Requirements Engineering I

Chapter 8

Formal Specification Languages

$\Phi, \varepsilon \rightarrow \Psi$

Chapter roadmap

Φ,ε→Ψ

8.1 Algebraic specification

Elegant, but not practical

8.2 Model-based formal specification

Today's general approach to formal specification

8.3 An overview of Z

A classic model-based formal specification language

8.4 OCL (Object Constraint Language)

A popular formal specification language, embedded in UML

8.5 Proving properties

Beyond tests and beliefs

8.6 Benefits, limitations, and practical use

Where do formal specifications help?

What is a formal specification?

Requirements models with formal syntax and semantics

The vision

- Analyze the problem
- Specify requirements formally
- Implement by correctness-preserving transformations
- Maintain the specification, no longer the code

Typical languages

- “Pure” Automata / Petri nets
- Algebraic specification
- Temporal logic: LTL, CTL
- Set&predicate-based models: Z, OCL, Alloy, B

What does “formal” mean?

- Formal calculus, i.e., a specification language with
  - formally defined syntax

    and
  - formally defined semantics
- Primarily for specifying functional requirements

Potential forms

- Purely descriptive, e.g., algebraic specification
- Purely constructive, e.g., Petri nets
- Model-based hybrid forms, e.g. Alloy, B, OCL, VDM, Z

8.1 Algebraic specification

- Developed mid 1970ies for specifying complex data types
- Signatures of operations define the syntax
- Axioms (expressions being always true) define semantics
- Axioms describe properties that are invariant

```
TYPE Stack
...
push:   (Stack, elem)  →  Stack;
...
¬  full(s) → empty(push(s,e)) = false
...
```

+ Purely descriptive and mathematically elegant

– Hard to read

– Over- and underspecification difficult to spot

– Has never made it from research into industrial practice

8.2 Model-based formal specification

- Mathematical model of system state and state change
- Based on sets, relations and logic expressions
- Typical language elements
  - Base sets
  - Relationships (relations, functions)
  - Invariants (predicates)
  - State changes (by relations or functions)
  - Assertions for states

The formal specification language landscape

- VDM – Vienna Development Method (Björner and Jones 1978)
- Z (Spivey 1992)
- Alloy (Jackson 2002)
- TLA+ (Lamport 2003)
- B (Abrial 2009)
- OCL (OMG 2014)

8.3 An overview of Z

- A typical model-based formal language
- Only basic concepts covered here
- More detail in the literature, e.g., Jacky (1997)

The basic elements of Z

- Z is set-based
- Specification consists of sets, types, axioms and schemata
- Types are elementary sets: *[Name]* *[Date]* *IN*
- Sets have a type: *Person:* $\mathbb{P}$ *Name* *Counter: IN*
- Axioms define global variables and their (invariant) properties

string: ***seq*** *CHAR*	Declaration
#string $\leq$ *64*	Invariant

IN	Set of natural numbers
$\mathbb{P}$ *M*	Power set (set of all subsets) of *M*
seq	Sequence of elements
#M	Number of elements of set *M*

The basic elements of Z – 2

- Schemata
  - organize a Z-specification
  - constitute a name space

```
┌─ Counter ───────────────
│ Value, Limit: IN
├─────────────────────────
│ Value ≤ Limit ≤ 65535
└─────────────────────────
```

Name

Declaration part: Declaration of state variables

Predicate part:
- Restrictions
- Invariants
- Relationships
- State change

Relations, functions und operations

- Relations and functions are ordered set of tuples:

Order: ℙ (Part x Supplier x Date)

Birthday: Person $\rightarrow$ *Date*

A subset of all ordered triples (p, s, d) with p $\in$ *Part*, s $\in$ *supplier,* and d $\in$ *Date*

A function assigning a date to a person, representing the person's birthday

State change through operations:

Increment counter

Δ *Counter*

Value < Limit
Value' = Value + 1
Limit' = Limit

Δ *S*	The sets defined in schema S will be changed
M'	State of set M after executing the operation

Mathematical equality, no assignment!

Example: specification of a library system

The library has a stock of books and a set of persons who are library users.

Books in stock may be borrowed.

