

# Formale Methoden der Softwaretechnik Vorlesung vom 09.06.11: Das Isabelle-System

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2011

Rev. 1483

1 [19]

## Fahrplan

- ▶ Aussagenlogik
- ▶ Prädikatenlogik
- ▶ Isabelle I: Grundlagen
  - ▶ Aussagenlogik und natürliches Schließen
  - ▶ Prädikatenlogik und Quantoren
  - ▶ Logik höherer Stufe
  - ▶ Isabelle: Definitionen und konservative Erweiterung
  - ▶ Isabelle: Automatische Beweisprozeduren
- ▶ Isabelle II: Anwendungen

2 [19]

## Heute

- ▶ Vertrauenswürdigkeit von Isabelle
  - ▶ Systemarchitektur
  - ▶ Konservative Erweiterung
- ▶ Typdefinitionen
  - ▶ Nützliche Typen

3 [19]

## Vertrauenswürdigkeit

- ▶ Wann können wir Isabelle-Beweisen **trauen**?
  1. Sinnvolle Modellierung
  2. Isabelle korrekt implementiert
  3. Logik konsistent
- ▶ Problem: ca. 150 Kloc SML Quellcode, ca. 310 Kloc Beweise
- ▶ Maßnahmen:
  1. Review
  2. LCF-Systemarchitektur
  3. Konservative Erweiterung

4 [19]

## LCF-Architektur

- ▶ Problem: Korrektheit der Implementierung
- ▶ Reduktion des Problems:
  - ▶ Korrektheit eines logischen Kerns
  - ▶ Rest durch Typisierung
- ▶ Abstrakter Datentyp `thm`, Inferenz-Regeln als Operationen

```
val assume: cterm -> thm
val implies_intr: cterm -> thm -> thm
```
- ▶ Logischer Kern:
  - ▶ Typcheck, Signaturen, Unifikation, Meta-Logik: ca. 5500 LOC
  - ▶ Handhabbare Größe

5 [19]

## Konservative Erweiterung

- ▶ Signatur  $\Sigma = \langle T, \Omega \rangle$ 
  - ▶ Typdeklarationen  $T$ , Operationen  $\Omega$
  - ▶ Definiert die Syntax: Terme  $T_\Sigma$  über  $\Sigma$
- ▶ Theorie  $\mathcal{Th} = \langle \Sigma, \mathcal{Ax} \rangle$ 
  - ▶ Axiome  $\mathcal{Ax}$
  - ▶ Theoreme:  $\mathcal{Thm}(\mathcal{Th}) \stackrel{\text{def}}{=} \{t \mid \mathcal{Th} \vdash t\}$
- ▶ Erweiterung:  $\Sigma \subseteq \Sigma', \mathcal{Th} \subseteq \mathcal{Th}'$ 
  - ▶ Konservativ gdw.  $t \in \mathcal{Thm}(\mathcal{Th}'), t \in T_\Sigma$  dann  $t \in \mathcal{Thm}(\mathcal{Th})$
  - ▶ Keine neuen Theoreme über alte Symbole

6 [19]

## Konservative Erweiterung in Isabelle

- ▶ Isabelle: Konstruktion von Theorien durch konservative Erweiterungen
- ▶ Konstantendefinition (zur Signatur  $\Sigma$ ):
$$c :: \sigma \quad c \equiv t$$
  - ▶  $c \notin \Sigma$
  - ▶  $t \in T_\Sigma$  (enthält nicht  $c$ )
  - ▶ Typvariablen in  $t$  auch in  $\sigma$
- ▶ Lemma: Konstantendefinition ist konservative Erweiterung
- ▶ Weitere konservative Erweiterungen:
  - ▶ Typdefinitionen, Datentypen.

7 [19]

## Isabelle Theorien

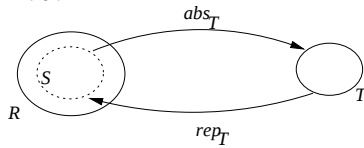
- ▶ Strukturierung der Entwicklung in Theorien
  - ▶ Theorien bilden DAG
- ▶ Theorieelemente (Auszug):
  - ▶ Konstantendeklaration: `consts f :: T`
  - ▶ Konstantendefinitionen `definition f :: T where "f == E"`
    - ▶  $E$  darf  $f$  nicht enthalten,  $FV(E) \subseteq FV(f)$ ,  $FV_{Type}(E) \subseteq FV_{Type}(f)$
  - ▶ Typdeklaration: `typedec1 t`
  - ▶ Typsynonym: `types t = S`
  - ▶ Beweise: `lemma` oder `theorem`
  - ▶ Sonstiges (Syntax erweiterbar)

8 [19]

## Typdefinitionen in Isabelle

**Definition** eines neuen Typen:

- Gegeben Typ  $R$ , Prädikat  $S : R \Rightarrow \text{Bool}$
- Neuer Typ  $T$  mit  $\text{abs}_T : R \Rightarrow T, \text{rep}_T : T \Rightarrow R$  so dass
  - $S$  und  $T$  **isomorph**:  $\forall t. \text{abs}_T(\text{rep}_T t) = t, \forall r. S r \longrightarrow \text{rep}_T(\text{abs}_T r) = r$
  - $T$  **nicht leer**:  $\exists t. S t$



- Lemma**: Typdefinitionen sind **konservative Erweiterungen**.
- In dieser Form **selten** direkt benutzt.

