Softwaretechnik / Software-Engineering

Lecture 10: Live Sequence Charts Cont'd

2015-06-15

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

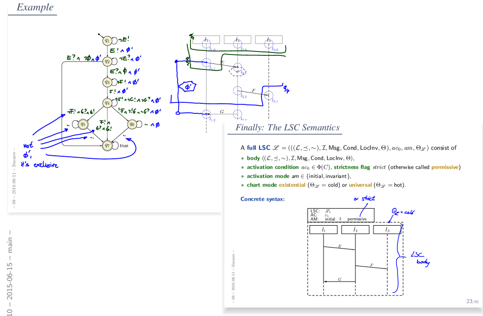
Last Lecture:

- 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



Finally: The LSC Semantics

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

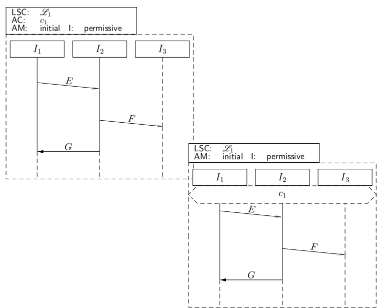
- **body** $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$,
- **activation condition** $ac_0 \in \Phi(C)$, **strictness flag** $strict$ (otherwise called **permissive**)
- **activation mode** $am \in \{\text{initial}, \text{invariant}\}$,
- **chart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

A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$ if and only if

$\Theta_\mathscr{L}$	$am = \text{initial}$	$am = \text{invariant}$
cold	$\exists w \in W \bullet w^0 \models ac \wedge w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists w \in W\ \exists k \in \mathbb{N}_0 \bullet w^k \models ac \wedge w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall w \in W \bullet w^0 \models ac \implies w^0 \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall w \in W\ \forall k \in \mathbb{N}_0 \bullet w^k \models ac \implies w^k \models \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

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

Activation Condition



LSCs vs. Software

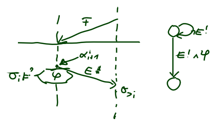
LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.
S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, 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 σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

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



LSCs vs. Software

Let S be a software with $[\![S]\!] = \{\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \mid \cdots\}$.
S is called **compatible** with LSC $\mathscr{L}$ over C and $\mathcal{E}$ is if and only if

- $\Sigma = (C \to \mathbb{B})$, 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 σ_i and α_{i+1} (viewed as a valuation of $E!, E?$):

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

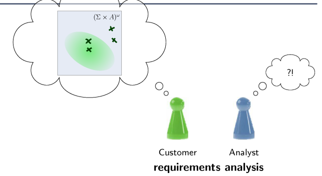
We say S **satisfies** LSC $\mathscr{L}$ (e.g. universal, invariant), denoted by $S \models \mathscr{L}$, if and only if

$$\forall \pi \in [\![S]\!]\ \forall k \in \mathbb{N}_0 \bullet w(\pi)^k \models ac \implies w(\pi)^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w(\pi)/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$$

$\Theta_{\mathscr{L}}$	am = initial	am = invariant
cold	$\exists w \in W \bullet w^0 \models ac \wedge$ $w^0 \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists w \in W\ \exists k \in \mathbb{N}_0 \bullet w^k \models ac \wedge$ $w^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall w \in W \bullet w^0 \models ac \implies$ $w^0 \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall w \in W\ \forall k \in \mathbb{N}_0 \bullet w^k \models ac \implies$ $w^k \models \psi_{\mathsf{hot}}^{\mathsf{Cond}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

Software S satisfies **a set of** LSCs $\mathscr{L}_1, \ldots, \mathscr{L}_n$ if and only if $S \models \mathscr{L}_i$ for all $1 \le i \le n$.

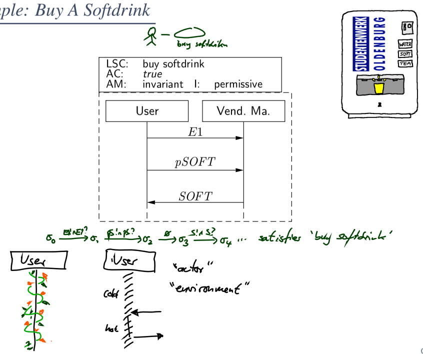
Recall: The Crux of Requirements Engineering



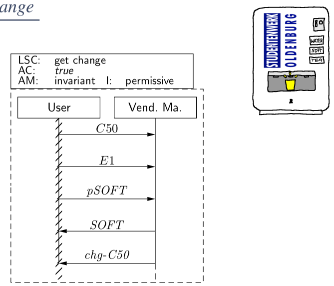
One quite effective approach:
try to **approximate** the requirements with positive and negative **scenarios**.

