

*Software Design, Modelling and Analysis in UML*

*Lecture 21: Model-based Software  
Development*

*2017-02-06*

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

Albert-Ludwigs-Universität Freiburg, Germany

- **Live Sequence Charts**

- **Semantics**

- **Full LSCs**

- **Existential and Universal**

- **Pre-Charts**

- **Forbidden Scenarios**

- **LSCs and Tests**

- **Model-Based/-Driven Software Engineering**

- **Model Element Coverage** of test cases

- **Model-based Testing**

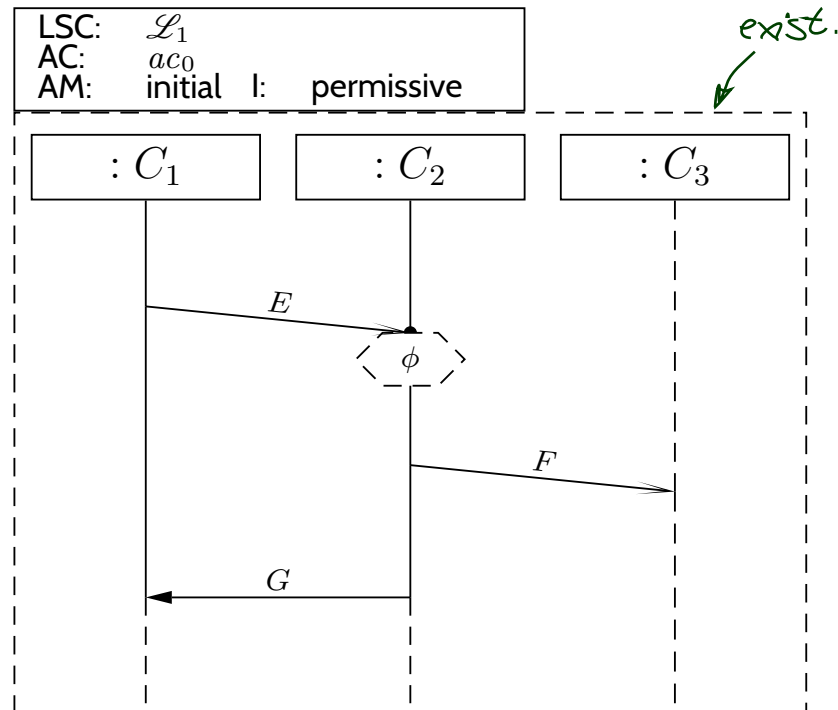
# *Live Sequence Charts — Full LSC Semantics*

# Full LSCs

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

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

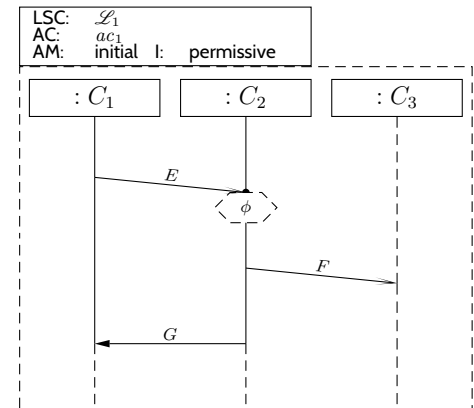
Concrete syntax:



# Full LSCs

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

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



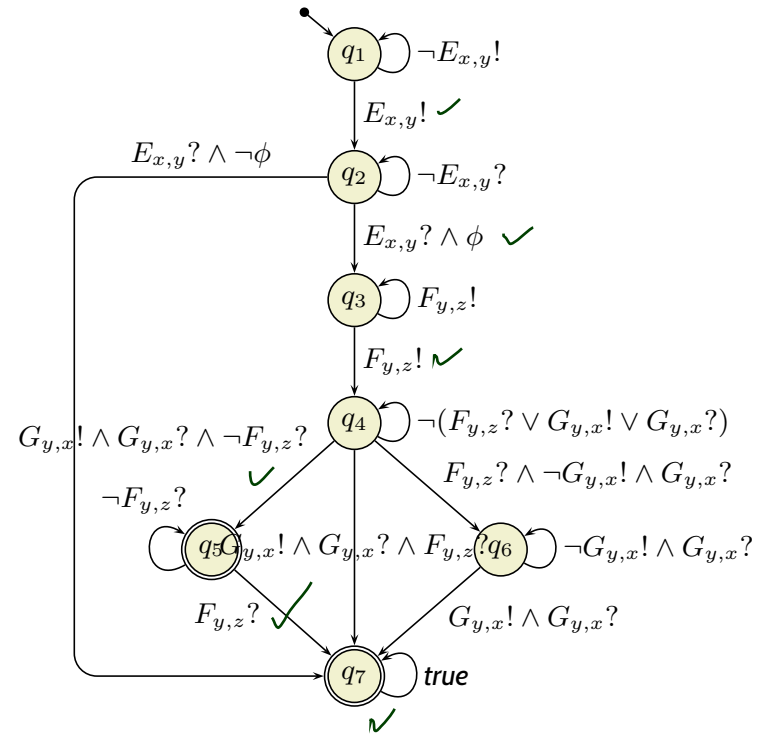
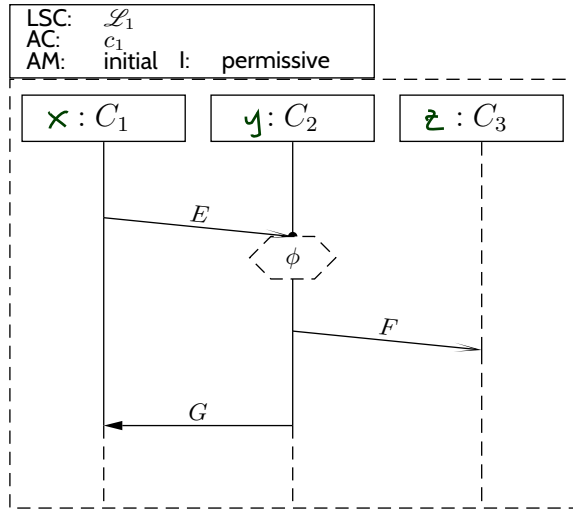
A **set of words**  $W \subseteq (\text{Expr}_{\mathcal{B}} \rightarrow \mathbb{B})^\omega$  is **accepted** by  $\mathcal{L}$  if and only if

suffix of  $w$  starting at index 1

$\Theta_{\mathcal{L}}$	$am = \text{initial}$	$am = \text{invariant}$
<b>cold</b>	$\exists \beta \exists w \in W \bullet w^0 \models_{\beta} ac \wedge \neg \psi_{\text{exit}}(C_0)$ $\wedge w^0 \models_{\beta} \psi_{\text{prog}}(\emptyset, C_0) \wedge w/1 \in \mathcal{L}_{\beta}(\mathcal{B}(\mathcal{L}))$	$\exists \beta \exists w \in W \exists k \in \mathbb{N}_0 \bullet w^k \models_{\beta} ac \wedge \neg \psi_{\text{exit}}(C_0)$ $\wedge w^k \models_{\beta} \psi_{\text{prog}}(\emptyset, C_0) \wedge w/k + 1 \in \mathcal{L}_{\beta}(\mathcal{B}(\mathcal{L}))$
<b>hot</b>	$\forall \beta \forall w \in W \bullet w^0 \models_{\beta} ac \wedge \neg \psi_{\text{exit}}(C_0)$ $\implies w^0 \models_{\beta} \psi_{\text{prog}}(\emptyset, C_0) \wedge w/1 \in \mathcal{L}_{\beta}(\mathcal{B}(\mathcal{L}))$	$\forall \beta \forall w \in W \forall k \in \mathbb{N}_0 \bullet w^k \models_{\beta} ac \wedge \neg \psi_{\text{exit}}(C_0)$ $\implies w^k \models_{\beta} \psi_{\text{hot}}^{\text{Cond}}(\emptyset, C_0) \wedge w/k + 1 \in \mathcal{L}_{\beta}(\mathcal{B}(\mathcal{L}))$

where  $C_0$  is the minimal (or **instance heads**) cut.

