Softwaretechnik / Software-Engineering

Lecture 10: Live Sequence Charts & RE Wrap-Up

2018-06-11

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Introduction

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

Definition: Software & SW Specification

Wrap-Up
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
    - Software and Software Specification, formally
    - Software satisfies Software Specification
  - 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

- Outlook: Formal Methods in Design & QA
LSC Semantics: TBA Construction
Language of LSC Body: Example
The TBA $B(\mathcal{L})$ of LSC $\mathcal{L}$ over $C$ and $E$ is $(\mathcal{C}_B, Q, q_{ini}, \rightarrow, Q_F)$ with

- $\mathcal{C}_B = C \cup \mathcal{E}_{1?}^T$, where $\mathcal{E}_{1?}^T = \{ E_1^{i,j}, E_2^{i,j} \mid E \in \mathcal{E}, i, j \in I \}$,
- $Q$ is the set of cuts of $\mathcal{L}$, $q_{ini}$ is the instance heads cut,
- $\rightarrow$ consists of loops, progress transitions (from $\leadsto \mathcal{F}$), 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 $\mathcal{B}(\mathcal{L})$ of LSC $\mathcal{L}$ is $(C, Q, q_{ini}, \rightarrow, Q_F)$ with

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- \(\psi_{\text{Msg}}(q) = \neg \bigvee_{1 \leq i \leq n} \psi_{\text{Msg}}(q, q_i) \land (\text{strict} \implies \bigwedge_{\psi \in \mathcal{E} \cap \text{Msg}(\mathcal{L})} \neg \psi)
\)

\[= : \psi_{\text{strict}}(q)\]

- \(\psi_{\text{LocInv}}^{\theta}(q) = \bigwedge_{\ell = (l, \ell_0, \phi, l_0, l_1, \ell_1) \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 \(\not\exists l' \in C \bullet l \prec l'\).

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

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

- \(\text{Msg}(\mathcal{F}_1, \ldots, \mathcal{F}_n) = \bigcup_{1 \leq i \leq n} \text{Msg}(\mathcal{F}_i)\)
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 \(\bullet\)-active at \(q\) if and only if

- \(l_0 < l < l_1\), or
- \(l = l_0 \land \iota_0 = \bullet\), 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
    - Software and Software Specification, formally
    - Software satisfies Software Specification
  - 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

- Outlook: Formal Methods in Design & QA
Excursion: Symbolic Büchi Automata
From Finite Automata to Symbolic Büchi Automata

$A$: $\Sigma = \{0, 1\}$

$B$: $\Sigma = \{0, 1\}$

$B'$: $\Sigma = \{0, 1\}$

$A_{sym}$: $\Sigma = (\{a, b, c, d\} \rightarrow B)$

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

$\omega = 01010(0\ldots \in L(B) = \omega(01))\omega$

$\omega = 01010(0\ldots \in L(B') = ?$

$\mathcal{L}(A) = \{01\}^*$

e.g. $01010_0$

$0_1^4$

$0_1^1$  $0_1^0$

$B$: $\Sigma = \{0, 1\}$

$B'$: $\Sigma = \{0, 1\}$

$\mathcal{L}(B) = ?$

$\mathcal{L}(B') = ?$

$C = \{a, b, c, d\}$

$A_{sym}$: $\Sigma = (\{a, b, c, d\} \rightarrow B)$

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

$\mathcal{L}(A_{sym})$

$\mathcal{L}(B_{sym})$

$\mathcal{L}(B_A)$

$\mathcal{L}(B_A)$
Definition. A Symbolic Büchi Automaton (TBA) is a tuple

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

where

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

Example:

$$\mathcal{B}_{sym}: \quad \Sigma = \{a, b, c, d\} \rightarrow \mathcal{B}$$
**Definition.** Let $\mathcal{B} = (\mathcal{C}_\mathcal{B}, Q, q_{\text{ini}}, \rightarrow, Q_F)$ be a TBA and

\[ w = \sigma_1, \sigma_2, \sigma_3, \ldots \in (\mathcal{C}_\mathcal{B} \rightarrow \mathbb{B})^\omega \]

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

An infinite sequence

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

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

- $q_0 = q_{\text{ini}}$,
- 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:**

$\Sigma = (\{a, b, c, d\} \rightarrow \mathbb{B})$

$\mathcal{B}_{\text{sym}}$: $\{a, b\}$ for short

\[ w = \{a \mapsto \text{true}, b \mapsto \text{true}, c \mapsto \text{false}, d \mapsto \text{false}\}, \{c\}, \{a, b\}, (\{d\}, \{a, b\})^\omega \]
**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 $Lang(B) \subseteq (C_B \rightarrow B)^\omega$ of words that are accepted by $B$ the language of $B$.

