4 \\
& \xrightarrow{\tau} \sigma_5 \\
& \xrightarrow{SOFT^{V,U}} \sigma_6 \\
\end{align*}
\cdots \in [S]
\]

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

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

---

**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

Full LSC Syntax (without pre-chart)

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

- (non-empty) main-chart $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), I_M, Msg_M, Cond_M, LocInv_M, \Theta_M)$,
- activation condition $ac_0 \in \Phi(C)$,
- strictness flag 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

\[
\begin{align*}
\Theta_{\mathcal{L}} & \quad am = \text{initial} & am = \text{invariant} \\
\text{cold} & \quad \exists w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \land w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \quad \exists w \in W_S \exists k \in \mathbb{N}_0 \cdot w^k \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \land w^k \models \psi_{\text{prog}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L})) \\
\text{hot} & \quad \forall w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \implies w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \quad \forall w \in W_S \forall k \in \mathbb{N}_0 \cdot w^k \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \implies w^k \models \psi_{\text{hot}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))
\end{align*}
\]

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

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

\[
\begin{align*}
\Theta_{\mathcal{L}} & \quad am = \text{initial} & am = \text{invariant} \\
\text{cold} & \quad \exists w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \land w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \quad \exists w \in W_S \exists k \in \mathbb{N}_0 \cdot w^k \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \land w^k \models \psi_{\text{prog}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L})) \\
\text{hot} & \quad \forall w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \implies w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L})) \\
& \quad \forall w \in W_S \forall k \in \mathbb{N}_0 \cdot w^k \models ac \land \neg \psi_{\text{ext}}(C_0) \\
& \quad \implies w^k \models \psi_{\text{hot}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))
\end{align*}
\]

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 $1 \leq i \leq n$. 
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.
(Slightly) Advanced LSC Topics
A full LSC $\mathcal{L} = (PC, MC, ac_0, am, \Theta_\mathcal{L})$ consists of

- (non-empty) **main-chart** $MC = ((L_M, \leq_M, \sim_M), T_M, Msg_M, Cond_M, LocInv_M, \Theta_M)$.
- **activation condition** $ac_0 \in \Phi(C)$,
- **strictness flag** strict (if false, $\mathcal{L}$ is permissive)
- **activation mode** $am \in \{\text{initial}, \text{invariant}\}$,
- **chart mode** existential ($\Theta_\mathcal{L} = \text{cold}$) or universal ($\Theta_\mathcal{L} = \text{hot}$).

LSC Semantics with Pre-chart

<table>
<thead>
<tr>
<th>$am = \text{initial}$</th>
<th>$am = \text{invariant}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall w \in W \exists m \in \mathbb{N}_0 \bullet$</td>
<td>$\exists w \in W \exists k &lt; m \in \mathbb{N}_0 \bullet$</td>
</tr>
<tr>
<td>$\wedge w^k \models ac \land \neg \psi_{\text{cond}}(C_0^P) \land \psi_{\text{prog}}(\emptyset, C_0^P)$</td>
<td>$\wedge w^k \models ac \land \neg \psi_{\text{cond}}(C_0^P) \land \psi_{\text{prog}}(\emptyset, C_0^P)$</td>
</tr>
<tr>
<td>$\wedge w/1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
<td>$\wedge w/1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
</tr>
<tr>
<td>$w^{m+1} \models \neg \psi_{\text{cond}}(C_0^P)$</td>
<td>$w^{m+1} \models \psi_{\text{cond}}(C_0^M)$</td>
</tr>
<tr>
<td>$w^{m+1} \models \psi_{\text{prog}}(\emptyset, C_0^M)$</td>
<td>$w^{m+1} \models \psi_{\text{prog}}(\emptyset, C_0^M)$</td>
</tr>
<tr>
<td>$\wedge w/m + 2 \in \text{Lang}(B(MC))$</td>
<td>$\wedge w/m + 2 \in \text{Lang}(B(MC))$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\Theta_\mathcal{L} = \text{cold}$</th>
<th>$\Theta_\mathcal{L} = \text{hot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall w \in W \forall m \in \mathbb{N}_0 \bullet$</td>
<td>$\forall w \in W \forall k \leq m \in \mathbb{N}_0 \bullet$</td>
</tr>
<tr>
<td>$\wedge w^k \models ac \land \neg \psi_{\text{cond}}(C_0^P) \land \psi_{\text{prog}}(\emptyset, C_0^P)$</td>
<td>$\wedge w^k \models ac \land \neg \psi_{\text{cond}}(C_0^P) \land \psi_{\text{prog}}(\emptyset, C_0^P)$</td>
</tr>
<tr>
<td>$\wedge w/1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
<td>$\wedge w/1, \ldots, w/m \in \text{Lang}(B(PC))$</td>
</tr>
<tr>
<td>$w^{m+1} \models \neg \psi_{\text{cond}}(C_0^M)$</td>
<td>$w^{m+1} \models \psi_{\text{cond}}(C_0^M)$</td>
</tr>
<tr>
<td>$w^{m+1} \models \psi_{\text{prog}}(\emptyset, C_0^M)$</td>
<td>$w^{m+1} \models \psi_{\text{prog}}(\emptyset, C_0^M)$</td>
</tr>
<tr>
<td>$\wedge w/m + 2 \in \text{Lang}(B(MC))$</td>
<td>$\wedge w/m + 2 \in \text{Lang}(B(MC))$</td>
</tr>
</tbody>
</table>

where $C_0^P$ and $C_0^M$ are the minimal (or instance heads) cuts of pre- and main-chart.
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.
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**
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 $\pi \in [S]$ 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)
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! (*: 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 $S \nmid \mathcal{L}$ by demonstrating one $\pi$ such that $w(\pi)$ is not accepted by $\mathcal{L}$.

---

Pushing Things Even Further

*(Harel and Marelly, 2003)*
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
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

- **Negotiation.** (with customer, marketing department, or ...)
  - 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

- **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 a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → systematic testing hardly possible

- **Acceptance by customer, resolving later objections or regress claims.**
  - Without specification, 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 → risk of unexpected changes
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).

---

Formalisation Validation

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 $v$, 
3. FM expert evaluates DT on $v$, 
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
References

