• Introduction
• Definition: Software & SW Specification
• Requirements Specification
• Desired Properties
• Kinds of Requirements
• Analysis Techniques
• Documents
• Dictionary, Specification
• Specification Languages
• Natural Language
• Decision Tables
• Syntax, Semantics
• Completeness, Consistency, ...

Vocabulary
Techniques
informal
semi-formal
formal

• Scenarios
• User Stories, Use Cases
• Live Sequence Charts
• Syntax, Semantics
• Wrap-Up
A Successor Relation on Cuts

The partial order "\(\preceq\)" and the simultaneity relation "\(\sim\)" of locations induce a direct successor relation on cuts of an LSC body as follows:

**Definition.** Let \(C \subseteq L\) be a cut of an LSC body \((L, \preceq, \sim, I, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta)\).

A set \(\emptyset \neq F \subseteq L\) of locations is called a fired-set \(F\) of cut \(C\) if and only if

- \(C \cap F = \emptyset\) and \(C \cup F\) is a cut, i.e., \(F\) is closed under simultaneity,
- all locations in \(F\) are direct \(\preceq\)-successors of the front of \(C\), i.e., \(\forall l \in F \exists l' \in C \cdot l' \preceq l \land (\nexists l'' \in L \cdot l' \preceq l'' \preceq l)\),
- locations in \(F\) that lie on the same instance line are pairwise unordered, i.e., \(\forall l \neq l' \in F \cdot (\exists I \in I \cdot \{l, l'\} \subseteq I) \Rightarrow l \not\preceq l' \land l' \not\preceq l,
- for each asynchronous message reception in \(F\), the corresponding sending is already in \(C\), i.e., \(\forall (l, E, l') \in \text{Msg} \cdot l' \in F \Rightarrow l \in C\).

The cut \(C' = C \cup F\) is called a direct successor of \(C\) via \(F\), denoted by \(C \Rightarrow F\).
From Finite Automata to Symbolic Büchi Automata

**Definition.**
A Symbolic Büchi Automaton (TBA) is a tuple $A = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ where:
- $C_B$ is a set of atomic propositions,
- $Q$ is a finite set of states,
- $q_{ini} \in Q$ is the initial state,
- $\rightarrow \subseteq Q \times \Phi(C_B) \times Q$ is the finite transition relation.
Each transition $(q, \psi, q') \in \rightarrow$ from state $q$ to state $q'$ is labelled with a propositional formula $\psi \in \Phi(C_B)$.
- $Q_F \subseteq Q$ is the set of fair (or accepting) states.

**Example:**
$q_1 q_2 a \wedge b c \vee d B_{sym} : \Sigma = (\{a,b,c,d\} \rightarrow B)$
Definition.
Let $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ be a TBA and $w = \sigma_1, \sigma_2, \sigma_3, \cdots \in (C_B \rightarrow B)$ an infinite word, each letter is a valuation of $C_B$.

An infinite sequence $\pi = q_0, q_1, q_2, \ldots \in Q$ of states is called a run of $B$ over $w$ if and only if
- $q_0 = q_{ini}$,
- for each $i \in \mathbb{N}_0$ there is a transition $(q_i, \psi_i, q_{i+1}) \in \rightarrow$ s.t. $\sigma_i | = \psi_i$.

Example:
$q_1 q_2 a \wedge b c \vee d_B$ sym:
$\Sigma = (\{a, b, c, d\} \rightarrow B)$
\[
\begin{align*}
\{a, b\} & \quad \text{for short,} \\
\{c\} & \\
\{a, b\} & \\
(\{d\}, \{a, b\}) &
\end{align*}
\]

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 $\pi = (q_i)_{i \in \mathbb{N}_0}$ over $w$ such that fair (or accepting) states are visited infinitely often by $\pi$, i.e.,
\[\forall i \in \mathbb{N}_0 \exists j > i: q_j \in Q_F.\]

We call the set $\text{Lang}(B) \subseteq (C_B \rightarrow B)^\omega$ of words that are accepted by $B$ the language of $B$. 

Example:
$q_1 q_2 a \wedge b c \vee d_B$ sym:
$\Sigma = (\{a, b, c, d\} \rightarrow B)$

LSCs vs. Software
Requirements Engineering Wrap-Up.

\[ \text{where} \]

\[ \text{induces} \]

\[ \text{states} \]

\[ \text{if and only if} \]

\[ \text{LSCs as Software Specification} \]
A set of software satisfies LSCs at work.