---

**Example:**

Given TBA $B_{sym}$:

$$\Sigma = \{a, b, c, d\} \rightarrow B$$

with

- $B_{sym} : a \land b$ to $q_1$
- $B_{sym} : c \lor d$ to $q_2$
- $B_{sym} : q_1 \rightarrow q_2$
LSCs vs. Software
\[
\begin{align*}
\mathcal{L} = & \{ \}, \{ E_1^{U,V}, E_1^{U,V} \}, \{ pSOFT_1^{U,V}, pSOFT_2^{U,V} \}, \{ \}, \{ \}, \{ \}, \{ SOFT_1^{V,U}, SOFT_2^{V,U} \}, \{ \}, \ldots \in \text{Lang}(B(\mathcal{L})) \\
E_1: & \quad \text{insert 1€ coin} \\
pSOFT: & \quad \text{press ‘SOFT’ button} \\
SOFT: & \quad \text{dispense soft drink}
\end{align*}
\]
Excursion: Software Specification, Formally
Recall:

- We would like to **precisely** and **objectively** specify the **set of allowed softwares** that make the customer happy. (The designers and developers then choose one from the set.)

- In other words, we want to formally define a **satisfies** relation between softwares and software specifications.

  That is, given a software $S$ and a software specification $\mathcal{P}$, we want to define when (and only when) software $S$ **satisfies** software specification $\mathcal{P}$, denoted by

  $$S \models \mathcal{P}.$$ 

- Once again:
  - $S \models \mathcal{P}$: specification is **satisfied**, $S$ is one “allowed” design, should be accepted.
  - $S \not\models \mathcal{P}$: specification is **not satisfied**, $S$ may not satisfy customer’s needs.
Definition. **Software** is a finite description $S$ of a (possibly infinite) set $\llbracket S \rrbracket$ 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 $\llbracket \cdot \rrbracket : S \mapsto \llbracket S \rrbracket$ is called interpretation of $S$.

**Example**: room ventilation system

- $\Sigma = \{ b, on, off, go, stop \} \rightarrow B$, $A = \{ \tau \}$.
- computation path:

\[
\begin{pmatrix}
  b & \mapsto & false \\
  on & \mapsto & false \\
  off & \mapsto & true \\
  go & \mapsto & false \\
  stop & \mapsto & false
\end{pmatrix}
= ( off ) \xrightarrow{\tau} ( b \begin{pmatrix}
  b \\
  on
\end{pmatrix} \xrightarrow{\tau} ( on ) \xrightarrow{\tau} ( b \begin{pmatrix}
  b \\
  on \\
  stop
\end{pmatrix} \xrightarrow{\tau} ( off ) \cdots
\]
**Software, formally**

**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 $S$.

**Example:** vending machine

- computation path:

$$
\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E_1^{U,V}} \sigma_2 \xrightarrow{p\text{SOFT}^{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{\text{SOFT}^{V,U}} \cdots
$$

- machine switched on
- user inserts $1 \in \mathbb{N}$
- buttons light up
- user presses ‘SOFT’ button
- prepare dispenser
- drink ready
- notify user
**Software Satisfies Software Specification**

**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>$-$</td>
</tr>
<tr>
<td>$\text{off}$ ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>$\text{on}$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>$\text{go}$ start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$\text{stop}$ stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

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

$$\sigma_0 \xrightarrow{a_1} \sigma_1 \xrightarrow{a_2} \sigma_2 \cdots \in [S]$$

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

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

Software

- Assume we have a program $S$ for the room ventilation controller.
- Assume we can observe at well-defined points in time the conditions $b$, $\text{off}$, $\text{on}$, $\text{go}$, $\text{stop}$ 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 $b$, $\text{off}$, $\text{on}$, $\text{go}$, $\text{stop}$, i.e. $\sigma_i : \{b, \text{off}, \text{on}, \text{go}, \text{stop}\} \rightarrow \mathbb{B}$. 
Software Satisfies Software Specification: Another Example

