Given a set of LSCs, which scenario/anti-scenario/requirement is formalised by them?

Given a set of LSCs, give a computation path which is (not) accepted by the LSCs.

Capabilities for following tasks/questions.

- TBA: automata for infinite words
- Recall: TBA Construction and Full LSC

Educational Objectives:

- TBA
- Last Lecture: Recall: TBA Construction and Full LSC

Contents & Goals
LSC: buy water
AC: true
AM: invariant
I: strict

User
CoinValidator
ChoicePanel
Dispenser

C
50
pWATER

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

water in stock

dWATER OK

¬ (dSoft ! ∨ dTEA !)

Requirements on Requirements Specifications

- 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: 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 analysable within LSCs;
- consistency: can formally be analysed!
- neutral/abstract: 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

Recall: Software Specification Example

- Alphabet:
  - M - dispense cash only
  - C - return card only
  - MC - dispense cash and return card.

- Customer 1 “don’t care”

\[ \text{MC} \mathrel{|} \text{MC} \]

- Customer 2 “you choose, but be consistent”

\[ (\text{MC}) \mathrel{|} (\text{CM}) \]

- Customer 3 “consider human errors”

\[ (\text{CM}) \]
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.

- Once again (can it be mentioned too often?): Distinguish domain elements and software elements and (try to) keep them apart to avoid confusion.

---

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 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)