Formale Methoden der Softwaretechnik 1
Vorlesung vom 19.10.09:
Einführung

Christoph Lüth, Lutz Schröder

WS 09/10

Organisatorisches

- Veranstalter:

Christoph Lüth	Lutz Schröder
christoph.lueth@dfki.de	lutz.schroeder@dfki.de
Cartesium 2.043, Tel. 64223	Cartesium 2.051, Tel. 64216

- Termine: Vorlesung: Montag, 12 – 14, MZH 7210
  Übung: Mittwoch, 12 – 14, MZH 7220

Therac-25

- Neuartiger Linearbeschleuniger in der Strahlentherapie.
  - Computergesteuert (PDP-11, Assembler)
- Fünf Unfälle mit Todesfolge (1985– 1987)
  - Zu hohe Strahlendosis (4000 – 20000 rad, letal 1000 rad)
- Problem: Softwarefehler
  - Ein einzelner Programmierer (fünf Jahre)
  - Alles in Assembler, kein Betriebssystem
  - Programmierer auch Tester (Qualitätskontrolle)

Ariane-5

Die Vasa

Lernziele

1. Modellierung — Formulierung von Spezifikationen
2. Formaler Beweis — Nachweis von Eigenschaften
3. Verifikation — Beweis der Korrektheit von Programmen

Darüber hinaus:

- Vertrautheit mit aktuellen Techniken

Themen

- Grundlagen:
  - Formale Logik, formales Beweisen
- Anwendung:
  - Der Theorembeweiser Isabelle
- Formale Spezifikation und Verifikation
  - Funktionale Programme
  - Imperative Programme

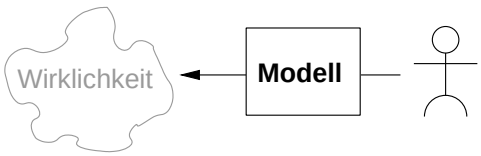
Plan

- Nächste sieben Wochen:
  - Formale Logik und formaler Beweis
  - Vorlesung: Grundlagen
  - Übung: Isabelle
- Nach Weihnachten:
  - Grundlagen der Verifikation imperativer Programme
  - Semantik, Hoare-Kalkül.

Der Theorembeweiser Isabelle

- Interaktiver Theorembeweiser
- Entwickelt in Cambridge und München
- Est. 1993 (?), ca. 500 Benutzer
- Andere: PVS, Coq, ACL-2
- Vielfältig benutzt:
  - VeriSoft (D) — `http://www.verisoft.de`
  - L4.verified (AUS) — `http://ertos.nicta.com.au/research/l4.verified/`
  - SAMS (Bremen) — `http://www.projekt-sams.de`

Das Problem



Formale Logik

- Formale (symbolische) Logik: Rechnen mit Symbolen
- Programme: Symbolmanipulation
- Auswertung: Beweis
- Curry-Howard-Isomorphie: funktionale Programme $\cong$ konstruktiver Beweis

Geschichte

- Gottlob Frege (1848– 1942)
  - 'Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens' (1879)
- Georg Cantor (1845– 1918), Bertrand Russel (1872– 1970), Ernst Zermelo (1871– 1953)
  - Einfache Mengenlehre: inkonsistent (Russel's Paradox)
  - Axiomatische Mengenlehre: Zermelo-Fränkel
- David Hilbert (1862– 1943)
  - Hilbert's Programm: 'mechanisierte' Beweistheorie
- Kurt Gödel (1906– 1978)
  - Vollständigkeitssatz, Unvollständigkeitssätze

Grundbegriffe der formalen Logik

- Ableitbarkeit $\mathcal{T}h \vdash P$
  - Syntaktische Folgerungung
- Gültigkeit $\mathcal{T}h \models P$
  - Semantische Folgerung — hier nicht relevant
- Klassische Logik: $P \vee \neg P$
- Entscheidbarkeit
  - Aussagenlogik
- Konsistenz: $\mathcal{T}h \not\vdash \bot$
  - Nicht alles ableitbar
- Vollständigkeit: jede gültige Aussage ableitbar
  - Prädikatenlogik erster Stufe

Unvollständigkeit

- Gödels 1. Unvollständigkeitssatz:
  - Jede Logik, die Peano-Arithmetik formalisiert, ist entweder inkonsistent oder unvollständig.
- Gödels 2. Unvollständigkeitssatz:
  - Jeder Logik, die ihre eigene Konsistenz beweist, ist inkonsistent.
- Auswirkungen:
  - Hilbert's Programm terminiert nicht.
  - Programme nicht vollständig spezifierbar.
  - Spezifikationssprachen immer unvollständig (oder uninteressant).
  - Mit anderen Worten: Es bleibt spannend.

