Contents & Goals

Last Lecture:
- TBA: automata for infinite words
- Cuts and firedsets of an LSC body
- TBA-construction for LSC body

This Lecture:
- **Educational Objectives:** Capabilities for following tasks/questions.
  - what is the existential/universal, initial/invariant interpretation of an LSC?
  - Given a set of LSCs, give a computation path which is (not) accepted by the LSCs.
  - Given a set of LSCs, which scenario/anti-scenario/requirement is formalised by them?
  - Formalise this positive scenario/anti-scenario/requirement using LSCs.
  - Could there be a relation between LSC (anti-)scenarios and testing?

- **Content:**
  - Full LSCs
  - Existential LSCs (scenarios)
  - pre-charts, universal LSCs
  - Requirements Engineering: conclusions
Recall: TBA Construction and Full LSC

Example

Finally: The LSC Semantics

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

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

Concrete syntax:
Finally: The LSC Semantics

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

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

A set of words $W \subseteq (C \rightarrow B)^\omega$ is **accepted** by $\mathcal{L}$ 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>cold</td>
<td>$\exists w \in W \bullet w^0 \models ac \land w^0 \models \psi_{\text{Cond}}^{\text{hot}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\exists w \in W \exists k \in \mathbb{N}<em>0 \bullet w^k \models ac \land w^k \models \psi</em>{\text{Cond}}^{\text{hot}}(\emptyset, C_0) \land w^k/1 \in \text{Lang}(B(\mathcal{L}))$</td>
</tr>
<tr>
<td>hot</td>
<td>$\forall w \in W \bullet w^0 \models ac \implies w^0 \models \psi_{\text{Cond}}^{\text{cold}}(\emptyset, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$</td>
<td>$\forall w \in W \forall k \in \mathbb{N}<em>0 \bullet w^k \models ac \implies w^k \models \psi</em>{\text{Cond}}^{\text{cold}}(\emptyset, C_0) \land w^k/1 \in \text{Lang}(B(\mathcal{L}))$</td>
</tr>
</tbody>
</table>

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

LSC: $\mathcal{L}_1$
AC: $c_1$
AM: initial I: permissive

$I_1$ $I_2$ $I_3$

$E$ $F$ $G$

LSC: $\mathcal{L}_1$
AM: initial I: permissive

$I_1$ $I_2$ $I_3$

c$_1$

$E$ $F$ $G$
LSCs vs. Software
Let $S$ be a software with $[S] = \{ \pi = (\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots | \cdots) \}$. $S$ is called **compatible** with LSC $\mathcal{L}$ over $C$ and $E$ if and only if

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

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

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \ldots$$
Let $S$ be a software with $[S] = \{ \pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots | \cdots \}$. 

$S$ is called compatible with LSC $L$ over $C$ and $E$ is if and only if

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

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

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

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

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

<table>
<thead>
<tr>
<th>$\Theta_L$</th>
<th>$am = \text{initial}$</th>
<th>$am = \text{invariant}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>$\exists w \in W \cdot w^0 \models ac \land$</td>
<td>$\exists w \in W \exists k \in \mathbb{N}_0 \cdot w^k \models ac \land$</td>
</tr>
<tr>
<td></td>
<td>$w^0 \models \psi_{\text{hot}}^\text{Cond} (\emptyset, C_0) \land w/1 \in \text{Lang}(B(L))$</td>
<td>$w^k \models \psi_{\text{hot}}^\text{Cond} (\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(L))$</td>
</tr>
<tr>
<td>hot</td>
<td>$\forall w \in W \cdot w^0 \models ac \implies$</td>
<td>$\forall w \in W \forall k \in \mathbb{N}_0 \cdot w^k \models ac \implies$</td>
</tr>
<tr>
<td></td>
<td>$w^0 \models \psi_{\text{hot}}^\text{Cond} (\emptyset, C_0) \land w/1 \in \text{Lang}(B(L))$</td>
<td>$w^k \models \psi_{\text{hot}}^\text{Cond} (\emptyset, C_0) \land w/k + 1 \in \text{Lang}(B(L))$</td>
</tr>
</tbody>
</table>

Software $S$ satisfies a set of LSCs $L_1, \ldots, L_n$ if and only if $S \models L_i$ for all $1 \leq i \leq n$. 
One quite effective approach: try to **approximate** the requirements with positive and negative **scenarios**.