# Full LSC Semantics: Example

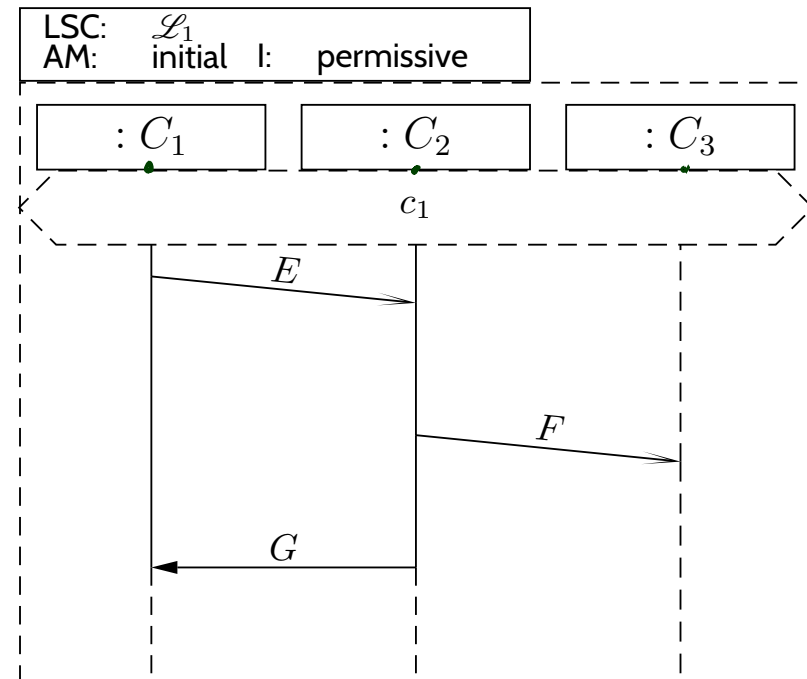
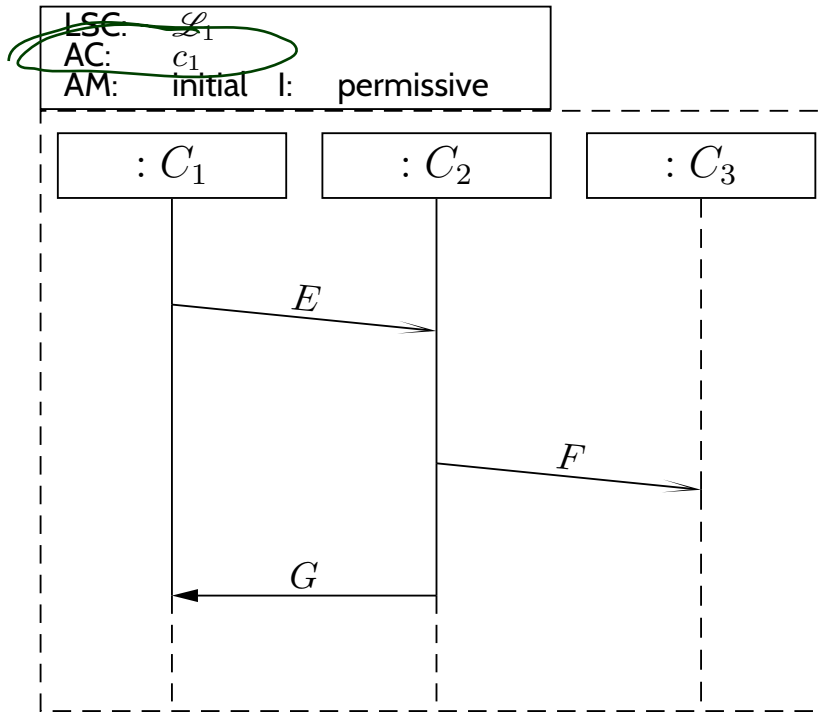


$$\begin{aligned}
 w: & (\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} \dots \rightarrow (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \xrightarrow[c_1]{(cons_1, \{(:E, c_2)\})} (\sigma_2, \varepsilon_2) \xrightarrow[c_2]{(\{(:E)\}, Snd_2)} \dots \\
 & (\sigma_3, \varepsilon_3) \xrightarrow[c_2]{(cons_3, \{(:F, c_3)\})} (\sigma_4, \varepsilon_4) \xrightarrow[c_2]{(cons_4, \{(:G(), c_1)\})} (\sigma_5, \varepsilon_5) \xrightarrow[c_3]{(\{(:F)\}, Snd_5)} (\sigma_6, \varepsilon_6) \rightarrow \dots
 \end{aligned}$$

$$\beta = \{x \mapsto c_1, y \mapsto c_2, z \mapsto c_3\}$$

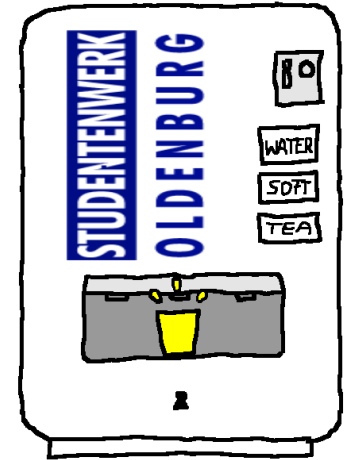
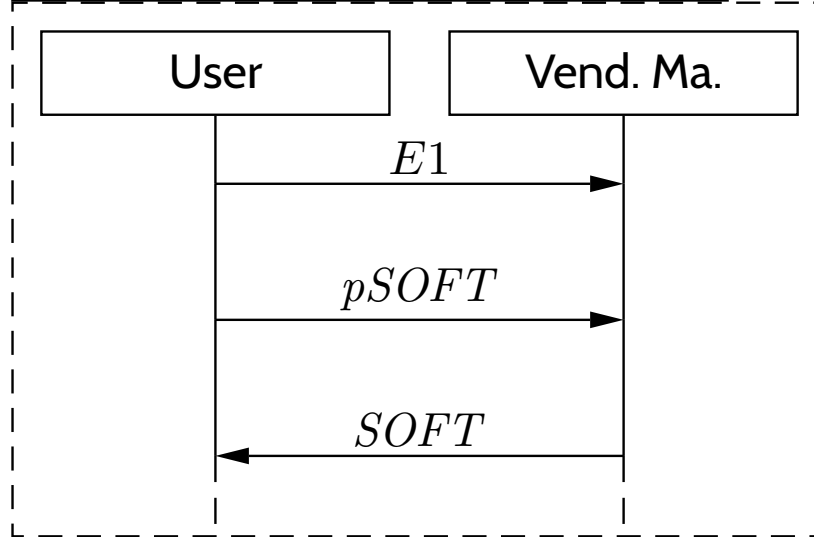
$\hookrightarrow w$  is accepted by  $\mathcal{L}_1$

# Note: Activation Condition



# Existential LSC Example: Buy A Softdrink

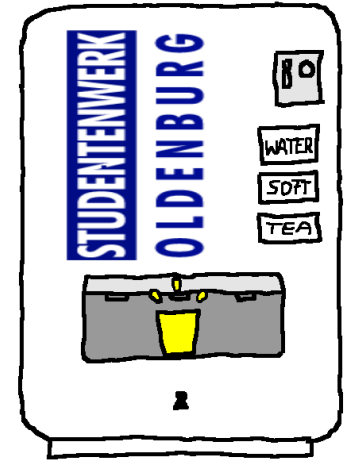
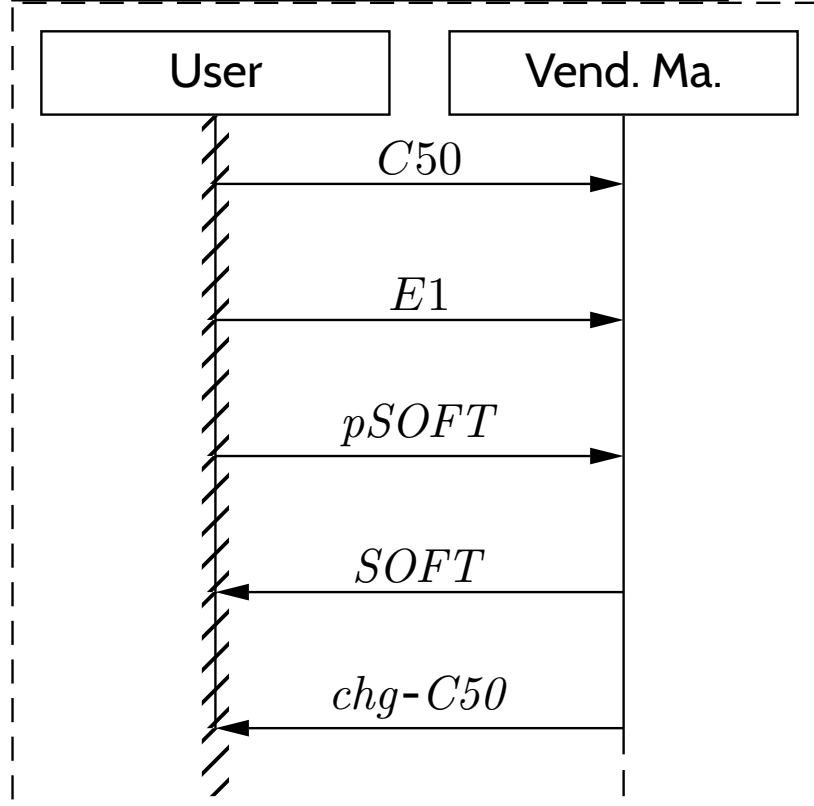
LSC: buy softdrink  
AC: true  
AM: invariant I: permissive