- 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.

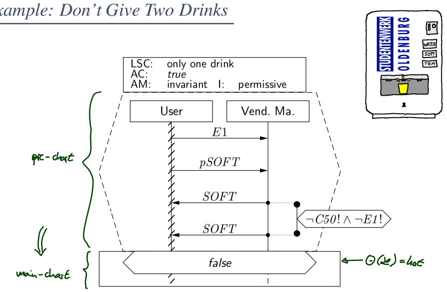
Example: Buy A Softdrink



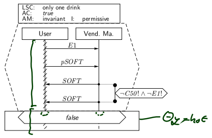
Example: Get Change



Example: Don't Give Two Drinks



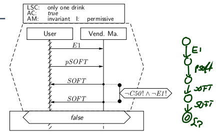
Pre-Charts



A **full LSC** $\mathscr{L} = (PC, MC, ac_0, am, \Theta_{\mathscr{L}})$ **actually** consist of

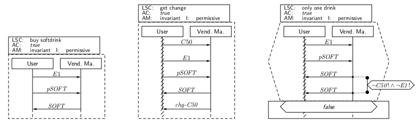
- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathsf{Msg}_P, \mathsf{Cond}_P, \mathsf{LocInv}_P, \Theta_P)$ (possibly empty),
- **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, \mathsf{Msg}_M, \mathsf{Cond}_M, \mathsf{LocInv}_M, \Theta_M)$ (non-empty),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

Pre-Charts Semantics



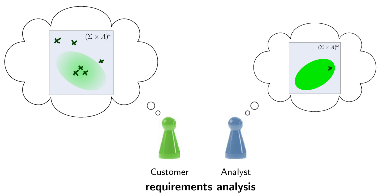
$\Theta_{\mathscr{L}}$	am = initial	am = invariant
cold	$\exists\, w \in W\ \exists\, m \in \mathbb{N}_0 \bullet w^0 \models ac$ $\wedge\ w^0 \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^P)$ $\wedge\ w/1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\ w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^M)$ $\wedge\ w/m+1 \in Lang(\mathcal{B}(MC))$	$\exists\, w \in W\ \exists\, k < m \in \mathbb{N}_0 \bullet w^k \models ac$ $\wedge\ w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^P)$ $\wedge\ w/k+1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\ w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^M)$ $\wedge\ w/m+1 \in Lang(\mathcal{B}(MC))$
hot	$\forall\, w \in W \bullet w^0 \models ac$ $\wedge\ w^0 \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^P)$ $\wedge\ w/1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\ w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0^M)$ $\implies w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0^M)$ $\wedge\ w/m+1 \in Lang(\mathcal{B}(MC))$	$\forall\, w \in W\ \forall\, k \leq m \in \mathbb{N}_0 \bullet w^k \models ac$ $\wedge\ w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0^P)$ $\wedge\ w/k+1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\ w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0^M)$ $\implies w^{m+1} \models \psi^{\mathsf{Cond}}_{\mathsf{cold}}(\emptyset, C_0^M)$ $\wedge\ w/m+1 \in Lang(\mathcal{B}(MC))$

Note: Scenarios and Acceptance Test

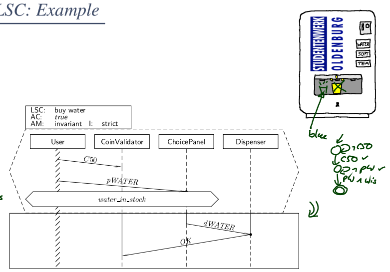


- **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
  (*: as well as (positive) scenarios in general, like use-cases)
- **Universal** LSCs (and negative/anti-scenarios) in general need **exhaustive analysis**!
  (Because they require that the software **never ever** exhibits the unwanted behaviour.)