- Dear customer, please describe example usages of the desired system. **“If the system is not at all able to do this, then it’s not what I want.”**
- Dear customer, please describe behaviour that the desired system must not show. **“If the system does this, then it’s not what I want.”**
- From there on, refine and generalise: what about exceptional cases? what about corner-cases? etc.
Example: Buy A Softdrink

LSC: buy softdrink
AC: true
AM: invariant I: permissive

User | Vend. Ma.

\[ E_1 \]
\[ pSOFT \]
\[ SOFT \]

\[ \sigma_0 \xrightarrow{E_1} \sigma_1, \quad \sigma_2 \xrightarrow{pSOFT} \sigma_3, \quad \sigma_4, \ldots \] satisfies 'buy softdrink'
Example: Get Change

- LSC: get change
- AC: true
- AM: invariant I: permissive

User → Vend. Ma.

C50

E1

pSOFT

SOFT

chg-C50
Example: Don’t Give Two Drinks

LSC: only one drink
AC: true
AM: invariant I: permissive

User   Vend. Ma.

$E_1$

$pSOFT$

$SOFT$

$SOFT$

$\neg C50! \land \neg E_1!$

false

$\Theta(\delta) = \text{bad}$
A **full LSC** \( \mathcal{L} = (PC, MC, ac_0, am, \Theta_\mathcal{L}) \) **actually** consist of

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

**LSC:** only one drink  
**AC:** true  
**AM:** invariant I: permissive

<table>
<thead>
<tr>
<th></th>
<th>( \Theta \mathcal{L} )</th>
<th>( am = \text{initial} )</th>
<th>( am = \text{invariant} )</th>
</tr>
</thead>
</table>
| **cold** | \( \exists w \in W \ \exists m \in \mathbb{N}_0 \cdot w^0 \models ac \)  
\( \land w^0 \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^P) \)  
\( \land w/1, \ldots, w/m \in \text{Lang}(\mathcal{B}(PC)) \)  
\( \land w^{m+1} \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \land w/m + 1 \in \text{Lang}(\mathcal{B}(MC)) \) | \( \exists w \in W \ \exists k < m \in \mathbb{N}_0 \cdot w^k \models ac \)  
\( \land w^k \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^P) \)  
\( \land w/k + 1, \ldots, w/m \in \text{Lang}(\mathcal{B}(PC)) \)  
\( \land w^{m+1} \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \land w/m + 1 \in \text{Lang}(\mathcal{B}(MC)) \) | |  
| **hot** | \( \forall w \in W \cdot w^0 \models ac \)  
\( \land w^0 \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^P) \)  
\( \land w/1, \ldots, w/m \in \text{Lang}(\mathcal{B}(PC)) \)  
\( \land w^{m+1} \models \psi_{\text{cold}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \implies w^{m+1} \models \psi_{\text{cold}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \land w/m + 1 \in \text{Lang}(\mathcal{B}(MC)) \) | \( \forall w \in W \ \forall k \leq m \in \mathbb{N}_0 \cdot w^k \models ac \)  
\( \land w^k \models \psi_{\text{hot}}^\text{Cond}(\emptyset, C_0^P) \)  
\( \land w/k + 1, \ldots, w/m \in \text{Lang}(\mathcal{B}(PC)) \)  
\( \land w^{m+1} \models \psi_{\text{cold}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \implies w^{m+1} \models \psi_{\text{cold}}^\text{Cond}(\emptyset, C_0^M) \)  
\( \land w/m + 1 \in \text{Lang}(\mathcal{B}(MC)) \) |
**Note: Scenarios and Acceptance Test**

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

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

Customer

requirements analysis

Analyst

(Σ × A)^ω
Universal LSC: Example

LSC: buy water
AC: true
AM: invariant I: strict

User
CoinValidator
ChoicePanel
Dispenser

C50
pWATER
water_in_stock
dWATER
OK

CSO
E1
SOFT
WATER WIS

CSO
E1
SOFT
WATER WIS

OK
Universal LSC: Example

LSC: buy water
AC: true
AM: invariant I: strict

User ─ CoinValidator ─ ChoicePanel ─ Dispenser

¬(C50! ∨ E1! ∨ pSOFT! ∨ pTEA! ∨ pFILLUP!)
Universal LSC: Example

LSC: buy water
AC: true
AM: invariant I: strict

User -> CoinValidator -> ChoicePanel -> Dispenser

¬(C50! ∨ E1! ∨ pSOFT! ∨ pTEA! ∨ pFILLUP!)
¬(dSoft! ∨ dTEA!)