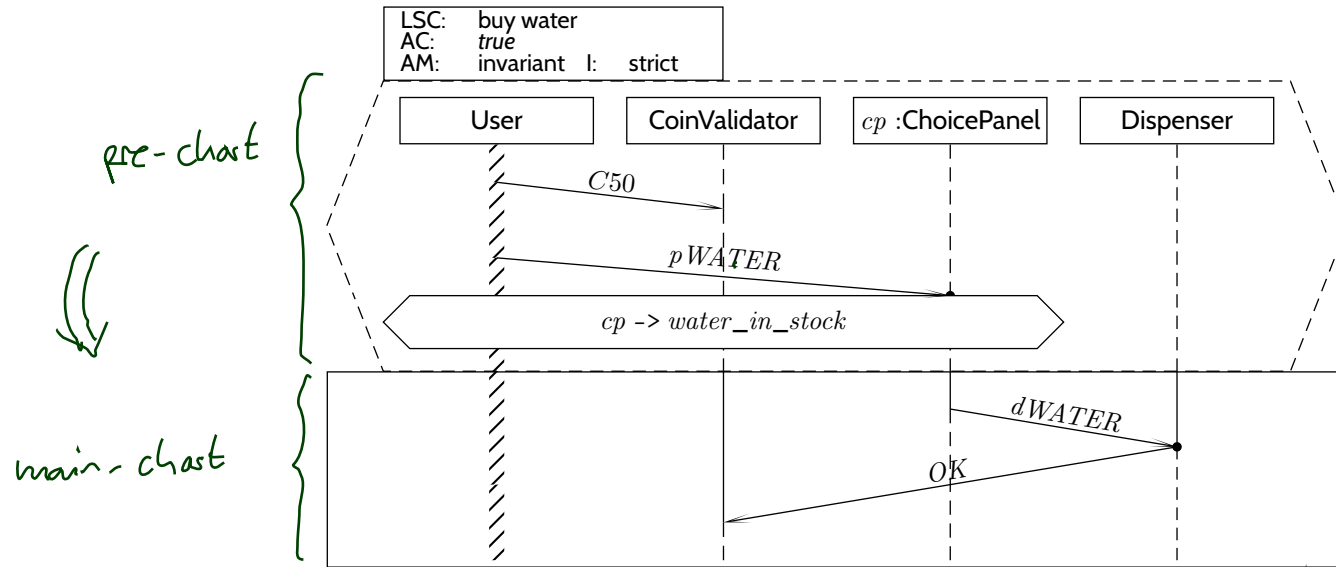
# Existential LSC Example: Get Change

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



# *Live Sequence Charts — Precharts*

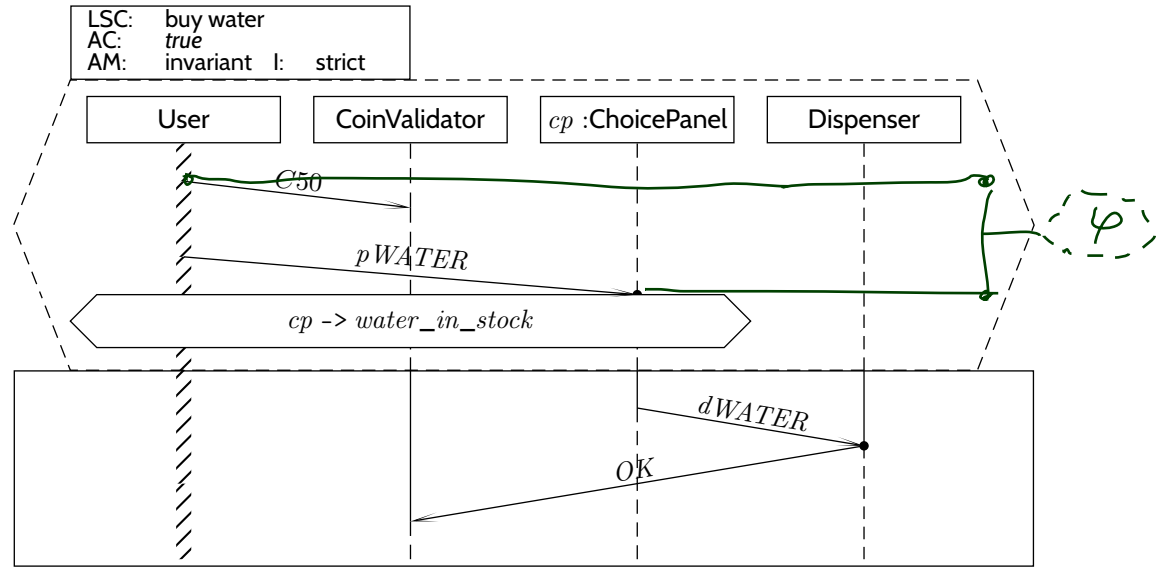
# Pre-Charts



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

- **pre-chart**  $PC = ((L_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathcal{S}, \text{Msg}_P, \text{Cond}_P, \text{LocInv}_P, \Theta_P)$  (possibly empty),
- **main-chart**  $MC = ((L_M, \preceq_M, \sim_M), \mathcal{I}_M, \mathcal{S}, \text{Msg}_M, \text{Cond}_M, \text{LocInv}_M, \Theta_M)$  (non-empty),
- **activation condition**  $ac_0 : \text{Bool} \in \text{Expr}_{\mathcal{S}}$ ,
- **strictness flag** *strict* (otherwise called **permissive**)
- **activation mode**  $am \in \{\text{initial}, \text{invariant}\}$ ,
- **chart mode** **existential** ( $\Theta_{\mathcal{L}} = \text{cold}$ ) or **universal** ( $\Theta_{\mathcal{L}} = \text{hot}$ ).

