Systeme hoher Qualität und Sicherheit
Universität Bremen WS 2015/2016

Lecture 06 (16-11-2015)

Formal Modelling with SysML and OCL

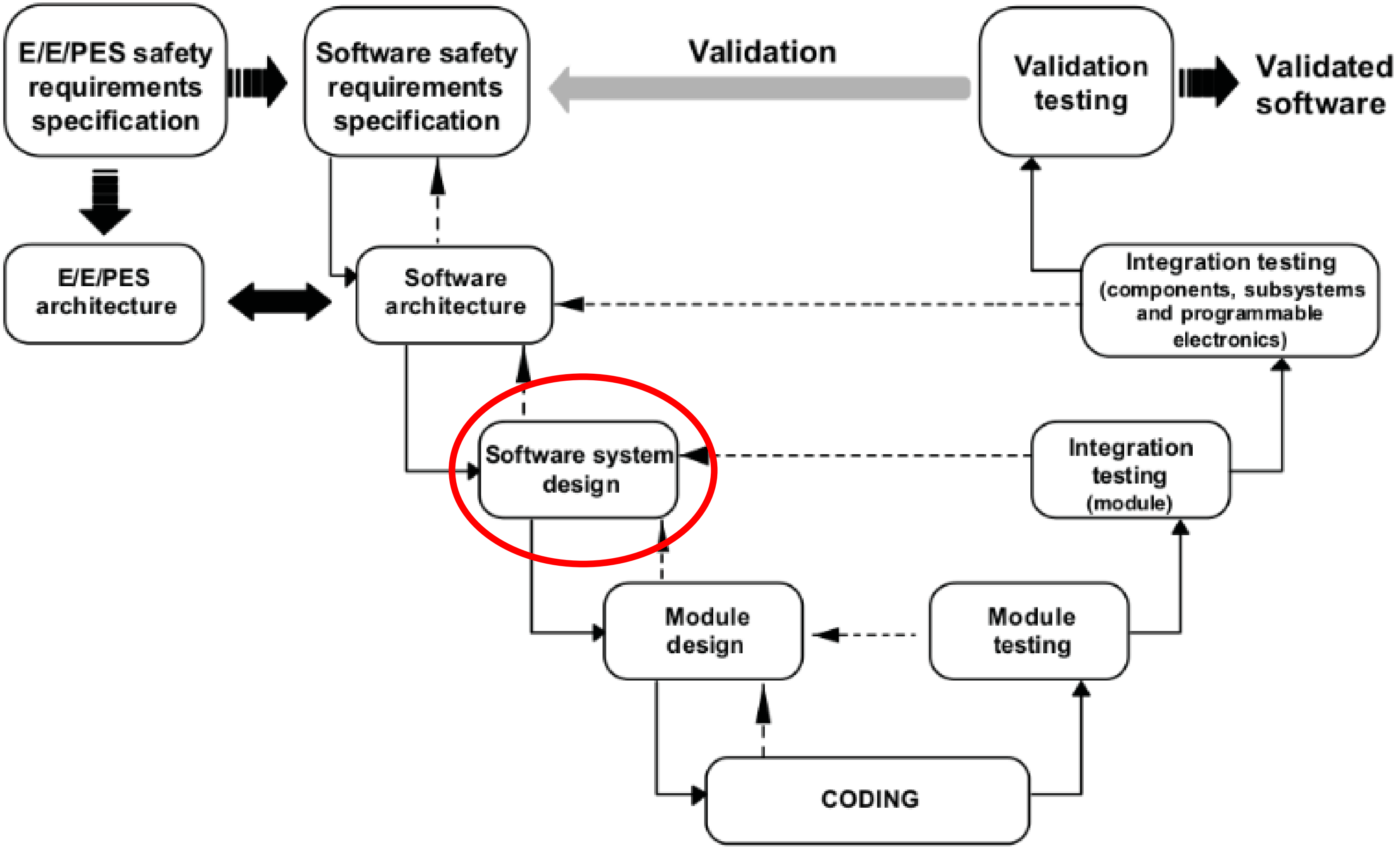
Christoph Lüth   Jan Peleska   Dieter Hutter

Universität Bremen

Where are we?