water_in_stock

C50 → pWATER
pWATER → water_in_stock

OK → dWATER
Shortcut: Forbidden Elements

Forbidden Elements: dSOFT, dTEA

only use event expressions like E! or E? regarded in Cond !Loc!ut
Modelling Idiom: Enforcing Order

LSC: $\mathcal{L}$
AM: invariant I: permissive

$I_1$ $I_2$ $I_3$ $I_4$

$E$ $F$

LSC: $\mathcal{L}$
AM: invariant I: permissive

$I_1$ $I_2$ $I_3$ $I_4$

$E$ ~ $true$

$F$
A requirements specification should be

- **correct**
  - it correctly represents the wishes/needs of the customer,

- **complete**
  - all requirements (existing in somebody’s head, or a document, or ...) should be present,

- **relevant**
  - things which are not relevant to the project should not be constrained,

- **consistent, free of contradictions**
  - each requirement is compatible with all other requirements; otherwise the requirements are not realisable,

- **neutral, abstract**
  - a requirements specification does not constrain the realisation more than necessary,

- **traceable, comprehensible**
  - the sources of requirements are documented, requirements are uniquely identifiable,

- **testable, objective**
  - the final product can objectively be checked for satisfying a requirement.
Requirements on LSC Specifications

- **correctness** is relative to “in the head of the customer” → still difficult;

- **complete**: we can at least define a kind of relative completeness in the sense of “did we cover all (exceptional) cases?”;

- **relevant** also not analyzable within LSCs;

- **consistency** can formally be analysed!

- **neutral/abstract** is relative to the realisation → still difficult;
  But LSCs tend to support abstract specifications; specifying technical details is tedious.

- **traceable/comprehensible** are meta-properties, need to be established separately;

- a formal requirements specification, e.g. using LSCs, is immediately **objective/testable**.

For Decision Tables, we formally defined additional quality criteria:

- **uselessness/vacuity**,  
  - pre-chart is not satisfiable  
  - system behaviour never satisfies pre-chart  

- **determinism** may be desired,

- **consistency** wrt. domain model.

What about LSCs?
LSCs vs. MSCs
Recall: Most severe drawbacks of, e.g., MSCs:

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

(Harel and Marelly, 2003)
Requirements Engineering Wrap-Up
Recall: Software Specification Example

**Alphabet:**

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

- **Customer 1** “don’t care”
  
  $$(M.C | C.M | M_C)$$

- **Customer 2** “you choose, but be consistent”
  
  $(M.C)$ or $(C.M)$

- **Customer 3** “consider human errors”
  
  $(C.M)$
Recall: Formal Software Development

Mmmh, Software!

Requirements

\[ S_1 = \{(M.C, [\cdot]_1), (C.M, [\cdot]_1)\} \]

Design

\[ S_2 = \{(M.TM.C, [\cdot]_1), (C.TC.M, [\cdot]_1)\} \]

Implementation

\[ S = \{\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{\tau} \sigma_2 \ldots, \ldots\} \]

Development Process/Project Management

Validation

Verification

- 10 - 2015-06-15 - Swrapup -

26/31
Recall: Formal Software Development

Mmmh, Software!

\[ S_1 = \{(M.C, [ \cdot ]_1), (C.M, [ \cdot ]_1)\} \]

Requirements

Development Process/Project Management

elicit req.(in.)

requirements elicitation

formalise req.(fo.)

requirements formalisation

verify req.(fo.)

requirements verification

fix req.(fo.)

requirements repair

validate req.(in.)

requirements validation
One sometimes distinguishes:

- **Systems Engineering** (develop software for an embedded controller)
  
  Requirements typically stated in terms of *system observables* ("press WATER button"), needs to be mapped to terms of the software!

- **Software Engineering** (develop software which interacts with other software)
  
  Requirements stated in terms of the software.

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

- **Once again** (can it be mentioned too often?):
  
  Distinguish *domain elements* and *software elements* and (try to) keep them apart to avoid confusion.
Lehmann (Lehman, 1980; Lehman and Ramil, 2001) distinguishes three classes of software (my interpretation, my examples):

- **S-programs**: solve mathematical, abstract problems; can exactly (in particular formally) be specified; tend to be small; can be developed once and for all.
  
  **Examples**: sorting, compiler (!), compute $\pi$ or $\sqrt{-1}$, cryptography, textbook examples, …

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

  **Examples**: cruise control, autopilot, traffic lights controller, plant automatisation, …

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

  **Examples**: basically everything else; word processor, web-shop, game, smart-phone apps, …
(Rupp and die SOPHISTen, 2014)
References