# Pre-Charts Semantics



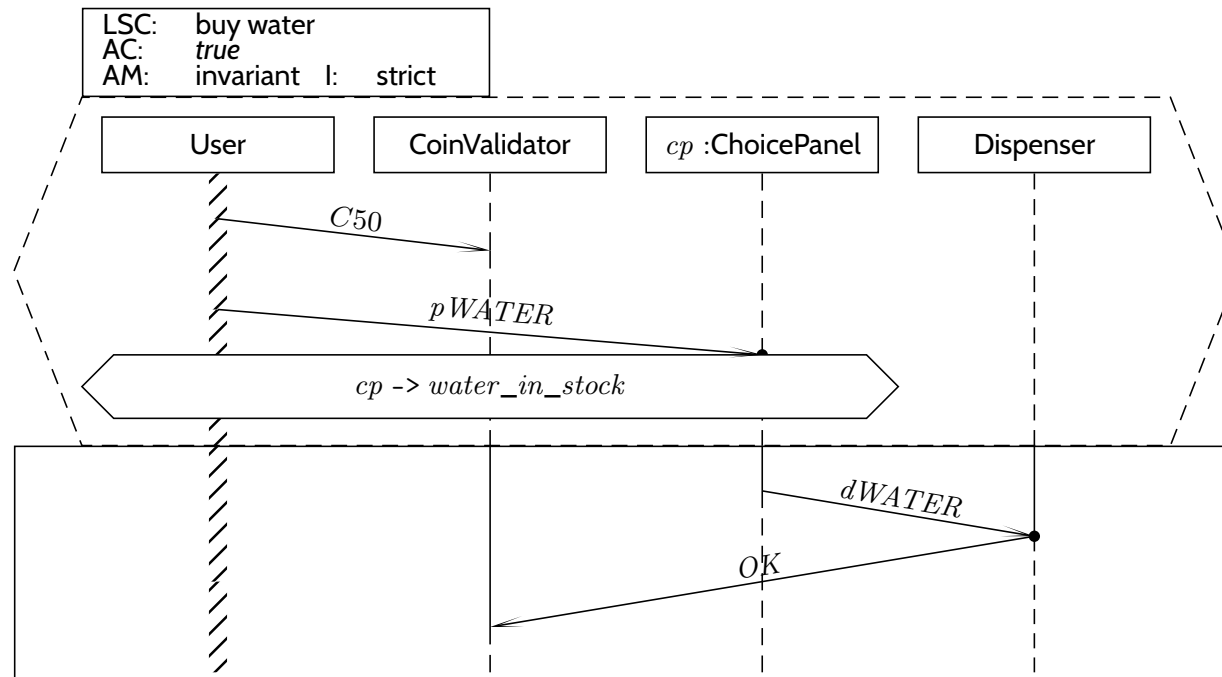
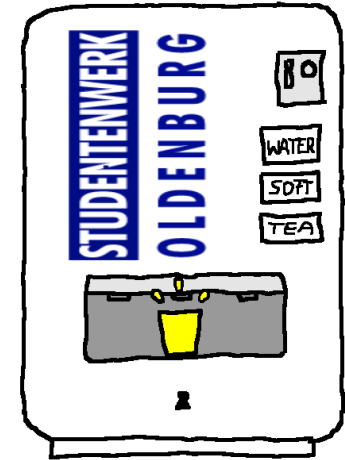
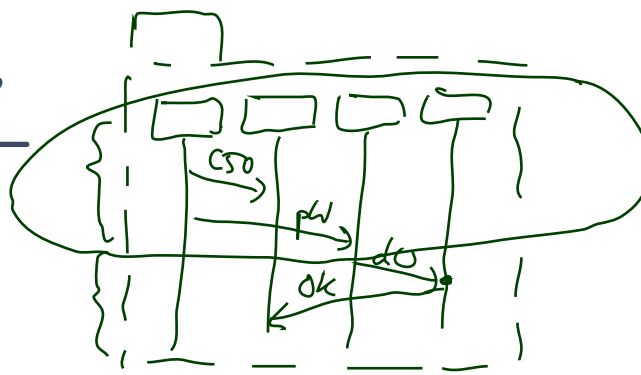
$$w = w_0 \ w_1 \ \dots \ w_m \ w_{m+1} \ w_{m+2} \ w_{m+3} \ \dots$$

$\Pi_{ac} \ \underbrace{\quad}_{\in \mathcal{L}(\mathcal{B}(PC))} \Rightarrow \ \underbrace{\quad}_{\in \mathcal{L}(\mathcal{B}(MC))}$

(\*) usually: reaching state of maximal cut of pre-chart

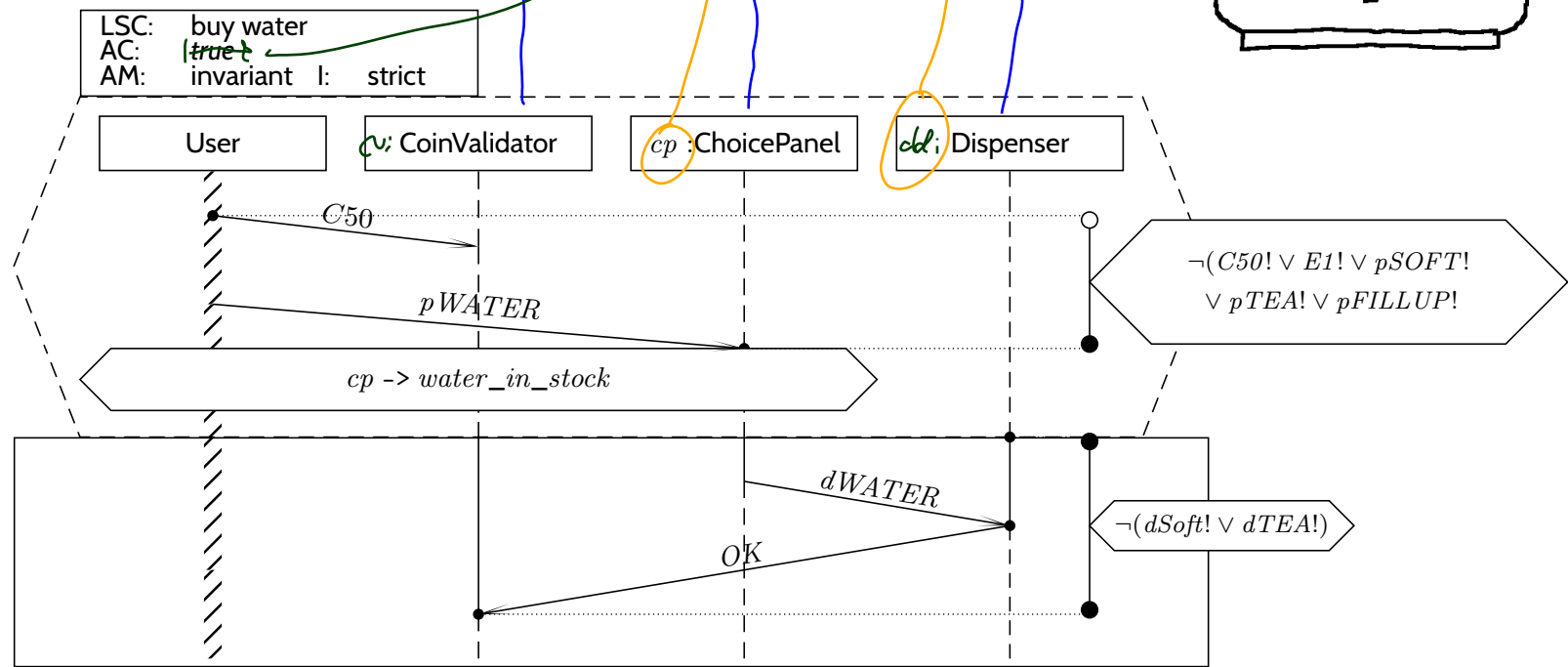
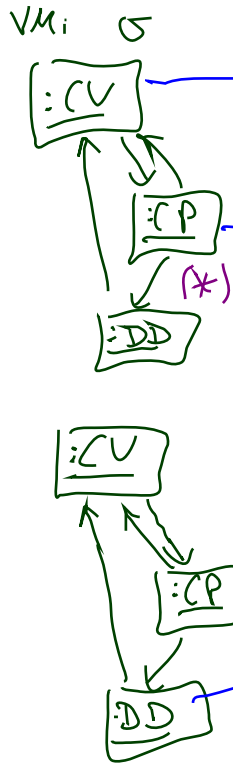
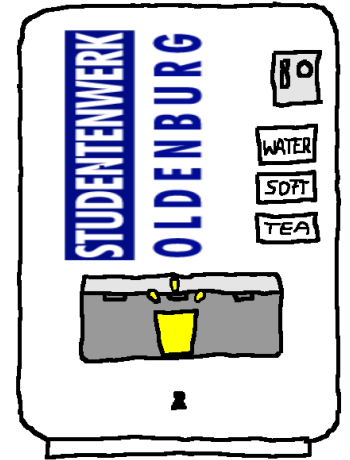
	$am = \text{initial}$	$am = \text{invariant}$
$\ominus_{\mathcal{L}} = \text{cold}$	$\exists \beta \exists w \in W \exists m \in \mathbb{N}_0 \bullet$ $\wedge w^0 \models_{\beta} ac \wedge \neg \psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge w^1, \dots, w^m \in \mathcal{L}_{\beta}(\mathcal{B}(PC))$ $\wedge w^{m+1} \models_{\beta} \neg \psi_{exit}(C_0^M)$ $\wedge w^{m+1} \models_{\beta} \psi_{prog}(\emptyset, C_0^M)$ $\wedge w/m + 2 \in \mathcal{L}_{\beta}(\mathcal{B}(MC))$	$\exists \beta \exists w \in W \exists k < m \in \mathbb{N}_0 \bullet$ $\wedge w^k \models_{\beta} ac \wedge \neg \psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge w^{k+1}, \dots, w^m \in \mathcal{L}_{\beta}(\mathcal{B}(PC))$ $\wedge w^{m+1} \models_{\beta} \neg \psi_{exit}(C_0^M)$ $\wedge w^{m+1} \models_{\beta} \psi_{prog}(\emptyset, C_0^M)$ $\wedge w/m + 2 \in \mathcal{L}_{\beta}(\mathcal{B}(MC))$
$\ominus_{\mathcal{L}} = \text{hot}$	$\forall \beta \forall w \in W \forall m \in \mathbb{N}_0 \bullet$ $\left\{ \begin{array}{l} \wedge w^0 \models_{\beta} ac \wedge \neg \psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P) \\ \wedge w^1, \dots, w^m \in \mathcal{L}_{\beta}(\mathcal{B}(PC)) \quad (*) \\ \wedge w^{m+1} \models_{\beta} \neg \psi_{exit}(C_0^M) \end{array} \right.$ $\Rightarrow \left\{ \begin{array}{l} w^{m+1} \models_{\beta} \psi_{prog}(\emptyset, C_0^M) \\ \wedge w/m + 2 \in \mathcal{L}_{\beta}(\mathcal{B}(MC)) \end{array} \right.$	$\forall \beta \forall w \in W \forall k \leq m \in \mathbb{N}_0 \bullet$ $\wedge w^k \models_{\beta} ac \wedge \neg \psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge w^{k+1}, \dots, w^m \in \mathcal{L}_{\beta}(\mathcal{B}(PC))$ $\wedge w^{m+1} \models_{\beta} \neg \psi_{exit}(C_0^M)$ $\Rightarrow \left\{ \begin{array}{l} w^{m+1} \models_{\beta} \psi_{prog}(\emptyset, C_0^M) \\ \wedge w/m + 2 \in \mathcal{L}_{\beta}(\mathcal{B}(MC)) \end{array} \right.$

