Formale Methoden der Softwaretechnik 1
Vorlesung vom 30.11.09: Isabelle/HOL

Christoph Lüth, Lutz Schröder

Universität Bremen

Wintersemester 2009/10

Fahrplan

- Teil I: Grundlagen der Formalen Logik
- Teil II: Arbeiten mit Isabelle
  - Grundlagen von Isabelle
  - Logik höherer Ordnung in Isabelle
  - Isabelle/HOL
  - Beweise über funktionale Programme in Isabelle/HOL
  - Beweisen mit Isabelle: Simplifikation, Taktiken
- Teil III: Modellierung imperative Programme

Heute

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

Generizität

- Isabelle ist ein logisches Rahmenwerk
- Beliebige Logiken in Isabelle einbetten
- Bsp: HOL, ZF, LK, Typentheorie ...
- Hier nur HOL

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

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

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

Konservative Erweiterung

- Signatur $\Sigma = \langle T, \Omega \rangle$
  - Typdeklarationen T, Operationen Ω
  - Definiert die Syntax: Terme T_Σ über Σ
- Theorie $\mathcal{T}h = \langle \Sigma, \mathcal{A}x \rangle$
  - Axiome $\mathcal{A}x$
  - Theoreme: $Thm(\mathcal{T}h) \stackrel{def}{=} \{t \mid \mathcal{T}h \vdash t\}$

Konservative Erweiterung

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

Konservative Erweiterung in Isabelle

- Isabelle: Konstruktion von Theorien durch konservative Erweiterungen
- Konstantendefinition (zur Signatur Σ):

$$c :: \sigma \qquad c \equiv t$$

  - $c \notin \Sigma$
  - $t \in T_\Sigma$ (enthält nicht c)
  - Typvariablen in t auch in σ
- Lemma: Konstantendefinition ist konservative Erweiterung
- Weitere konservative Erweiterungen:
  - Typdefinitionen, Datentypen.

Isabelle Theorien

- Strukturierung der Entwicklung in Theorien
- Kopf:

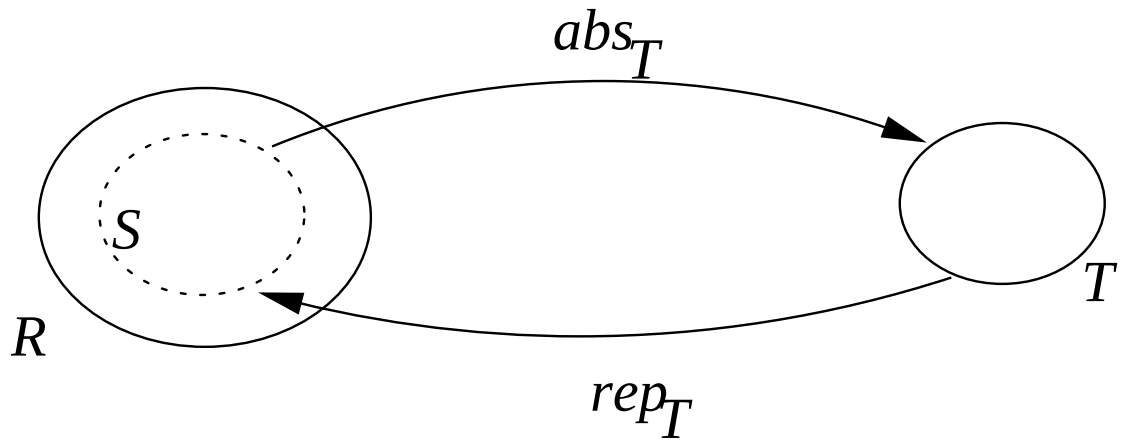
```
theory N
imports T1 T2 ... Tn
begin
```

- Theorien bilden DAG
- Theorieelemente (Auszug):
  - Typdeklaration: `typedecl t`
  - Typsynonym: `types t = S`
  - Konstantendeklaration: `consts f :: T`
  - Konstantendefinitionen `definition f :: T  "f == E"`
    - `E` darf `f` nicht enthalten, $FV(\texttt{E}) \subseteq FV(\texttt{f})$
  - Beweise: `lemma` oder `theorem`

Typdefinitionen in Isabelle

Definition eines neuen Typen:

- Gegeben Typ R, Prädikat $S : R \Rightarrow Bool$
- Neuer Typ T mit $abs_T : R \Rightarrow T, rep_T : T \Rightarrow R$ so dass
  - S und T isomorph: $\forall t. abs_T(rep_T\ t) = t, \forall r. S\ r \longrightarrow rep_T(abs_T\ r) = r$
  - T nicht leer: $\exists t. S\ t$



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

Beispiel: Produkte

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

Nützliche Isabelle/HOL-Typen

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

Natürliche Zahlen und unendliche Datentypen

- Natürliche Zahlen: nicht konservativ!
- Erfordert zusätzliches Axiom Unendlichkeit.
- Neuer Typ *ind* mit $Z :: ind, S : ind \Rightarrow S$ und

$$inj\ S \quad S\ x \neq Z$$

- Damit `nat` definierbar.
- Warum *ind*? Auch für andere Datentypen

Nützliche vordefinierte Typen

Theorie	Typ	Bedeutung
`Set`	`'a set`	getypte Mengen (`'a=> bool`)
`Sum`	`'a + 'b`	Summentyp
`Product_Type`	`'a * 'b`	Produktyp
`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`	Listen
`Map`	`'a ~=> 'b`	partielle Abbildungen (`'a => 'b option`)

* in `Complex_Main`

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

Algebraische Datentypen

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

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 "nat => 'a"
```

- Nicht erlaubt:

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

Labelled Records

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

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

  - Definiert:
    - Konstrukturen `(|x= 3, y= 4|)`
    - Selektoren `x :: point2d => int`
    - Update `p (|x := 3 |)`

- Erweiterbar:

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

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