- 01: Concepts of Quality
- 02: Legal Requirements: Norms and Standards
- 03: The Software Development Process
- 04: Hazard Analysis
- 05: High-Level Design with SysML
- 06: Formal Modelling with SysML and OCL
- 07: Detailed Specification with SysML
- 08: Testing
- 09 and 10: Program Analysis
- 11: Model-Checking
- 12: Software Verification (Hoare-Calculus)
- 13: Software Verification (VCG)
- 14: Conclusions

Formal Modelling in the Development Cycle



What is OCL?

- OCL is the **Object Constraint Language**.
- What is OCL?
  - *„A formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model."* (OCL standard, §7)
- Why OCL?
  - „A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specification. There is, among other things, a need to describe additional constraints about the objects in the model. " (OCL standard, §7.1)

Characteristics of the OCL

- OCL is a pure **specificication language**.
  - OCL expressions do not have side effects.
- OCL is **not** a programming language.
  - Expressions are not executable (though some may be).
- OCL is **typed** language
  - Each expression has type; all expressions must be well-typed.
  - Types are classes, defined by class diagrams.

OCL can be used for the following:

- as a query language
- to specify invariants on classes and types in the class
- to specify type invariant for Stereotypes
- to describe pre- and post conditions on Operations and Methods
- to describe Guards
- to specify target (sets) for messages and actions
- to specify constraints on operations
- to specify derivation rules for attributes for any expression over a UML model.

(OCL standard, §7.1.1)

Example: A Flight-Booking System

- Flight destinations are given by
  - an IATA id, and a string
- A flight is given by
  - Source and destination, arrival and departure date, capacity and free seats
- A query asks for
  - a flight from/to at a given time and number of free seats
- Operations:
  - Query
  - Book a flight

Example: A Flight-Booking System

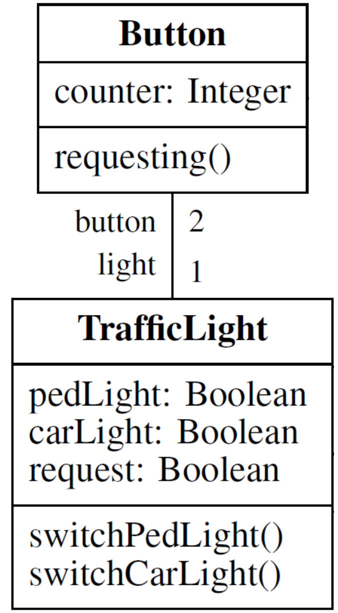
Possible constraints:

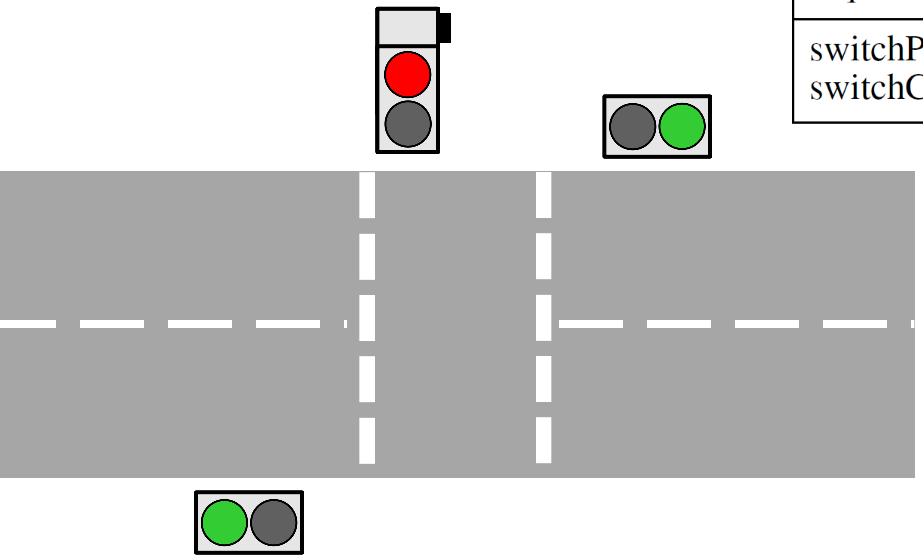
- No more free seats than capacity
- Source and destination must be disjoint
- Query must return „correct“ flight
- Destination identifiers must be unique
- To book a flight:
  - Possible if enough free seats
  - Afterwards, number of free seats reduced