Library

Stock: $\mathbb{P}$ *Book*
User: $\mathbb{P}$ *Person*
lent: *Book* $\nrightarrow$ *Person*

---

dom *lent* $\subseteq$ *Stock*
ran *lent* $\subseteq$ *User*

$\nrightarrow$	Partial function
dom	Domain ...
ran	Range...
	...of a relation

Example: specification of a library system – 2

Books in stock which currently are not lent to somebody may be borrowed

Borrow

Δ *Library*
BookToBeBorrowed?: Book
Borrower?: Person

BookToBeBorrowed? $\in$ *Stock* **dom** *lent*
Borrower? $\in$ *User*
lent' = lent $\cup$ *{(BookToBeBorrowed?, Borrower?)}*
Stock' = Stock
User' = User

x?	*x* is an input variable
a $\in$ *X*	*a* is an element of set *X*
\	Set difference operator
$\cup$	Set union operator

Example: specification of a library system – 3

It shall be possible to inquire whether a given book is available

InquireAvailability

Ξ *Library*
InquiredBook?: Book
isAvailable!: {yes, no}

InquiredBook? $\in$ *Stock*
isAvailable! = **if** *InquiredBook?* $\notin$ **dom** *lent*
then *yes* **else** *no*

Ξ *S*	The sets defined in schema S can be referenced, but not changed
x!	x is an output variable

Mini-Exercise: Specifying in Z

Specify a system for granting and managing authorizations for a set of individual documents.

The following sets are given:

Authorization

Stock $\mathbb{P}$ *Document*
Employee: $\mathbb{P}$ *Person*
authorized: $\mathbb{P}$ *(Document x Person)*
prohibited: $\mathbb{P}$ *(Document x Date)*

Specify an operation for granting an employee access to a document as long as access to this document is not prohibited. Use a Z-schema.

.

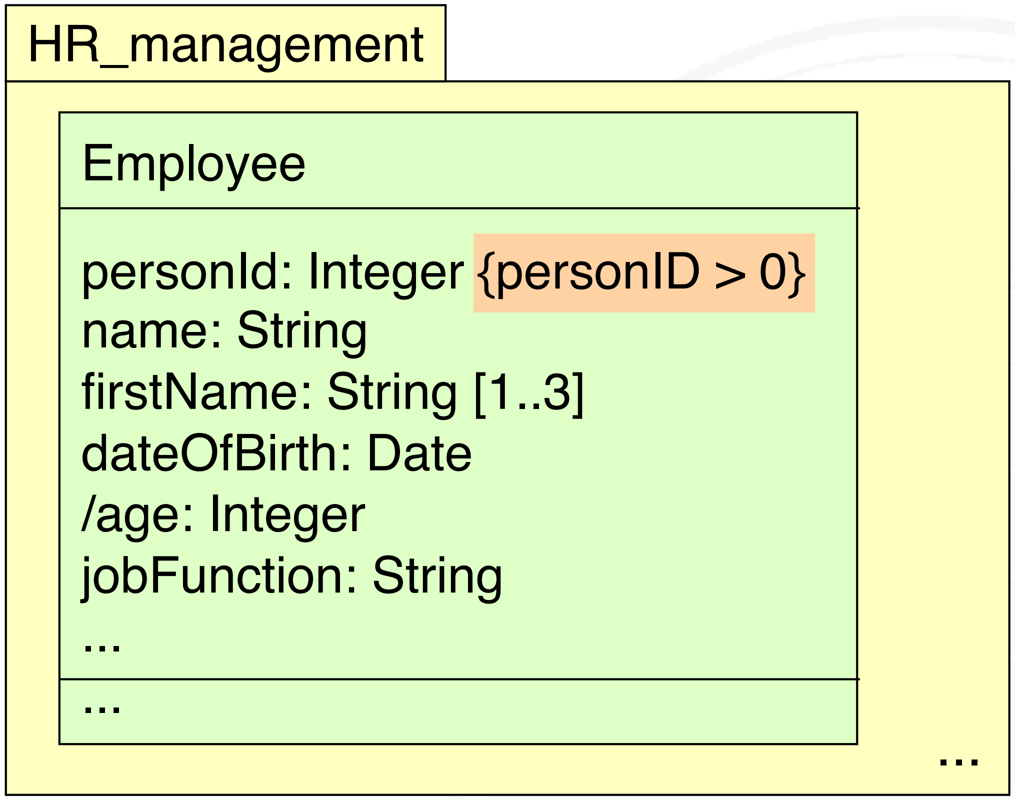
8.4 OCL (Object Constraint Language)