# Universal LSC: Example

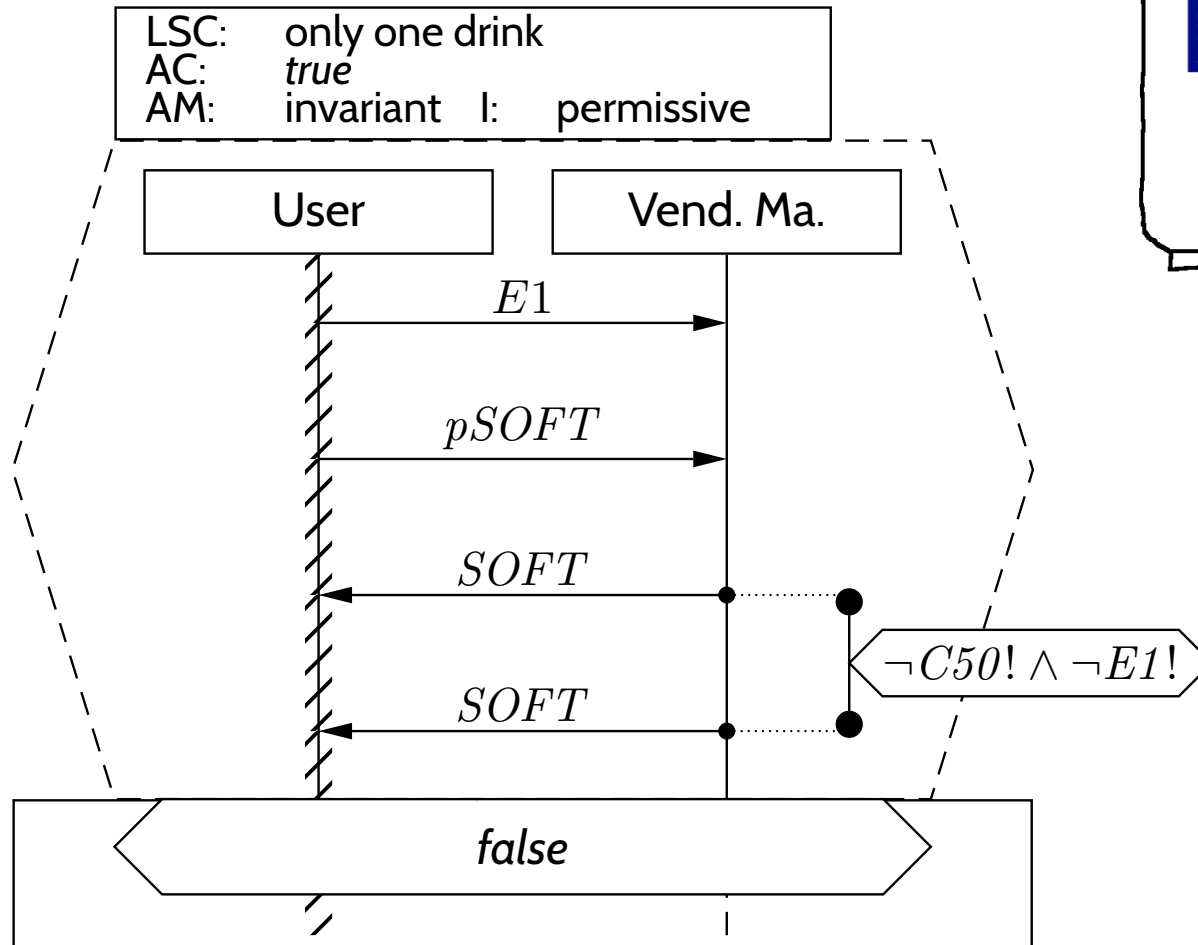
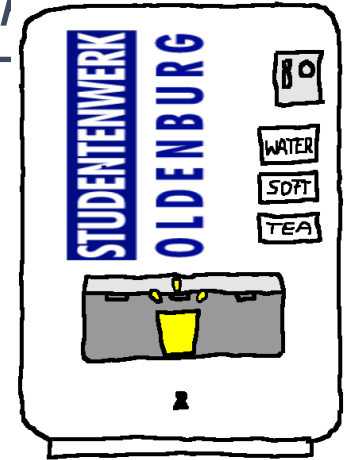


# Universal LSC: Example

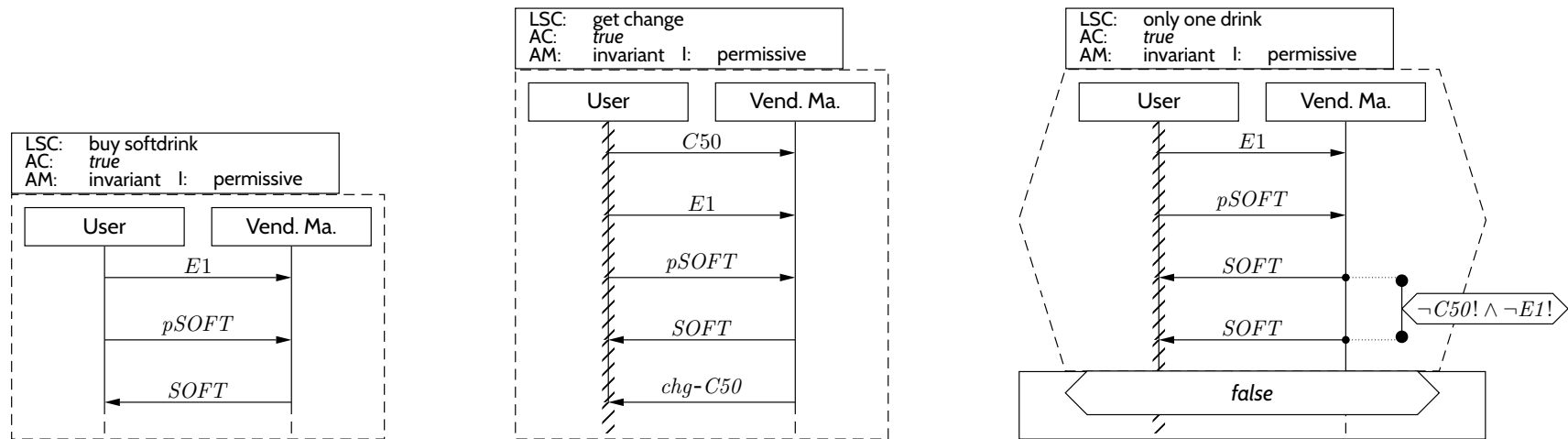
$cv.its\ CP = cp$   
 $\wedge cp.its\ CV = cv$   
 $\wedge dd.its\ CV = cv$   
 $\wedge cp.its\ DD = dd$