Possible extension:

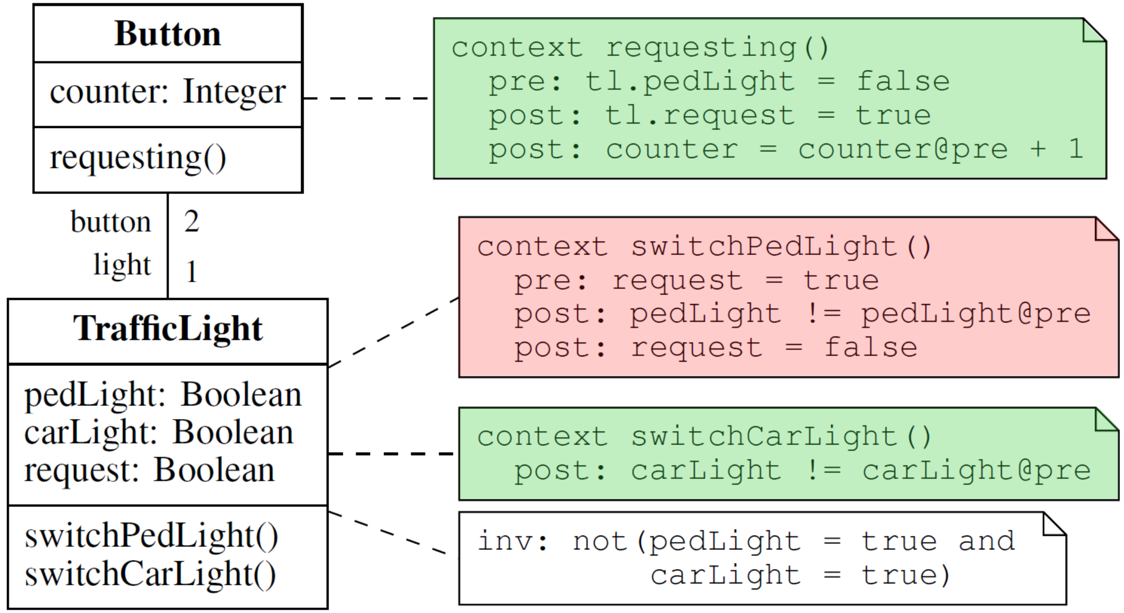
- Query returns a schedule --- list of connecting flights

