

# Formale Modellierung

## Vorlesung 11 vom 28.06.2014: Formale Modellierung von Software

Christoph Lüth

Universität Bremen

Sommersemester 2015

# Fahrplan

- ▶ Teil I: Formale Logik
- ▶ Teil II: Spezifikation und Verifikation
  - ▶ Formale Modellierung von Software
  - ▶ Temporale Logik und Modellprüfung
  - ▶ Zusammenfassung, Rückblick, Ausblick

# Das Tagesmenü

- ▶ Modellierung von Software: **Spezifikation**
  - ▶ Modellierung des **Verhaltens** (nicht des **Programmes**)
- ▶ Ein **weites** Feld:
  - ▶ Entwicklungsmodelle, Vorgehensmodelle, ...
  - ▶ Informelle Sprachen (**UML**)
- ▶ Hier: **formale Spezifikation**

# Algebraische Spezifikation

- ▶ Idee: Spezifikation ist **Signatur**, Programme sind **Algebren**
- ▶ Mathematische Grundlage: universelle **Algebra**
- ▶ **Geschichtliches:**
  - ▶ **Entstanden** um 1976 (ADJ-Gruppe)
  - ▶ In den 80ern **Vielzahl** von algebraischen Sprachen
  - ▶ Ende 90er Entwicklung der **Einheitssprache CASL**
- ▶ Beispielsprachen: CASL, OBJ, Maude

# Die Grundidee

- ▶ Deklaration von Typen und Operationen in **Signatur**
- ▶ Gewünschte **Eigenschaften** als **Axiome**
- ▶ Semantik: **lose** (alle Algebren) vs. **initial** (Termalgebra)

# Ein klassisches Beispiel

- ▶ Ein **Stack** hat zwei **Sorten** und vier **Operationen**:

```
typedecl 'a stack
```

```
axiomatization
```

```
  empty :: "'a stack"
```

```
  push  :: "'a stack => 'a => 'a stack"
```

```
  pop   :: "'a stack => 'a stack"
```

```
  top   :: "'a stack => 'a"
```

- ▶ **Axiome**: pop und top invers zu push

```
where a1: "top (push s x) = x"
```

```
      a2: "pop (push s x) = s"
```

- ▶ pop, top partiell?
- ▶ Keine Seiteneffekte

# Modellbasierte Spezifikation

- ▶ Grundidee: Konstruktion eines (nicht-ausführbaren) Modells
- ▶ Basiert auf konsistenter, ausdrucksstarker Logik:
  - ▶ Mengenlehre (getypt, ungetypt),
  - ▶ Typtheorie
  - ▶ HOL
- ▶ Geschichtliches:
  - ▶ VDM, entwickelt früher 70er (IBM-Labor Wien)
  - ▶ Früher industrieller Einsatz
  - ▶ Standardisierung in den 90ern (VDM, Z)
- ▶ Beispielsprachen: VDM, Z, B

# Das klassische Beispiel

- ▶ Der Stack modellbasiert:
  - ▶ **Sehr** einfach — der Stack ist eine Liste

```
type_synonym 'a stack = "'a list"
definition empty :: "'a stack"
where      "empty == []"
definition push  :: "'a stack => 'a => 'a stack"
where     "push s a == a# s"
definition pop   :: "'a stack => 'a stack"
where     "pop s == tl s"
definition top   :: "'a stack => 'a"
where     "top s == hd s"
```

# Vor- und Nachteile

- ▶ Algebraische Spezifikationen:
  - ▶ Abstrakter, leichter zu schreiben
  - ▶ **aber** werden leicht inkonsistent
- ▶ Modellbasierte Spezifikationen:
  - ▶ Konsistenz garantiert, ausdrucksmächtiger
  - ▶ **aber** manchmal **zu** mächtig

# Weitere Modellierungssprachen

- ▶ JML — light-weight oder code-based specification
- ▶ UML — semi-formal

# Java Modeling Language (JML)

- ▶ Zentral: funktionale Korrektheit
- ▶ “Design by contract”
- ▶ Spezifikation nahe am Code (Annotationen)
- ▶ Vor/Nachbedingungen, Invarianten
- ▶ Werkzeuge: ESC/Java2, Mobius