Formale Methoden der Softwaretechnik 1
Vorlesung vom 26.10.09:
Formale Logik und natürliches Schließen

Christoph Lüth, Lutz Schröder

Universität Bremen

Wintersemester 2009/10

Heute

- Einführung in die formale Logik
- Aussagenlogik
  - Beispiel für eine einfache Logik
  - Guter Ausgangspunkt
- Natürliches Schließen
  - Wird auch von Isabelle verwendet.
- Buchempfehlung:
  Dirk van Dalen: Logic and Structure. Springer Verlag, 2004.

Fahrplan

- Teil I: Grundlagen der Formalen Logik
  - Einführung
  - Natürliches Schließen, Aussagenlogik
  - Prädikatenlogik 1. Stufe
  - Grundlagen von Isabelle
  - Logik höherer Ordnung
- Teil II: Arbeiten mit Isabelle
- Teil III: Modellierung imperative Programme

Formale Logik

- Ziel: Formalisierung von Folgerungen wie
  - Wenn es regnet, wird die Straße nass.
  - Es regnet.
  - Also ist die Straße nass.
  - Nachts ist es dunkel.
  - Es ist hell.
  - Also ist es nicht nachts.
- Eine Logik besteht aus
  - Einer Sprache $\mathcal{L}$ von Formeln (Aussagen)
  - Schlußregeln (Folgerungsregeln) auf diesen Formeln.
- Damit: Gültige ("wahre") Aussagen berechnen.

Beispiel für eine Logik I

- Sprache $\mathcal{L} = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$
- Schlußregeln:

$$\frac{\diamondsuit}{\clubsuit}\,\alpha \qquad \frac{\diamondsuit}{\spadesuit}\,\beta \qquad \frac{\clubsuit \quad \spadesuit}{\heartsuit}\,\gamma \qquad \frac{}{\diamondsuit}\,\delta$$

- Beispielableitung: $\heartsuit$

Beispiel für eine Logik II

- Sprache $\mathcal{L} = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$
- Schlußregeln:

$$\frac{\diamondsuit}{\clubsuit}\,\alpha \qquad \frac{\diamondsuit}{\spadesuit}\,\beta \qquad \frac{\clubsuit \quad \spadesuit}{\heartsuit}\,\gamma \qquad \frac{\begin{array}{c}[\diamondsuit]\\ \vdots \\ \heartsuit\end{array}}{\heartsuit}\,\delta'$$

- Beispielableitung: $\heartsuit$

Aussagenlogik

- Sprache $\mathcal{P}rop$ gegeben durch:
  1. Variablen $V \subseteq \mathcal{P}rop$ (Menge V gegeben)
  2. $\mathit{false} \in \mathcal{P}rop$
  3. Wenn $\phi, \psi \in \mathcal{P}rop$, dann
     - $\phi \wedge \psi \in \mathcal{P}rop$
     - $\phi \vee \psi \in \mathcal{P}rop$
     - $\phi \longrightarrow \psi \in \mathcal{P}rop$
     - $\phi \longleftrightarrow \psi \in \mathcal{P}rop$
  4. Wenn $\phi \in \mathcal{P}rop$, dann $\neg\phi \in \mathcal{P}rop$.

Wann ist eine Formel gültig?

- Semantische Gültigkeit $\models P$: Wahrheitstabellen etc.
  - Wird hier nicht weiter verfolgt.
- Syntaktische Gültigkeit $\vdash P$: formale Ableitung,
  - Natürliches Schließen
  - Sequenzenkalkül
  - Andere (Hilbert-Kalkül, gleichungsbasierte Kalküle, etc.)
- Ziel: Kalkül, um Gültigkeit in $\mathcal{P}rop$ zu beweisen

Natürliches Schließen

- Vorgehensweise:
  1. Erst Kalkül nur für $\wedge, \longrightarrow, \mathit{false}$
  2. Dann Erweiterung auf alle Konnektive.
- Für jedes Konnektiv: Einführungs- und Eliminitationsregel
- NB: konstruktiver Inhalt der meisten Regeln

Natürliches Schließen — Die Regeln

$$\frac{\phi \quad \psi}{\phi \wedge \psi}\,\wedge I \qquad \frac{\phi \wedge \psi}{\phi}\,\wedge E_L \qquad \frac{\phi \wedge \psi}{\psi}\,\wedge E_R$$

$$\frac{\begin{array}{c}[\phi]\\ \vdots\\ \psi\end{array}}{\phi \longrightarrow \psi}\,\longrightarrow I \qquad \frac{\phi \quad \phi \longrightarrow \psi}{\psi}\,\longrightarrow E$$

$$\frac{\mathit{false}}{\phi}\,\mathit{false} \qquad \frac{\begin{array}{c}[\phi \longrightarrow \mathit{false}]\\ \vdots\\ \mathit{false}\end{array}}{\phi}\,\mathrm{raa}$$