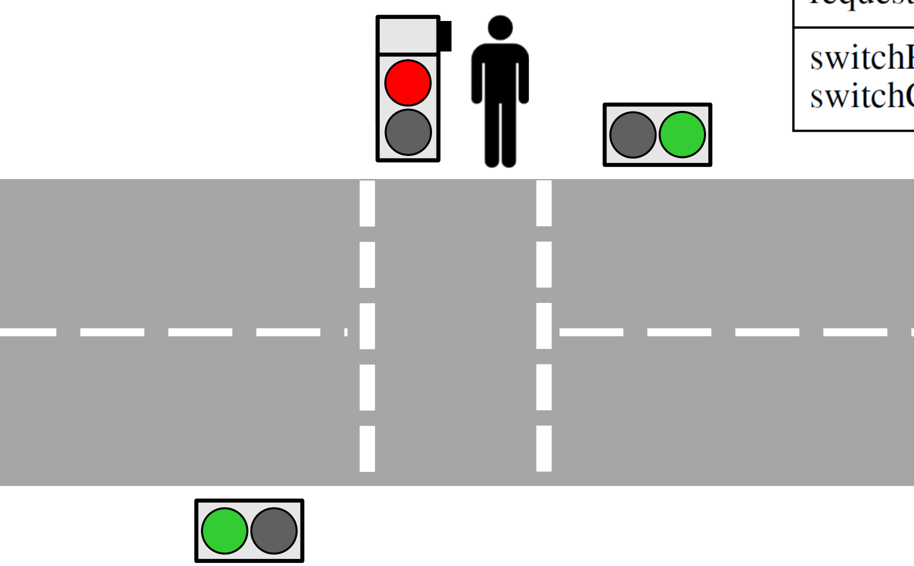
Example: The Traffic Light





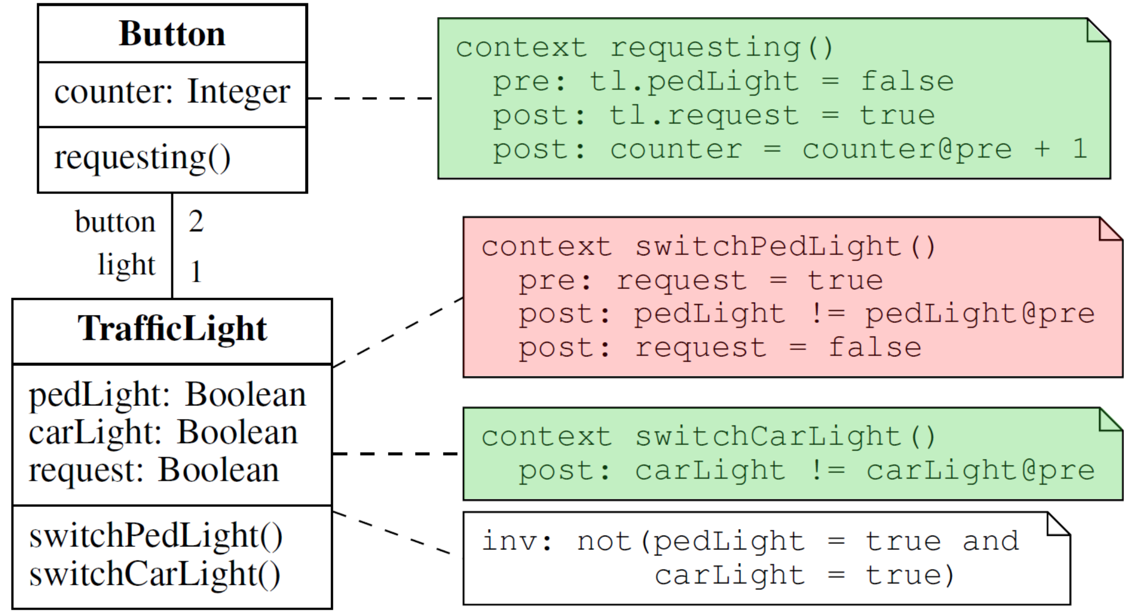
Example: The Traffic Light

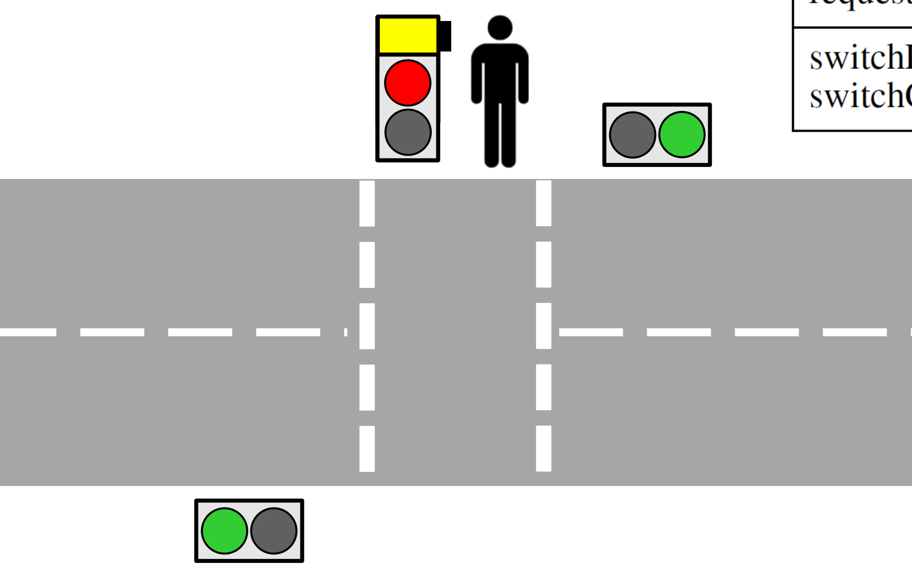




```
pedLight:     False
carLight:     True
request:      False
counter:      0
```