## JML: Erstes Beispiel

```
public abstract class LinearSearch
{
    //@ requires j >= 0;
    public abstract /*@ pure @*/ boolean f(int j);

    //@ ensures 0 <= \result;
    //@ ensures (\exists int j; 0 <= j && j <= \result; f(j));
    public abstract /*@ pure @*/ int limit();

    /*@ public normal_behavior
        @   requires (\exists int i; 0 <= i && i <= limit(); f(i));
        @   assignable \nothing;
        @   ensures f(\result) &&
        @           (\forall int i; 0 <= i && i < \result; ! f(i));
        @*/
    public int find()
}
}
```

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	Ja
Sequenzdiagramm	Kommunikation	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	Ja
Sequenzdiagramm	Kommunikation	Ja
Kommunikationsdiagramm	Struktur der Kommunikation	

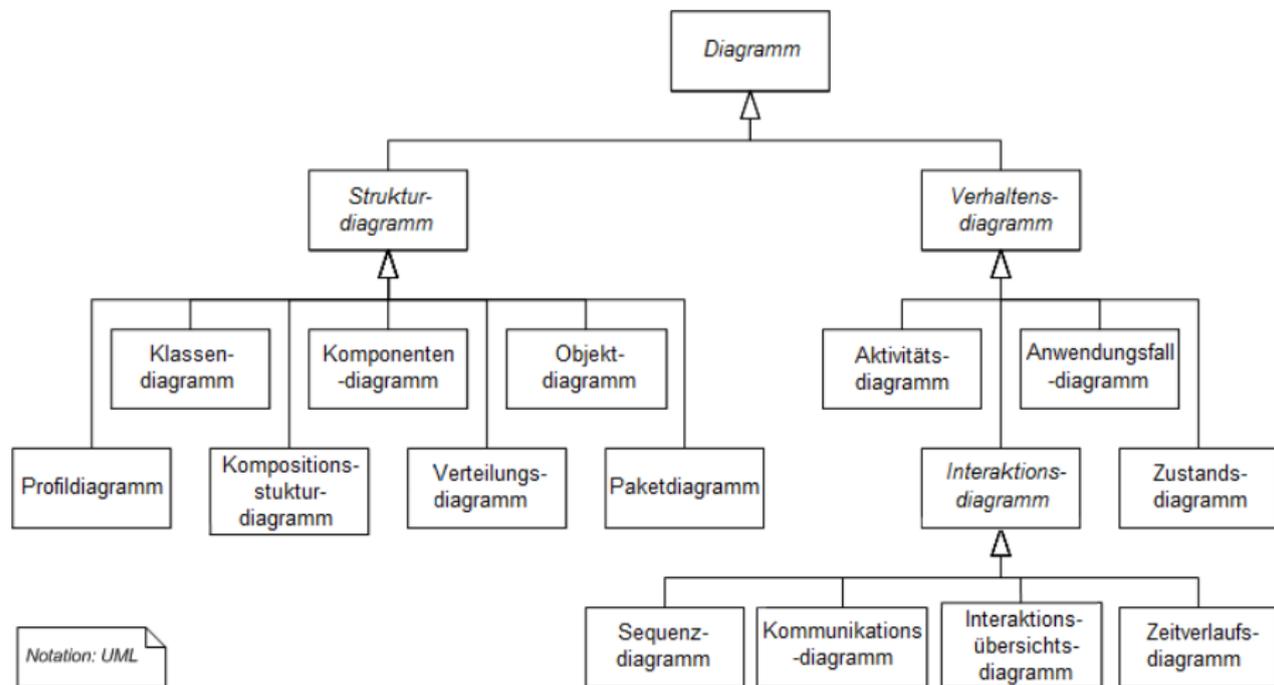
# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	Ja
Sequenzdiagramm	Kommunikation	Ja
Kommunikationsdiagramm	Struktur der Kommunikation	(Ja)
Zeitverlaufdiagramm	Echtzeitaspekte	

# UML als **formale** Spezifikationsprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	Ja
Sequenzdiagramm	Kommunikation	Ja
Kommunikationsdiagramm	Struktur der Kommunikation	(Ja)
Zeitverlaufdiagramm	Echtzeitaspekte	(Ja)