Software Specification

\( \mathcal{S} : \)

\[\begin{array}{|c|}
\hline
\text{LSC: } & \text{buy water} \\
\text{AC: } & \text{true} \\
\text{API: } & \text{invariant } I : \text{ strict} \\
\hline
\end{array}\]

User \quad CoinValidator \quad ChoicePanel \quad Dispenser

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

\( \text{tja... (in a minute)} \)

Software

\begin{itemize}
\item Assume we can observe at well-defined points in time the observables relevant for the LSC (conditions and messages) when the software \( S \) runs.
\item Then the behaviour \( [S] \) of \( S \) can be viewed as computation paths over the LSC’s observables.
\item And then we can relate \( S \) to \( \mathcal{S} \).
\end{itemize}
Back to LSCs vs. Software
A software $S$ is called **compatible** with LSC $\mathcal{L}$ over $\mathcal{C}$ and $\mathcal{E}$ if and only if

- $\Sigma = (C \to \mathbb{B}), \mathcal{C} \subseteq C$, i.e. the **states** comprise valuations of the conditions in $\mathcal{C}$,
- $A = (B \to \mathbb{B}), \mathcal{E}^\mathcal{I} \subseteq B$, i.e. the **events** comprise valuations of $E^{i,j}_1, E^{i,j}_2$.

A computation path $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in \llbracket S \rrbracket$ of software $S$ **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 $\llbracket S \rrbracket$, i.e.

$$W_S = \{w(\pi) \mid \pi \in \llbracket S \rrbracket\}.$$
LSCs vs. Software (or Systems)

\[ \tau \rightarrow \sigma_1 \xrightarrow{E_1} \sigma_2 \xrightarrow{pSOFT} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT} \cdots \in [S] \]

\( w(\pi) = \sigma_0, (\sigma_1 \cup \{E_1, E_1\}), (\sigma_2 \cup \{pSOFT, pSOFT\}), \sigma_3, \sigma_4, \sigma_5, \)

\( (\sigma_6 \cup \{SOFT, SOFT\}), \ldots \)

\[ \{\}, \{E_1, E_1\}, \{pSOFT, pSOFT\}, \{\}, \{\}, \{\}, \{SOFT, SOFT\}, \{\}, \ldots \]

\( \in \text{Lang}(B(L)) \)

---

**User**

**Vend. Mach.**

**E1**: insert 1€ coin

**pSOFT**: press ‘SOFT’ button

**SOFT**: dispense soft drink

---

TBA over \( C_B = C \cup \mathcal{E} ? \)

where \( C = \emptyset \) and 
\( \mathcal{E} ? = \{E_1, E_1\}, \)
\( E_1, pSOFT, pSOFT, \)
\( SOFT, SOFT, \)
\( \ldots \}. \)
The Plan: A Formal Semantics for a Visual Formalism

concrete syntax
(diagram)

abstract syntax

read out relevant information

apply construction procedure

semantics
(Büchi automaton)

software

does the software satisfy the LSC?

(((L, ≤, ~), I, Msg, Cond, LocInv, Θ))

read out relevant information

does the software satisfy the LSC?
Content

- Live Sequence Charts
  - TBA Construction
  - LSCs vs. Software
    - Software and Software Specification, formally
    - Software satisfies Software Specification
  - 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
- Outlook: Formal Methods in Design & QA
Activation Condition and Mode
A full LSC $\mathcal{L} = (MC, ac_0, am, \Theta_\mathcal{L})$ consists of

- (non-empty) **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, Msg_M, Cond_M, LocInv_M, \Theta_M),$
- **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** ($\Theta_\mathcal{L} = \text{hot}$).
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