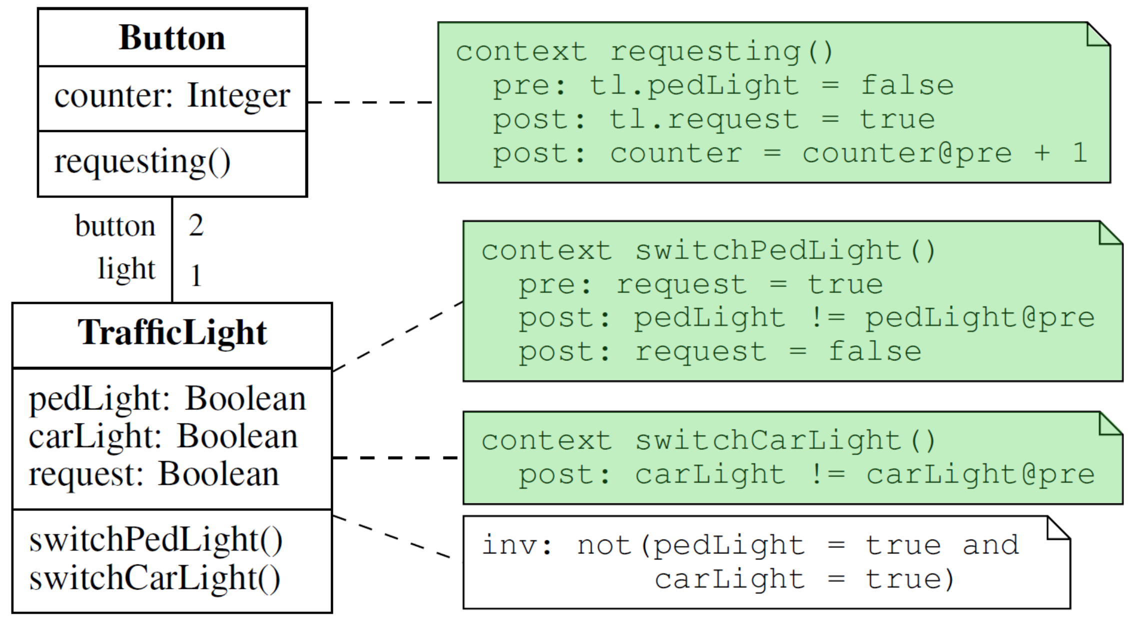
Example: The Traffic Light

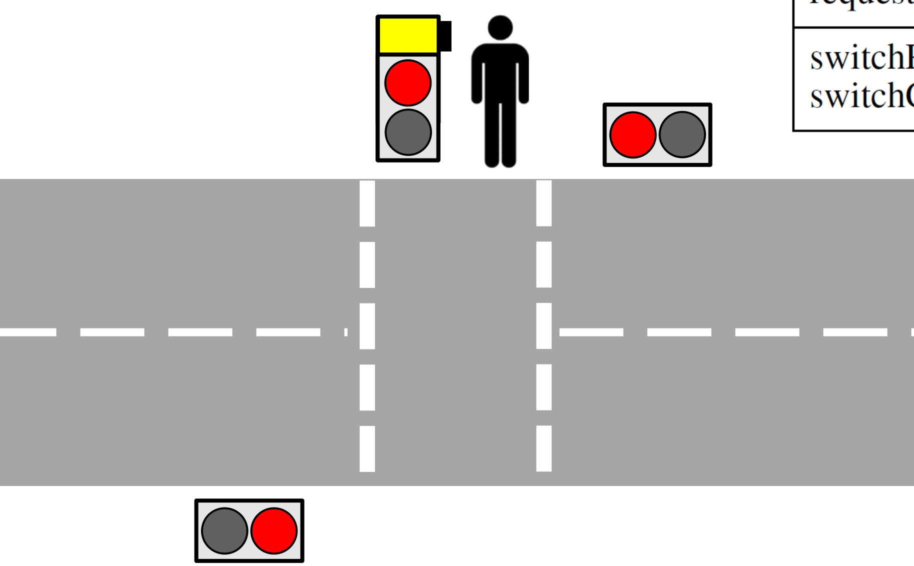




```
pedLight:     False
carLight:     True
request:      True
counter:      1
```