- What is OCL?
  - A textual formal language
  - Serves for making UML models more precise
  - Every OCL expression is attached to an UML model element, giving the context for that expression
  - Originally developed by IBM as a formal language for expressing integrity constraints (called ICL)
  - In 1997 integrated into UML 1.1
  - Current standardized version is Version 2.4 of 2014

Why OCL?

- Making UML models more precise
  - Specification of invariants (i.e., additional restrictions) on UML models
  - Specification of the semantics of operations in UML models
- Also usable as a language to query UML models

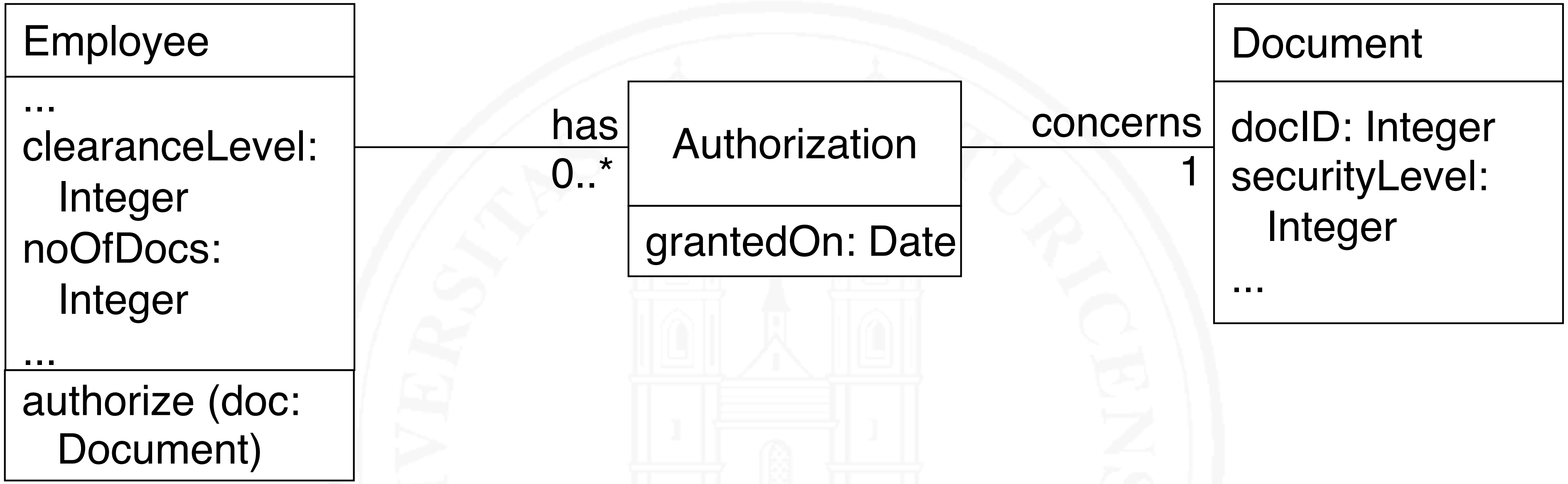
OCL expressions: invariants



context HR_manangement::Employee **inv**:
self.jobFunction = “driver” **implies** self.age ≥ 18

- OCL expression may be part of a UML model element
- Context for OCL expression is given implicitly
- OCL expression may be written separately
- Context must be specified explicitly

OCL expressions: Semantics of operations



context Employee::authorize (doc: Document)
pre: self.clearanceLevel ≥ doc.securityLevel
post: noOfDocs = noOfDocs@pre + 1
and
self.has->**exists** (a: Authorization | a.concerns = doc)

Navigation, statements about sets in OCL

- Persons having Clearance level 0 can’t be authorized for any document:

context Employee **inv**: self.clearanceLevel = 0 **implies**
self.has->isEmpty()