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

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

where and $C_0$ is the minimal (or **instance heads**) cut of the main-chart.
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>$\Theta_\mathcal{L}$</th>
<th>$am = \text{initial}$</th>
<th>$am = \text{invariant}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cold</strong></td>
<td>$\exists w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{exit}}(C_0)$ $\land w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\exists w \in W_S \exists k \in \mathbb{N}<em>0 \cdot w^k \models ac \land \neg \psi</em>{\text{exit}}(C_0)$ $\land w^k \models \psi_{\text{prog}}(\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))$</td>
</tr>
<tr>
<td><strong>hot</strong></td>
<td>$\forall w \in W_S \cdot w^0 \models ac \land \neg \psi_{\text{exit}}(C_0)$ $\implies w^0 \models \psi_{\text{prog}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\forall w \in W_S \forall k \in \mathbb{N}<em>0 \cdot w^k \models ac \land \neg \psi</em>{\text{exit}}(C_0)$ $\implies w^k \models \psi_{\text{hot}}^{\text{cond}}(\emptyset, C_0) \land w/k+1 \in \text{Lang}(B(\mathcal{L}))$</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.
Live Sequence Charts
- TBA Construction
- LSCs vs. Software
  - Software and Software Specification, formally
  - Software satisfies Software Specification
- 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

Outlook: Formal Methods in Design & QA
(Slightly) Advanced LSC Topics
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** $am \in \{\text{initial}, \text{invariant}\}$,
- **chart mode** existential ($\Theta_{\mathcal{L}} = \text{cold}$) or **universal** ($\Theta_{\mathcal{L}} = \text{hot}$).
where $C_0^P$ and $C_0^M$ 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.
Live Sequence Charts

- TBA Construction

- LSCs vs. Software
  - Software and Software Specification, formally
  - Software satisfies Software Specification

- 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

Outlook: Formal Methods in Design & QA
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.”
  \((\rightarrow \text{positive use-cases, existential LSC})\)
- **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.”
  \((\rightarrow \text{negative use-cases, LSC with pre-chart and hot-false})\)

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

- **Investigate** *preconditions, side-conditions, exceptional cases* and *corner-cases*.
  \((\rightarrow \text{extend use-cases, refine LSCs with conditions or local invariants})\)
- **Generalise** into universal requirements, e.g., *universal LSCs*.
- **Validate** with customer using new positive / negative scenarios.
Strengthening Scenarios Into Requirements

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

Needs

Solution

Customer Developer

Announcement (Lastenheft)

Customer Developer

Offer (Pflichtenheft)

Customer Developer

Software contract (incl. Pflichtenheft)

Developer software delivery
• Ask customer for (pos./neg.) scenarios, note down as existential LSCs:

![Diagram of scenarios](image-url)
**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
How to Prove that a Software Satisfies an LSC?

- Software $S$ satisfies existential LSC $L$ if there exists $\pi \in \llbracket S \rrbracket$ such that $L$ accepts $w(\pi)$. Prove $S \models L$ 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 $\pi \in [S]$ 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! (*: 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 $S \nvDash \mathcal{L}$ 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
Topic Area Requirements Engineering: Content

- Introduction
- 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
- Definition: Software & SW Specification
- 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 …)

**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-implementation,**
- 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, 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

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

- difficult integration
Requirements Analysis in a Nutshell

- 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 $\sigma$.
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)
Big-Picture Outlook
Example: Software Specification

Alphabet:

- $M$ – dispense cash only,
- $C$ – return card only,
- $M/C$ – dispense cash and return card.

- **Customer 1:** “don’t care”
  \[
  \mathcal{S}_1 = (M.C|C.M|M/C)\omega
  \]

- **Customer 2:** “you choose, but be consistent”
  \[
  \mathcal{S}_2 = (M.C)\omega \text{ or } (C.M)\omega
  \]

- **Customer 3:** “consider human errors”
  \[
  \mathcal{S}_3 = (C.M)\omega
  \]
Customer 2

Mmmh, Software!

validate

analyse

verify

Requirements

\[ [\mathcal{S}_1] = \{ (M.C, [\cdot]_1), (C.M, [\cdot]_1) \} \]

\[ [\mathcal{S}_2] = \{ (M.T\cdot M.C, [\cdot]_1), (C.T\cdot M, [\cdot]_1) \} \]

\[ [S_1] = \{ \sigma_0^1 \xrightarrow{\alpha_1^1} \sigma_1^1 \xrightarrow{\alpha_2^1} \sigma_2^1 \cdot \cdot \cdot , \ldots \} \]

Design

Development Process/Project Management

verify

analyse

Implementation

\[ [S_2] = \{ \sigma_0^2 \xrightarrow{\alpha_1^2} \sigma_1^2 \xrightarrow{\alpha_2^2} \sigma_2^2 \cdot \cdot \cdot , \ldots \} \]
References
References