Example: The Traffic Light

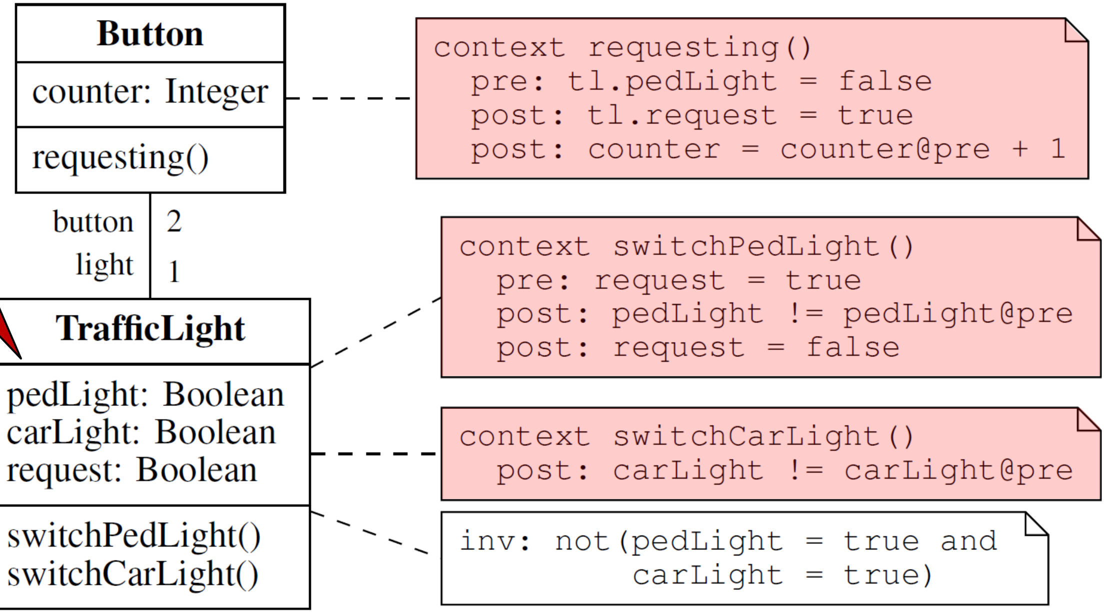


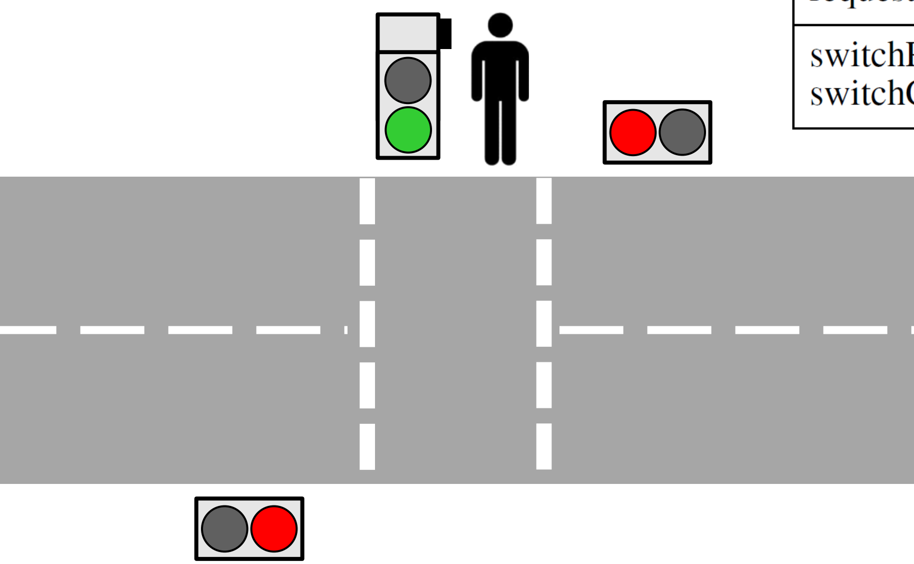


```
pedLight:     False
carLight:     False
request:      True
counter:      1
```