# Diagramme in UML 2.3



Quelle: Wikipedia

# OCL

- ▶ Object Constraint Language
- ▶ Mathematisch präzise Sprache für UML
- ▶ OO meets Z
- ▶ Entwickelt in den 90ern
- ▶ Formale Constraints an UML-Diagrammen

# OCL Basics

- ▶ **Getypte** Sprache
- ▶ Dreiwertige Logik
- ▶ Ausdrücke immer im **Kontext**:
  - ▶ **Invarianten** an Klassen, Interfaces, Typen
  - ▶ **Vor/Nachbedingungen** an Operationen oder Methoden

# OCL Syntax

- ▶ Invarianten:

```
context class
  inv: expr
```

- ▶ Vor/Nachbedingungen:

```
context Type :: op(arg1 : Type) : Return Type
  pre: expr
  post: expr
```

- ▶ expr ist ein OCL-Ausdruck vom Typ Boolean

# Undefiniertheit in OCL

- ▶ Undefiniertheit **propagiert** (alle Operationen **strikt**)
- ▶ Ausnahmen:
  - ▶ Boolesche Operatoren (and, or **beidseitig** nicht-strikt)
  - ▶ Fallunterscheidung
  - ▶ Test auf Definiertheit: `oclIsUndefined` mit

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

- ▶ Resultierende Logik: **dreiwertig**

# Dreiwertige Logik

- ▶ Wahrheitstabelle (starke Kleene-Logik,  $K_3$ ):

	$\neg$	$\wedge$	$\perp$	0	1	$\vee$	$\perp$	0	1
$\perp$	$\perp$	$\perp$	$\perp$	0	$\perp$	$\perp$	$\perp$	$\perp$	1
0	1	0	0	0	0	0	$\perp$	0	1
1	0	1	$\perp$	0	1	1	1	1	1

$\longrightarrow$	$\perp$	0	1
$\perp$	$\perp$	$\perp$	1
0	1	1	1
1	$\perp$	0	1

$\longleftrightarrow$	$\perp$	0	1
$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	1	0
1	$\perp$	0	1

- ▶ Fun Fact:  $K_3$  hat keine Tautologien.
- ▶ Alternative: schwache Kleene-Logik (alle Operatoren strikt)

# OCL Typen

- ▶ Basistypen:
  - ▶ Boolean, Integer, Real, String
  - ▶ OclAny, OclType, OclVoid
- ▶ Collection types: Set, OrderedSet, Bag, Sequences
- ▶ Modelltypen

# Basistypen und Operationen

- ▶ Integer ( $\mathbb{Z}$ )
- ▶ Real ( $\mathbb{R}$ )
  - ▶ Integer Subklasse von Real
  - ▶ round, floor von Real nach Integer
- ▶ String (Zeichenketten)
  - ▶ substring, toReal, toInteger, characters etc.
- ▶ Boolean (Wahrheitswerte)
  - ▶ or, xor, and, implies
  - ▶ Sowie Relationen auf Real, Integer, String

# Collection Types

- ▶ Set, OrderedSet, Bag, Sequence
- ▶ Operationen auf allen Kollektionen:
  - ▶ size, includes, count, isEmpty, flatten
  - ▶ Kollektionen werden immer flachgeklopft
- ▶ Set
  - ▶ union, intersection,
- ▶ Bag
  - ▶ union, intersection, count
- ▶ Sequence
  - ▶ first, last, reverse, prepend, append

## Collection Types: Iteratoren

- ▶ Iteratoren: Funktionen höherer Ordnung
- ▶ Alle definiert über `iterate` :

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

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

- ▶ Iteratoren sind alle **strikt**

# Zusammenfassung

- ▶ Formale **Spezifikation** definieren das **Verhalten** eines Softwaresystems
- ▶ Verschiedene Ansätze:
  - ▶ **Algebraische** Spezifikation (axiomatisch, abstrakt)
  - ▶ **Modellbasierte** Spezifikation (konkret, konsistent)
  - ▶ **Leichtgewichtige** Spezifikation (annotierter Code)
- ▶ In der Praxis oft hybride Ansätze (z.B. UML)