# Forbidden Scenario Example: Don't Give Two Drinks



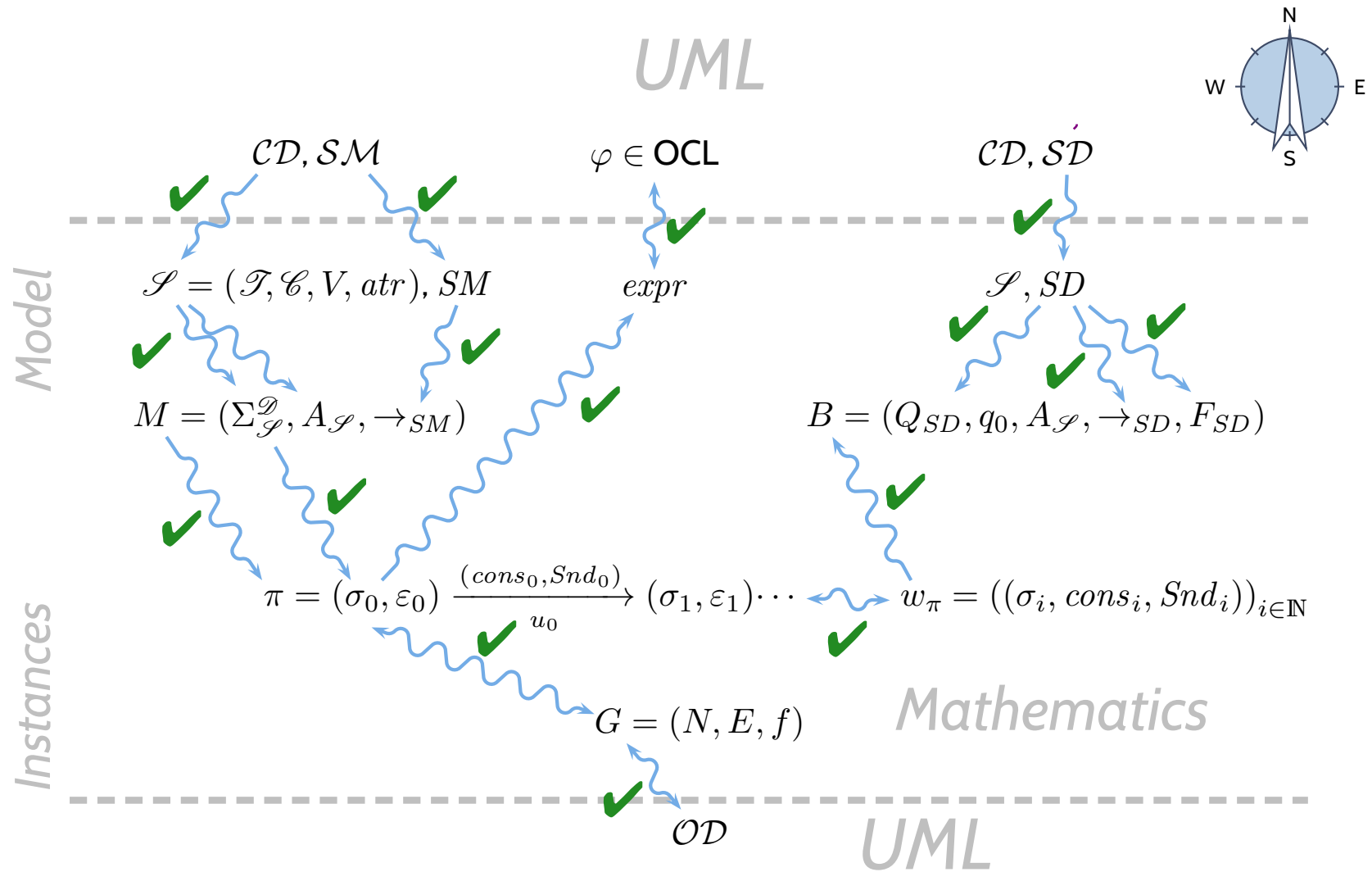
# Note: Sequence Diagrams and (Acceptance) Test



- **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.)