Example: The Traffic Light

Deadlock





pedLight:	True
carLight:	False
request:	False
counter:	1

OCL Basics

- The language is typed: each expression has a type.
- Three-valued logic (**Kleene logic**)
  - Actually, more like four-valued **`(null)`**
- Expressions always live in a **context:**
  - **Invariants** on classes, interfaces, types.

```
context Class
  inv Name: expr
```

  - **Pre/postconditions** on operations or methods

```
context Type :: op(a1: Type) : Type
  pre Name: expr
  post Name: expr
```

OCL Types

- Basic types:
  - **`Boolean, Integer, Real, String`**
  - **`OclAny, OclType, OclVoid`**
- Collection types:
  - **`Sequences, Bag, OrderedSet, Set`**
- Model types

Basic types and operations

- Integer ($\mathbb{Z}$) OCL-Std. §11.5.2

- Real ($\mathbb{R}$) OCL-Std. §11.5.1
  - **`Integer`** is a subclass of **`Real`**
  - **`round`**, **`floor`** from **`Real`** to **`Integer`**

- String (Zeichenketten) OCL-Std. §11.5.3
  - **`substring`**, **`toReal`**, **`toInteger`**, **`characters`**, etc.

- Boolean (Wahrheitswerte) OCL-Std. §11.5.4
  - **`or`**, **`xor`**, **`and`**, **`implies`**
  - Relationen auf **`Real`**, **`Integer`**, **`String`**

Collection Types

- Sequence, Bag, OrderedSet, Set — OCL-Std. §11.7
- Operations on all collections:
  - **`size, includes, count, isEmpty, flatten`**
  - Collections are always „flattened“
- Set
  - **`union, intersection`**
- Bag
  - **`union, intersection, count`**
- Sequence
  - **`first, last, reverse, prepend, append`**

Collection Types: Iterators

- Iterators are **higher-order functions**
- All iterators defined via **`iterate`** OCL-Std. §7.7.6

```
coll->iterate(elem: Type, acc: Type= expr | expr[el, acc])

iterate(e: T, acc: T= v)
{ acc= v;
  for (Enumeration e= c.elements(); e.hasMoreElements();) {
       e= e.nextElement();
       acc.add(expr[e, acc]);
       }
  return acc;
}
```

Model types

- Model types are given by
  - attributes,
  - operations, and
  - Associations of the model
- Navigation along the association
  - If cardinality is 1, type is of target type **`T`**
  - Otherise, it is **`Set(T)`**
- User-defined operations in expressions have to be stateless (stereotype `<<query>>`)

Undefinedness in OCL

- Undefinedness is **propagated** OCL-Std §7.5.11
  - In other words, all operations are **strict**
- Exceptions:
  - Boolean operators (`and`, `or` non-strict on **both sides**)
  - Case distinction
  - Test on definedness: `oclIsUndefined` with

$$oclIsUndefined(e) = \begin{cases} true & if\ e = \bot \\ false & otherwise \end{cases}$$

- Resulting logic is **three-valued** (Kleene-Logic)
- In fact, four-valued: there is always `null`
- Iterators are “semi-strict”

OCL Style Guide

- Avoid **complex** navigation („Loose coupling“)
  - Otherwise changes in models break OCL constraints
- Always choose **adequate context**
- „Use of `allInstances()` is **discouraged**“
- Split up invariants if possible
- Consider defining **auxiliary operations** if expressions become too complex.

Summary

- OCL is a typed, state-free specification language which allows us to denote constraints on models.
- We can define or models much more precise.
  - Ideally: no more natural language needed.
- OCL is part of the more „academic“ side of UML/SysML.
  - Tool support is not great, some tools ignore OCL, most tools at least type-check OCL, hardly any do proofs.
- However, in critical system development, the kind of specification that OCL allows is **essential.**
- Next week: detailed specification with SysML.
  - Behavioural diagrams: state diagrams, sequence charts ...