Konsistenz

- Def: Γ konsistent gdw. $\Gamma \nvdash \mathit{false}$
- Lemma: Folgende Aussagen sind äquivalent:

(i) Γ konsistent

(ii) Es gibt ein ϕ so dass $\Gamma \nvdash \phi$

(iii) Es gibt kein ϕ so dass $\Gamma \vdash \phi$ und $\Gamma \vdash \neg\phi$

- Satz: Aussagenlogik mit natürlichem Schließen ist konsistent.
- Satz: Aussagenlogik mit natürlichem Schließen ist vollständig und entscheidbar

Die fehlenden Konnektive

- Einführung als Abkürzung:

$$\neg\phi \stackrel{def}{=} \phi \longrightarrow \mathit{false}$$

$$\phi \vee \psi \stackrel{def}{=} \neg(\neg\phi \wedge \neg\psi)$$

$$\phi \longleftrightarrow \psi \stackrel{def}{=} (\phi \longrightarrow \psi) \wedge (\psi \longrightarrow \phi)$$

- Ableitungsregeln als Theoreme.

Die fehlenden Schlußregeln

$$\frac{\phi}{\phi \vee \psi}\,\vee I_L \qquad \frac{\psi}{\phi \vee \psi}\,\vee I_R$$

$$\frac{\phi \vee \psi \qquad \begin{array}{c}[\phi]\\ \vdots\\ \sigma\end{array} \qquad \begin{array}{c}[\psi]\\ \vdots\\ \sigma\end{array}}{\sigma}\,\vee E$$

$$\frac{\begin{array}{c}[\phi]\\ \vdots\\ \mathit{false}\end{array}}{\neg\phi}\,\neg I \qquad \frac{\phi \quad \neg\phi}{\mathit{false}}\,\neg E$$

$$\frac{\phi \longrightarrow \psi \quad \psi \longrightarrow \phi}{\phi \longleftrightarrow \psi}\,\longleftrightarrow I \qquad \frac{\phi \quad \phi \longleftrightarrow \psi}{\psi}\,\longleftrightarrow E_L \qquad \frac{\psi \quad \phi \longleftrightarrow \psi}{\phi}\,\longleftrightarrow E_R$$

Zusammenfassung

- Formale Logik formalisiert das (natürlichsprachliche) Schlußfolgern
- Logik: Aussagen plus Schlußregeln (Kalkül)
- Aussagenlogik: Aussagen mit $\wedge, \longrightarrow, \mathit{false}$
  - $\neg, \vee, \longleftrightarrow$ als abgeleitete Operatoren
- Natürliches Schließen: intuitiver Kalkül
- Aussagenlogik konsistent, vollständig, entscheidbar.
- Nächstes Mal: Quantoren, HOL.

Formale Methoden der Softwaretechnik 1
Vorlesung vom 02.11.09:
Prädikatenlogik erster Stufe

Christoph Lüth, Lutz Schröder

Universität Bremen

Wintersemester 2009/10

Das Tagesmenü

- Logik mit Quantoren
- Von Aussagenlogik zur Prädikatenlogik
- Natürliches Schließen mit Quantoren
- Die Notwendigkeit von Logik höherer Stufe

Fahrplan

- Teil I: Grundlagen der Formalen Logik
  - Einführung
  - Natürliches Schließen, Aussagenlogik
  - Prädikatenlogik 1. Stufe
  - Gleichungslogik und natürliche Zahlen
- Teil II: Arbeiten mit Isabelle
- Teil III: Modellierung imperative Programme

Prädikatenlogik

- Beschränkung der Aussagenlogik:
  - Eine Zahl n ist eine Primzahl genau dann wenn sie nicht 1 ist und nur durch 1 und sich selbst teilbar ist.
  - Eine Zahl m ist durch eine Zahl n teilbar genau dann wenn es eine Zahl p gibt, so dass $m = n \cdot p$.
  - Nicht in Aussagenlogk formalisierbar.
- Ziel: Formalisierung von Aussagen wie
  - Alle Zahlen sind ein Produkt von Primfaktoren.
  - Es gibt keine größte Primzahl.

Erweiterung der Sprache

- Terme beschreiben die zu formalisierenden Objekte.
- Formeln sind logische Aussagen.
- Unser Alphabet:
  - Prädikatensymbole: $P_1, \ldots, P_n, \doteq$ mit Arität $ar(P_i) \in \mathbb{N}$, $ar(\doteq) = 2$
  - Funktionssymbole: $f_1, \ldots, f_m$ mit Arität $ar(t_i) \in \mathbb{N}$
  - Menge X von Variablen (abzählbar viele)
  - Konnektive: $\wedge, \longrightarrow, \mathit{false}, \forall$, abgeleitet: $\vee, \longleftarrow, \neg, \longleftrightarrow, \exists$