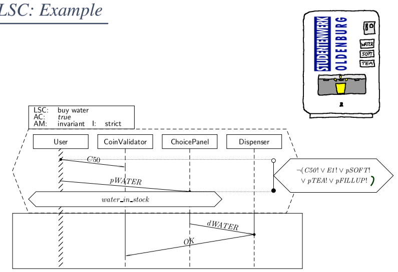
Strengthening Scenarios Into Requirements



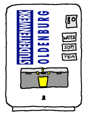
Universal LSC: Example

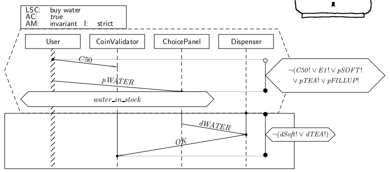


Universal LSC: Example

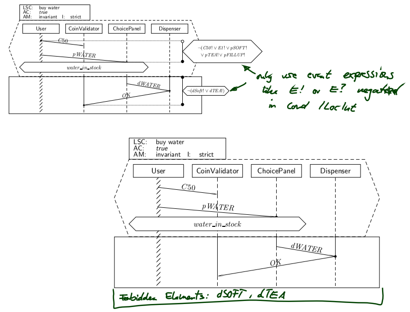


Universal LSC: Example

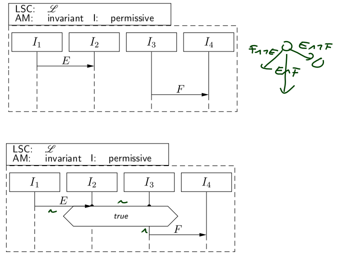




Shortcut: Forbidden Elements



Modelling Idiom: Enforcing Order



Requirements on Requirements Specifications

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 analyseable **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.

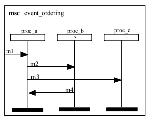
What about LSCs?

LSCs vs. MSCs

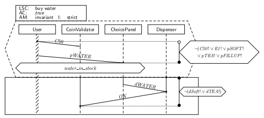
LSCs vs. MSCs

Recall: Most severe **drawbacks** of, e.g., MSCs:

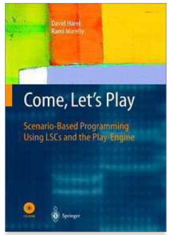
- unclear **interpretation**: example scenario or invariant?
- unclear **activation**: what triggers the requirement?
- unclear **progress** requirement: must all messages be observed?
- **conditions** merely comments
- no means (in language) to express **forbidden scenarios**



(ITU-T, 2011)



Pushing It Even Further



(Harel and Marelly, 2003)

Requirements Engineering Wrap-Up

Recall: Software Specification Example

Alphabet:

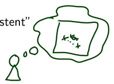
- M – dispense cash only,
- C – return card only,
- $\frac{M}{C}$ – dispense cash and return card.

- **Customer 1** "don't care"

  $\left(M.C \middle| C.M \middle| \frac{M}{C}\right)$

- **Customer 2** "you choose, but be consistent"

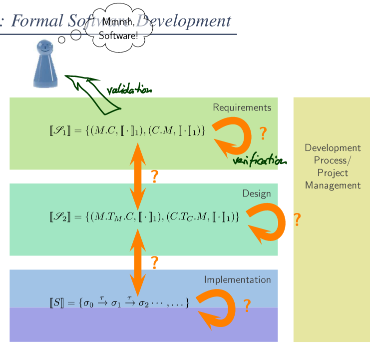
  $(M.C)$ or $(C.M)$



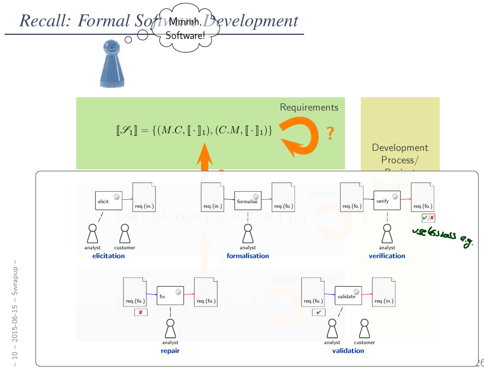
- **Customer 3** "consider human errors"

  $(C.M)$

Recall: Formal Software Development



Recall: Formal Software Development



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 π 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 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, ...

Literature Recommendation

(Rupp and die SOPHISTen, 2014)

References

References

Harel, D. and Marelly, R. (2003). *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag.

ITU-T (2011). *ITU-T Recommendation Z.120: Message Sequence Chart (MSC)*, 5 edition.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rupp, C. and die SOPHISTen (2014). *Requirements-Engineering und -Management*. Hanser, 6th edition.