Navigation from current object to a set of associated objects (self.has)

Application of a function to a set of objects (->isEmpty())

Navigation, statements about sets in OCL – 2

More examples:

- The number of documents listed for an employee must be equal to the number of associated authorizations:

  context Employee **inv**: self.has->size() = self.noOfDocs

- The documents authorized for an employee are different from each other

  context Employee **inv**: self.has->**forAll** (a1, a2: Authorization |
  a1 <> a2 **implies** a1.concerns.docID <> a2.concerns.docID)

- There are no more than 1000 documents:

  context Document **inv**: Document.allInstances()->size() ≤ 1000

Summary of important OCL constructs

- Kind and context: **context**, **inv**, **pre**, **post**
- Boolean logic expressions: **and**, **or**, **not**, **implies**
- Predicates: **exists**, **forAll**
- Alternative: **if then else**
- Set operations: size(), isEmpty(), notEmpty(), sum(), ...
- Model reflection, e.g., *self.oclIsTypeOf (Employee)* is true in the context of Employee
- Statements about all instances of a class: allInstances()
- Navigation: dot notation — self.has.date = ...
- Operations on sets: arrow notation — self.has->size()
- State change: @pre notation — noOfDocs = noOfDocs@pre + 1

8.5 Proving properties

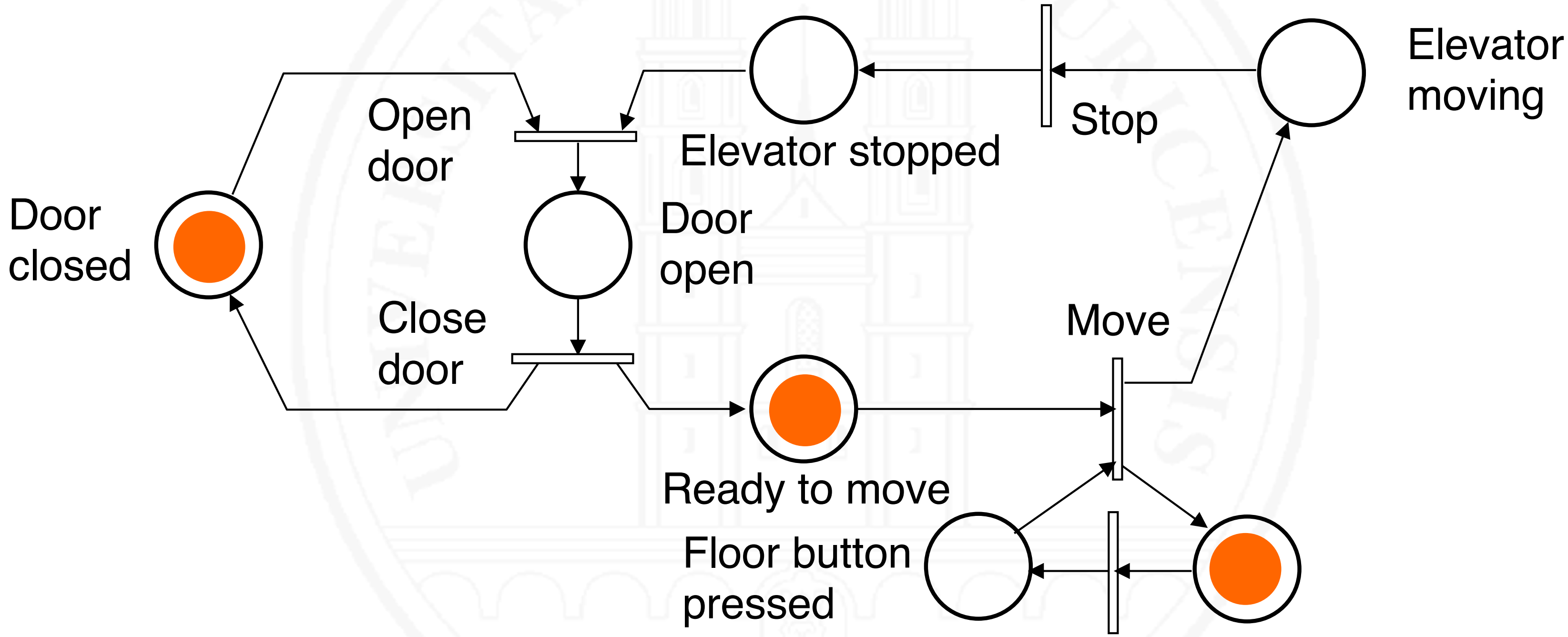
Formal specifications enable proofs (e.g., safety invariants)

