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
Finally: The LSC Semantics

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

$\Theta_{\mathcal{L}}$

$\text{am = initial}$

$\forall w \in W \quad w^0 \models ac \land w^0 \models \psi_{\text{Cond}}(0, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$

$\exists w \in W \exists k \in \mathbb{N}_0 \cdot w^k \models ac \land w^k \models \psi_{\text{Cond}}(0, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))$

$\text{am = invariant}$

$\forall w \in W \quad w^0 \models ac \iff w^0 \models \psi_{\text{Cond}}(0, C_0) \land w/1 \in \text{Lang}(B(\mathcal{L}))$

$\forall w \in W \forall k \in \mathbb{N}_0 \cdot w^k \models ac \iff w^k \models \psi_{\text{Cond}}(0, C_0) \land w/k + 1 \in \text{Lang}(B(\mathcal{L}))$

where $ac = ac_0 \land \psi_{\text{Cond}}^{\text{cold}}(0, C_0) \land \psi_{\text{Msg}}(0, C_0)$; $C_0$ is the minimal (or instance heads) cut.
Activation Condition

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 $L$ over $C$ and $E$ is if and only if

- $\Sigma = (C \rightarrow B)$, 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 $E!$, $E?$):

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

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

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

Software $S$ satisfies a set of LSCs $L_1, \ldots, L_n$ if and only if $S \models L_i$ for all $1 \leq i \leq n$. 

- **Theta**
Recall: The Crux of Requirements Engineering

One quite effective approach: try to approximate the requirements with positive and negative scenarios.

- Dear customer, please describe example usages of the desired system.
  
  "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

<table>
<thead>
<tr>
<th>LSC</th>
<th>buy softdrink</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>true</td>
</tr>
<tr>
<td>AM</td>
<td>invariant I:</td>
</tr>
<tr>
<td></td>
<td>permissive</td>
</tr>
</tbody>
</table>

User | Vend. Ma.

E1
pSOFT
SOFT

User

"prefer"
"can’t be"

**Example: Get Change**

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

User ➔ C50 ➔ E1 ➔ pSOFT ➔ SOFT ➔ chg-C50

**Example: Don’t Give Two Drinks**

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

User ➔ E1 ➔ pSOFT ➔ SOFT ➔ SOFT ➔ ¬C50! ∧ ¬E1!
A full LSC $L = (PC, MC, ac_0, am, \Theta_{\tau'})$ actually consist of
- main-chart $MC = ((L_M, \preceq_M, \sim_M), I_M, Msg_M, Cond_M, LocInv_M, \Theta_M)$ (non-empty),
- activation condition $ac \in \Phi(C)$, strictness flag $strict$ (otherwise called permissive)
- activation mode $am \in \{initial, invariant\}$,
- chart mode existential ($\Theta_{\tau'} = cold$) or universal ($\Theta_{\tau'} = hot$).

\[\exists w \in W \forall m \in \mathbb{N}_0 \bullet w^0 \models ac\]
\[\land w^0 \models \psi_{hot}(0, C^0_P)\]
\[\land w/1, \ldots, w/m \in \text{Lang}(B(PC))\]
\[\land w^{m+1} \models \psi_{hot}(0, C^M_P)\]
\[\land w/m + 1 \in \text{Lang}(B(MC))\]

\[\exists w \in W \exists k < m \in \mathbb{N}_0 \bullet w^k \models ac\]
\[\land w^k \models \psi_{hot}(0, C^k_P)\]
\[\land w/k + 1, \ldots, w/m \in \text{Lang}(B(PC))\]
\[\land w^{m+1} \models \psi_{hot}(0, C^M_P)\]
\[\land w/m + 1 \in \text{Lang}(B(MC))\]

\[\forall w \in W \bullet w^0 \models ac\]
\[\land w^0 \models \psi_{cond}(0, C^0_P)\]
\[\land w/1, \ldots, w/m \in \text{Lang}(B(PC))\]
\[\land w^{m+1} \models \psi_{cond}(0, C^M_P)\]
\[\Rightarrow w^{m+1} \models \psi_{cond}(0, C^M_P)\]
\[\land w/m + 1 \in \text{Lang}(B(MC))\]

\[\forall w \in W \forall k \leq m \in \mathbb{N}_0 \bullet w^k \models ac\]
\[\land w^k \models \psi_{cond}(0, C^k_P)\]
\[\land w/k + 1, \ldots, w/m \in \text{Lang}(B(PC))\]
\[\land w^{m+1} \models \psi_{cond}(0, C^M_P)\]
\[\Rightarrow w^{m+1} \models \psi_{cond}(0, C^M_P)\]
\[\land w/m + 1 \in \text{Lang}(B(MC))\]
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.)

**Strengthening Scenarios Into Requirements**
Universal LSC: Example

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

User CoinValidator ChoicePanel Dispenser

C50

¬ (C50 \lor E1 \lor pSOFT \lor pTEA \lor pFILLUP)

water in stock

dWATER OK
Universal LSC: Example

Shortcut: Forbidden Elements

Forbidden Elements: SOFT, TEA
Modelling Idiom: Enforcing Order

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**;

- **determinism** may be desired;

- **consistency** wrt. domain model.

What about LSCs?

**LSCs vs. MSCs**
**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**

(ITU-T, 2011)

---

**Pushing It Even Further**

(There is a diagram here, but it is not described in the text.)

(Harel and Marelly, 2003)
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”
  
  $\left( M.C \bigg| C.M \bigg| \begin{array}{c} M \\ C \end{array} \right)$

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

- **Customer 3** “consider human errors”
  
  $(C.M)$
Final Remarks

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.

Systems vs. Software Engineering

A Classification of Software

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 \( \sqrt{\cdot} \), 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 trystich 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, . . .
Literature Recommendation

(Rupp and die SOPHISTen, 2014)

References
References