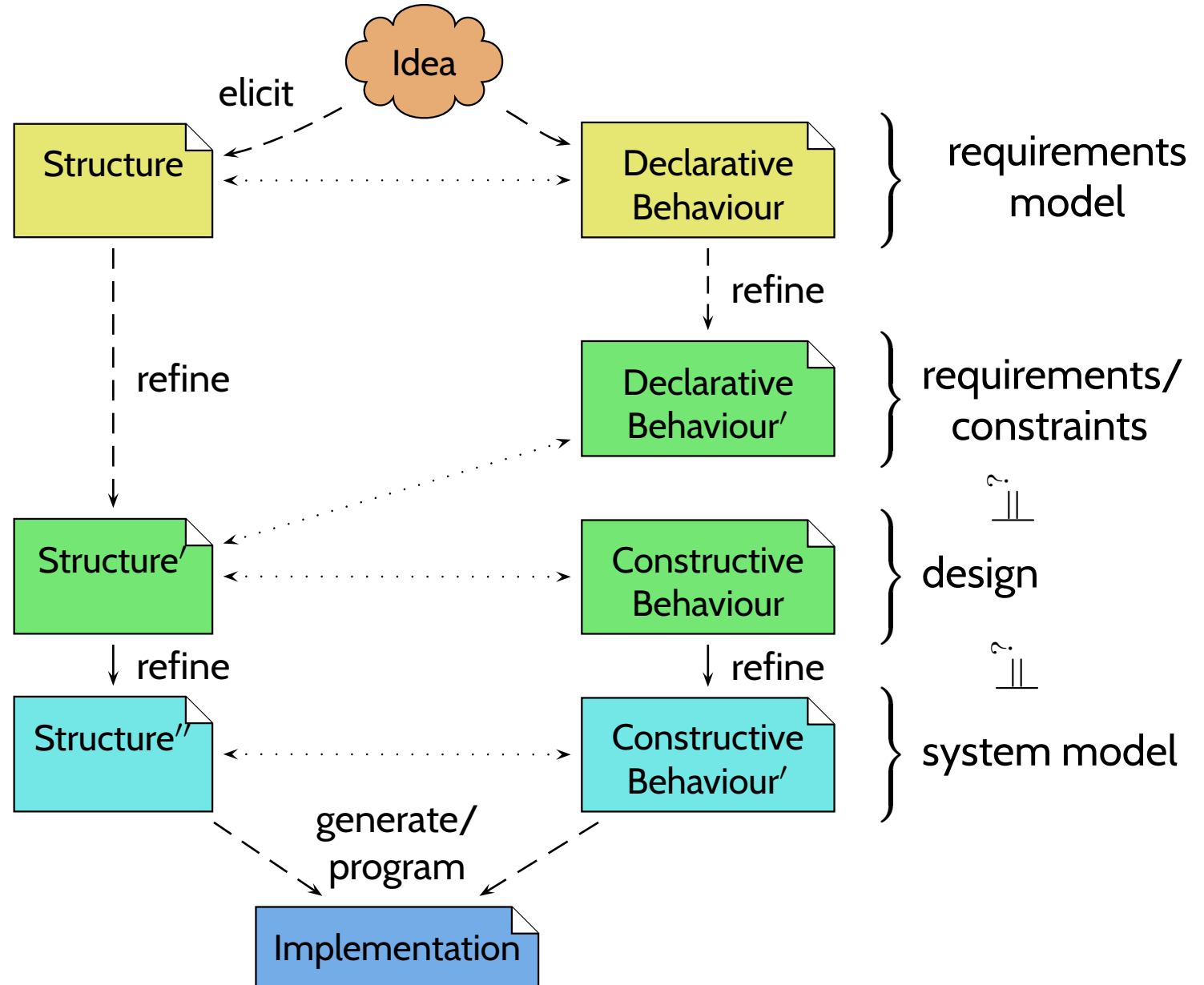


# Course Map

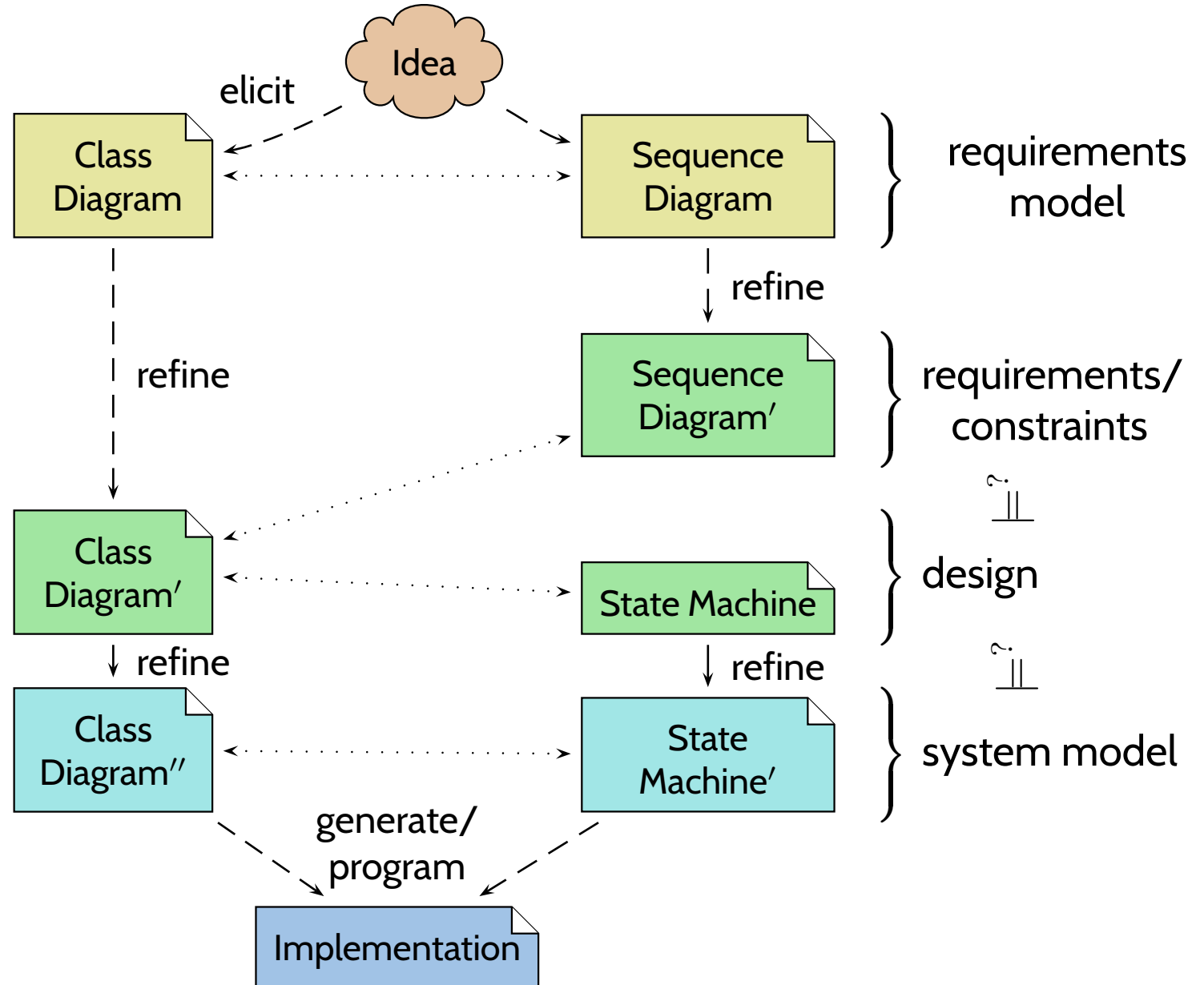


# *Model-Based/-Driven Software Engineering*

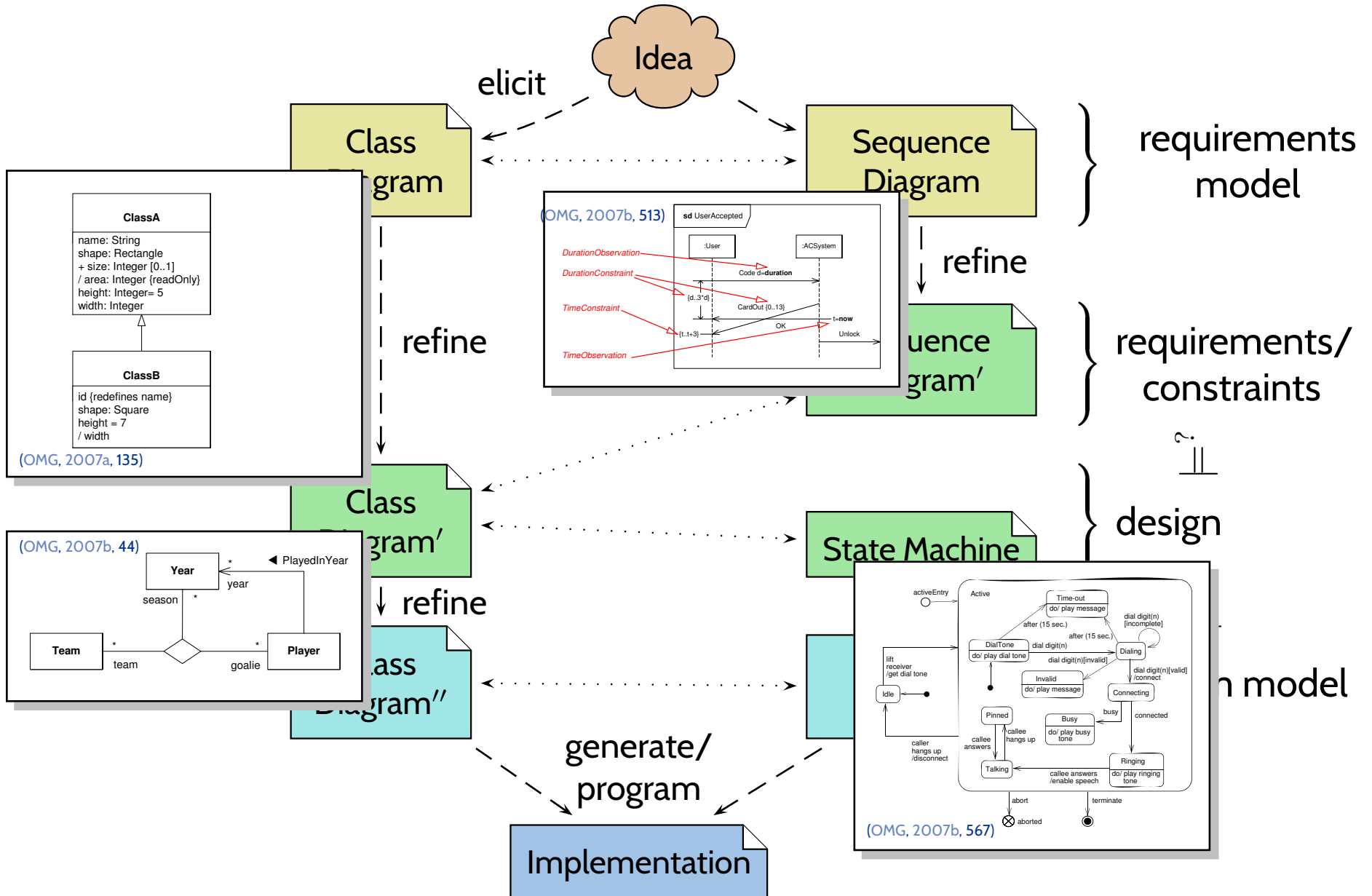
# Model-Driven Software Engineering



# Model-Driven Software Engineering with UML



# Model-Driven Software Engineering with UML



# *Model-Based Testing*

## Recall: Test Case

**Definition.** A **test case**  $T$  is a pair  $(In, Soll)$  consisting of

- a description  $In$  of sets of finite **input sequences**,
  - a description  $Soll$  of **expected outcomes**,
- and an interpretation  $\llbracket \cdot \rrbracket$  of these descriptions.