- Classic proofs (usually supported by theorem proving software) establish that a property can be inferred from a set of given logical statements
- Model checking explores the full state space of a model, demonstrating that a property holds in every possible state

– Classic proofs are still hard and labor-intensive
+ Model checking is fully automatic and produces counter-examples in case of failure
– Exploring the full state state space is frequently infeasible
+ Exploring feasible subsets is a systematic, automated test

Example: Proving a safety property

A (strongly simplified) elevator control system has been modeled with a Petri net as follows:



The property that an elevator never moves with doors open shall be proved

Example: Proving a safety property – 2

The property to be proven can be restated as:

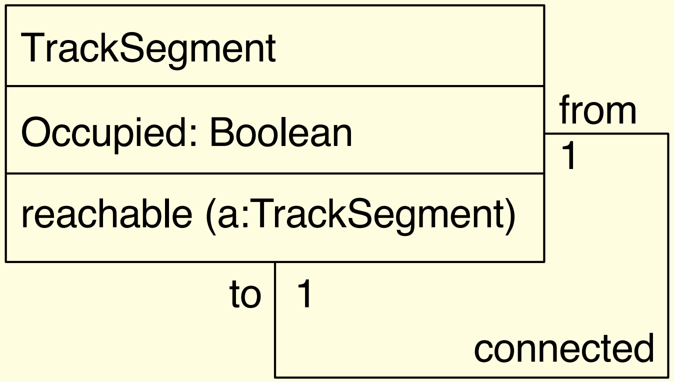
(P) The places *Door open* and *Elevator moving* never hold tokens at the same time

Due to the definition of elementary Petri Nets we have

- The transition *Move* can only fire if *Ready to move* has a token (1)
- There is at most one token in the cycle *Ready to move – Elevator moving – Elevator stopped – Door open* (2)
- (2) ⇒ If *Ready to move* or *Elevator moving* have a token, *Door open* hasn't one (3)
- If *Door open* has no token, *Door closed* must have one (4)
- (1) & (3) & (4) ⇒ (P) ▯

Mini-Exercise: A circular metro line

A circular metro line with 10 track segments has been modeled in UML and OCL as follows:



Context TrackSegment::
  reachable (a: TrackSegment): Boolean
  post:
  result = (self.to = a) **or** (self.to.reachable (a))

context TrackSegment **inv**:
  TrackSegment.allInstances->size = 10

In a circle, every track segment must be reachable from every other track segment (including itself). So we must have:

context TrackSegment **inv** (1)
  TrackSegment.allInstances->forAll (x, y I x.reachable (y))

a) Falsify this invariant by finding a counter-example

Mini-Exercise: A circular metro line – 2

Only the following trivial invariant can be proved:

context TrackSegment **inv**:
TrackSegment.allInstances->forAll (x | x.reachable (x))

b) Prove this invariant using the definition of *reachable*

Obviously, this model of a circular metro line is wrong. The property of being circular is not mapped correctly to the model.

c) How can you modify the model such that the original invariant (1) holds?

8.6 Benefits, limitations, and practical use

Benefits

- Unambiguous by definition
- Fully verifiable
- Important properties can be
  - proven
  - or tested automatically (model checking)

Limitations / problems

- Cost vs. value
- Stakeholders can't read the specification: how to validate?
- Primarily for functional requirements

Role of formal specifications in practice

- Marginally used in practice
  - Despite its advantages
  - Despite intensive research (dating back to 1977)
- Actual situation today
  - Punctual use possible and reasonable, in particular
    - Safety-critical components
    - Complex distributed systems (Newcombe et al. 2015)
  - However, broad usage
    - not possible (due to validation problems)
    - not reasonable (cost exceeds benefit)
- Alternative: Formalize critical parts of semi-formal models