(i) Check water dispenser.

(ii) Check softdrink dispenser.

(iii) Check tea dispenser.

(iv) Get 50 cent change.

We say software performs a self-test on power-on.

Example: vending machine.

User: Perform self-test on power-on

Vend. Mach.: initial $I$:

true

Vend. Mach.: invariant $I$:

permissive

AM: invariant $I$:

true

AM: invariant $I$:

permissive

LSC: power-on self test $AC$:

true

LSC: buy softdrink $AC$:

true

C$\in\{\text{SOFT}, \text{TEA}\}$

User: Insert one 50 cent and one 1 euro coin.

User: Press the 'softdrink' button.

User: Get a softdrink.

User: Get change.

C$\in\{\text{SOFT}, \text{TEA}\}$

Positive scenario

• Example: vending machine

User: Insert one 50 cent and one 1 euro coin.

User: Press the 'softdrink' button.

User: Get a softdrink.

User: Get 50 cent change.

C$\in\{\text{SOFT}, \text{TEA}\}$

Positive scenario

• Example: vending machine

User: Insert one 50 cent and one 1 euro coin.

User: Press the 'softdrink' button.

User: Get a softdrink.

User: Get 50 cent change.
(i) Insert one 1 euro coin.

(ii) and press the 'water' button

(iii) Do not insert any more money.

Accept the software if we get water

Then

The dispenser gives cold water.
Live Sequence Charts

TBA Construction

LSCs vs. Software

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

LSCs in Requirements Analysis

LSCs vs. Quality Assurance

Requirements Engineering Wrap-Up

Requirements Analysis in a Nutshell

Recall: Validation by Translation
How to Prove that a Software Satisfies an LSC?

How to Prove that a Software Satisfies an LSC?

How to Prove that a Software Satisfies an LSC?

How to Prove that a Software Satisfies an LSC?

How to Prove that a Software Satisfies an LSC?

How to Prove that a Software Satisfies an LSC?
If in doubt, use formal notations to communicate with your developers. Ask for each requirements: what is the acceptance test (as safety precaution, e.g., in lawsuits).

Two broad directions: teach formalism (formal) to your developers 

- Option 1: serve as (Strong) Literature Recommendation
  - (Course’s Manifesto?)
  - compare expected outcome and real outcome.
  - (formalising)
  - FM expert
  - /c07
  - evaluates
  - system scenario to valuation translates FM expert
  - /c05
  - translates system scenario to valuation
  - /c04
  - /c01
  - may be valid
  - scenario: may be valid
  - usual: invariantI: strict
  - true
  - LSC: buy water
  - AC:
  - /c03
  - /c02
  - /c01
  - stock
  - in
  - water
  - AM: invariant
  - true
  - /c01
  - /c04
  - /c05
  - /c01
  - stop ventilation
  - /c01
  - stop
  - water
  - × button pressed?
  - b
  - 50
  - CoinValidator
  - Dispenser
  - /c01
  - ×
  - ×
  - /c01
  - /c01
  - else
  - r
  - T
  - AM: invariantI: strict
  - true
  - /c01
  - /c04
  - /c05
  - /c01
  - stop ventilation
  - /c01
  - stop
  - water
  - × button pressed?
  - b
  - 50
  - CoinValidator
  - Dispenser
  - /c01
  - ×
  - ×
  - /c01
  - /c01
  - else
  - r
  - T
  - AM: invariantI: strict
  - true
  - /c01
  - /c04
  - /c05
  - /c01
  - stop ventilation
  - /c01
  - stop
  - water
  - × button pressed?
  - b
  - 50
  - CoinValidator
  - Dispenser
  - /c01
  - ×
  - ×
  - /c01
  - /c01
  - else
  - r
  - T
  - AM: invariantI: strict
  - true
  - /c01
  - /c04
  - /c05
  - /c01
  - stop ventilation
  - /c01
  - stop
  - water
  - × button pressed?
  - b
  - 50
  - CoinValidator
  - Dispenser
  - /c01
  - ×
  - ×
  - /c01
  - /c01
  - else
  - r
  - T

For a given task, use the diagrams with text

References

- User Stories, Use Cases
- Live Sequence Charts
- Assumptions
- Syntax, Semantics
- Syndrome
- Argumentation
- User, System, Environment, ...
*Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. 
Springer-Verlag.

*Software Engineering*. 

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