A **test execution**  $\pi$ , i.e.  $((\pi^0, \dots, \pi^n) \downarrow \Sigma_{in}) \in In$  for some  $n \in \mathbb{N}_0$ , is called

- **successful** (or **positive**)

if it discovered an error,  
i.e., if  $\pi \notin \llbracket Soll \rrbracket$ .

(Alternative: test item  $S$  **failed to pass test**; confusing: “test failed”.)

- **unsuccessful** (or **negative**)

if it did not discover an error,  
i.e., if  $\pi \in \llbracket Soll \rrbracket$ .

(Alternative: test item  $S$  **passed test**; okay: “test passed”.)

# Glass-Box Testing: Coverage

- **Coverage** is a property of **test cases** and **test suites**.
- Execution  $\pi$  of test case  $T$  achieves  $p\%$  **statement coverage** if and only if

$$p = cov_{stm}(\pi) := \frac{|\bigcup_{i \in \mathbb{N}_0} stm(\sigma_i)|}{|Stm_S|}, |Stm_S| \neq 0.$$

Test case  $T$  achieves  $p\%$  **statement coverage** if and only if  $p = \min_{\pi \text{ execution of } T} cov_{stm}(\pi)$ .

- Execution  $\pi$  of  $T$  achieves  $p\%$  **branch coverage** if and only if

$$p = cov_{cnd}(\pi) := \frac{|\bigcup_{i \in \mathbb{N}_0} cnd(\sigma_i)|}{|Cnd_S|}, |Cnd_S| \neq 0.$$

Test case  $T$  achieves  $p\%$  **branch coverage** if and only if  $p = \min_{\pi \text{ execution of } T} cov_{cnd}(\pi)$ .

- **Define:**  $p = 100$  for empty program.
- Statement/branch coverage canonically extends to test suite  $\mathcal{T} = \{T_1, \dots, T_n\}$ , e.g. given executions  $\pi_1, \dots, \pi_n$ ,  $\mathcal{T}$  achieves

$$p = \frac{|\bigcup_{1 \leq j \leq n} \bigcup_{i \in \mathbb{N}_0} stm(\pi_j^i)|}{|Stm_S|}, |Stm_S| \neq 0, \text{ **statement coverage** .}$$

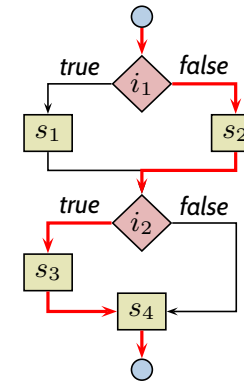


# Coverage Example

```

int f(int x, int y, int z)
{
  i1: if (x > 100 ∧ y > 10)
  s1:   z = z * 2;
      else
  s2:   z = z/2;
  i2:   if (x > 500 ∨ y > 50)
  s3:   z = z * 5;
  s4:   ;
}

```

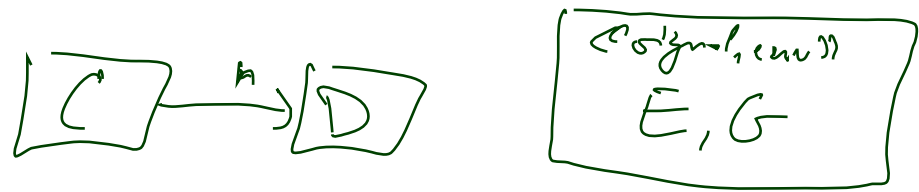


- **Requirement:**  $\{true\} f \{true\}$  (no abnormal termination), i.e.  $Soll = \Sigma^* \cup \Sigma^\omega$ .

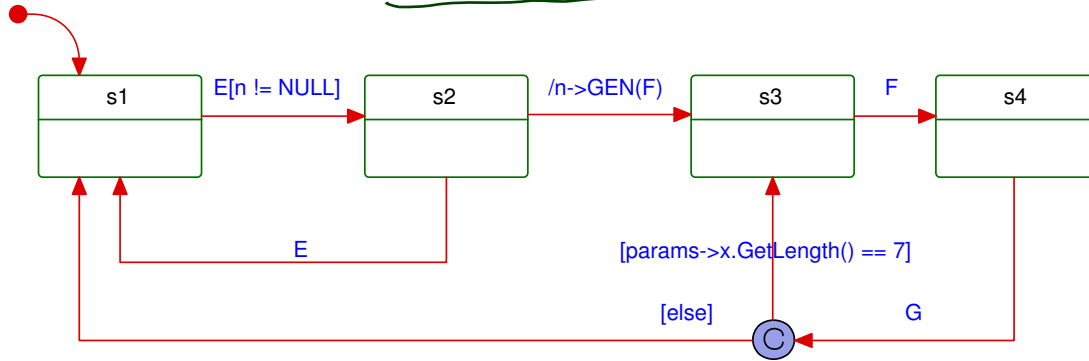
test suite coverage

<i>In</i>											%	%	<i>i2</i> /%
<i>x, y, z</i>	<i>i1</i> /t	<i>i1</i> /f	<i>s1</i>	<i>s2</i>	<i>i2</i> /t	<i>i2</i> /f	<i>c1</i>	<i>c2</i>	<i>s3</i>	<i>s4</i>	stm	cnd	term
501, 11, 0	✓		✓		✓		✓		✓	✓	75	50	25
501, 0, 0		✓		✓	✓		✓		✓	✓	100	75	25
0, 0, 0		✓		✓		✓				✓	100	100	75
0, 51, 0		✓		✓	✓			✓		✓	100	100	100

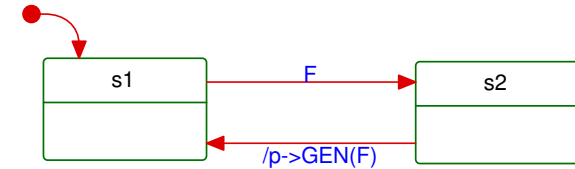
# Model-Element Coverage



State machine of *C*:



State machine of *D*:



## 100 % Element coverage of *C*'s state machine:

- a set of test cases (e.g. **Sequence Diagrams**) such that
- when conducting these test cases
  - **each state** of *C* is **reached at least once**,
  - **each transition** of *C* is **taken at least once**.

(state coverage)

(transition coverage)

## In general: **State coverage of a set of test cases**

- number-of-states reached / number-of-states in state machine.

## *Excursion: Automatic Test Generation*

# Model-based Testing

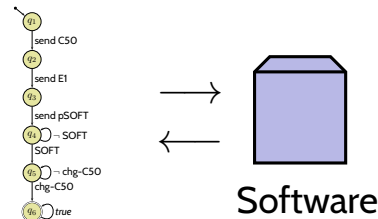
---

- Given a **set of test cases** passing for the model,
- and an **implementation of the model** (maybe hand-written).
- **Execute the test cases on the implementation** (or the final system).

This may need an appropriate **interpretation**. For example, if the test case says

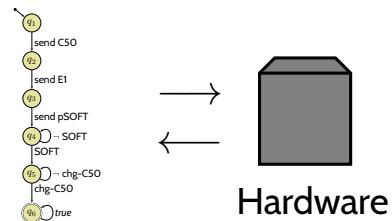
- send “C50” to the CoinValidator,
- rather insert a 50 Cent coin into the vending machine.
- If the vending machine **does not behave** according to the test,
  - then **there’s something wrong** (wrong test conduction, wrong implementation, etc.).
- If the vending machine **does behave** according to the test,
  - then we know that **this scenario works** – not more.

# Vocabulary



- **Software-in-the-loop:**

The final implementation is examined using a separate computer to simulate other system components.



- **Hardware-in-the-loop:**

The final implementation is running on (prototype) hardware which is connected by its standard input/output interface (e.g. CAN-bus) to a separate computer which simulates other system components.

# *References*

# References

---

Dobing, B. and Parsons, J. (2006). How UML is used. Communications of the ACM, 49(5):109–114.

OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.