9 [19]

## Beispiel: Produkte

- Ausgangstyp  $R_{\alpha,\beta} \equiv \alpha \Rightarrow \beta \Rightarrow \text{Bool}$
- Neuer Typ  $T_{\alpha,\beta} \equiv \alpha \times \beta$ 
  - Idee: Prädikat  $p : \alpha \Rightarrow \beta \Rightarrow \text{Bool}$  repräsentiert  $(a, b)$   
 $p \ x \ y = \text{true} \iff x = a \wedge y = b$
- $S \ f = \exists a \ b. (f = \lambda x \ y. x = a \wedge y = b)$
- Repräsentationsfunktion:  
 $\text{rep}_\times(a, b) = p$  mit  $p \ x \ y = \text{true} \iff x = a \wedge y = b$
- Abstraktionsfunktion:  
 $\text{abs}_\times p = \iota a \ b. p \ a \ b$
- Damit **Eigenschaften** als **Theoreme** herleitbar.

10 [19]

## Nützliche Isabelle/HOL-Typen

- Vordefinierte Typen
- Algebraische Datentypen
- Benannte Produkte (labelled records)

11 [19]

## Natürliche Zahlen und unendliche Datentypen

- Natürliche Zahlen: nicht **konservativ**!
- Erfordert **zusätzliches** Axiom **Unendlichkeit**.
- Neuer Typ  $\text{ind}$  mit Konstanten  $Z :: \text{ind}, S : \text{ind} \Rightarrow \text{ind}$  und  
 $\text{inj } S \quad S \ x \neq Z$
- Damit  $\text{nat}$  definierbar.
- Warum  $\text{ind}$ ? Auch für **andere** Datentypen

12 [19]

## Nützliche vordefinierte Typen

| Theorie      | Typ       | Bedeutung                                          |
|--------------|-----------|----------------------------------------------------|
| Set          | 'a set    | getypte <b>Mengen*</b> ('a => bool)                |
| Sum          | 'a + 'b   | Summentyp                                          |
| Product_Type | 'a * 'b   | Produkttyp                                         |
| Nat          | nat       | Natürliche Zahlen                                  |
| Int          | int       | ganze Zahlen                                       |
| Real         | real      | reelle Zahlen                                      |
| Complex      | complex   | komplexe Zahlen                                    |
| Datatype     | 'a option | Haskell's Maybe, Some x oder None                  |
| List         | 'a list   | <b>Listen</b>                                      |
| Map          | 'a -> 'b  | partielle <b>Abbildungen</b><br>( 'a => 'b option) |

\* Typsynonym

<http://isabelle.in.tum.de/dist/library/HOL/index.html>

13 [19]

## Algebraische Datentypen

- In Isabelle/HOL wie in Haskell
- Eigenschaften werden (hinter den Kulissen) **bewiesen**:
  - Injektivität** der Konstruktoren
  - Disjunktheit** der Konstruktoren
  - Generiertheit** durch Konstruktoren
- Abgeleitete Schemen:
  - Induktion
  - Fallunterscheidung
- Einschränkung: nur **kovariante** Rekursion

14 [19]

## Algebraische Datentypen

Beispiel:

```
datatype Colour = Red | Green | Blue | Yellow | White
```

```
datatype Result = OK | Error nat | Exception string
```

```
datatype 'a Tree = Node "'a Tree" 'a "'a Tree"
                  | Leaf
```

```
datatype 'a NTree = Node 'a "nat => 'a NTree"
```

Nicht erlaubt:

```
datatype T = C "T=> bool"
```

15 [19]

## Labelled Records

- In Isabelle/HOL ähnlich wie in Haskell
- Syntax:

```
record point2d = x :: int
                y :: int
```

Definiert:

- Konstruktoren** ( $|x = 3, y = 4|$ )
- Selektoren**  $x :: \text{point2d} \Rightarrow \text{int}$
- Update**  $p \ (|x := 3 \ |)$

Erweiterbar:

```
record point3d = point2d +
                z :: int
```

16 [19]

## Definition von Funktionen

- ▶ In HOL: alle Funktionen **total**
  - ▶ Need-to-know Prinzip: undefiniert ist unbeweisbar
- ▶ Der einfache Fall: **primitiv rekursiv**
- ▶ Der allgemeine Fall
- ▶ Immer mit **Terminationsbeweis**

17 [19]

## Rekursion

Primitiv:

**fun**  $f :: \tau$

**where**

*equations*

Allgemein:

**function**  $f :: \tau$

**where**

*equations*

**by** *completeness-proof*

**termination by** *termination-proof*

18 [19]

## Zusammenfassung

- ▶ Isabelle: **LCF-Architektur** mit logischem Kern und Meta-Logik
- ▶ Konservative Erweiterung: erhält **Konsistenz**
- ▶ **Typdefinitionen**: Einschränkung bestehender Typen
- ▶ Der Typ *ind*: **Unendlichkeitsaxiom**
- ▶ Nützliche Typen: vordefinierte, algebraische Datentypen, labelled records
- ▶ Funktionsdefinition: **fun** und **function**
  - ▶ Fast wie in Haskell, aber immer **total**.

19 [19]