Terme

- Menge $\mathcal{T}erm$ der Terme gegeben durch:
  - Variablen: $X \subseteq \mathcal{T}erm$
  - Funktionssymbol f mit $ar(f) = n$ und $t_1, \ldots, t_n \in \mathcal{T}erm$, dann $f(t_1, \ldots, t_n) \in \mathcal{T}erm$
  - Sonderfall: $n = 0$, dann ist f eine Konstante, $f \in \mathcal{T}erm$

Formeln

- Menge $\mathcal{F}orm$ der Formeln gegeben durch:
  - $\mathit{false} \in \mathcal{F}orm$
  - Wenn $\phi \in \mathcal{F}orm$, dann $\neg\phi \in \mathcal{F}orm$
  - Wenn $\phi, \psi \in \mathcal{F}orm$, dann $\phi \wedge \psi \in \mathcal{F}orm$, $\phi \vee \psi \in \mathcal{F}orm$, $\phi \longrightarrow \psi \in \mathcal{F}orm$, $\phi \longleftrightarrow \psi \in \mathcal{F}orm$
  - Wenn $\phi \in \mathcal{F}orm, x \in X$, dann $\forall x.\phi \in \mathcal{F}orm, \exists x.\phi \in \mathcal{F}orm$
  - Prädikatensymbol p mit $ar(p) = m$ und $t_1, \ldots, t_m \in \mathcal{T}erm$, dann $p(t_1, \ldots, t_m) \in \mathcal{F}orm$
    - Sonderfall: $t_1, t_2 \in \mathcal{T}erm$, dann $t_1 \doteq t_2 \in \mathcal{F}orm$

Beispielaussagen

- Alle Zahlen sind gerade oder ungerade.
- Keine Zahl ist gerade und ungerade.
- Es gibt keine größte Primzahl.
- Für jede Primzahl gibt es eine, die größer ist.
- Eine Funktion f ist stetig an der Stelle x_0, gdw. es für jedes $\varepsilon > 0$ ein $\delta > 0$ gibt, so dass für alle x mit $|x - x_0| < \delta$ gilt $|f(x) - f(x_0)| < \epsilon$.

Freie und gebundene Variable

- Variablen in $t \in \mathcal{T}erm, p \in \mathcal{F}orm$ sind frei, gebunden, oder bindend.
  - x bindend in $\forall x.\phi$, $\exists x.\psi$
  - Für $\forall x.\phi$ und $\exists x.\phi$ ist x in Teilformel ϕ gebunden
  - Ansonsten ist x frei
- $FV(\phi)$: Menge der freien Variablen in ϕ
- Beispiel:

$$(q(x) \vee \exists x.\forall y.p(f(x), z) \wedge q(a)) \vee \forall r(x, z, g(x))$$

Substitution

- $t\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right]$ ist Ersetzung von x durch s in t
- Definiert durch strukturelle Induktion:

$$\begin{array}{rcl}
y\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & \begin{cases} s & x = y \\ y & x \neq y \end{cases} \\
f(t_1, \ldots, t_n)\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & f(t_1\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right], \ldots, t_n\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right]) \\
\mathit{false}\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & \mathit{false} \\
(\phi \wedge \psi)\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & \phi\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] \wedge \psi\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] \\
(\phi \longrightarrow \psi)\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & \phi\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] \longrightarrow \psi\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] \\
p(t_1, \ldots, t_n)\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & p(t_1\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right], \ldots, t_n\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right]) \\
(\forall y.\phi)\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right] & \stackrel{def}{\equiv} & \begin{cases} \forall y.\phi & x = y \\ \forall y.(\phi\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right]) & x \neq y, y \notin FV(s) \\ \forall z.((\phi\left[\begin{smallmatrix} z \\ y \end{smallmatrix}\right])\left[\begin{smallmatrix} s \\ x \end{smallmatrix}\right]) & x \neq y, y \in FV(s) \\ & \text{mit } z \notin FV(s) \text{ (} z \text{ frisch)} \end{cases}
\end{array}$$

Natürliches Schließen mit Quantoren

$$\frac{\phi}{\forall x.\phi}\ \forall I\ (*) \qquad \frac{\forall x.\phi}{\phi\left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]}\ \forall E\ (\dagger)$$

- (*) Eigenvariablenbedingung:
  x nicht frei in offenen Vorbedingungen von ϕ (x beliebig)
- (†) Ggf. Umbenennung durch Substitution
- Gegenbeispiele für verletzte Seitenbedingungen

Der Existenzquantor

$$\exists x.\phi \stackrel{def}{=} \neg\forall x.\neg\phi$$

$$\frac{\phi\left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]}{\exists x.\phi}\ \exists I\ (\dagger) \qquad \frac{\exists x.\phi \quad \begin{array}{c}[\phi] \\ \vdots \\ \psi\end{array}}{\psi}\ \exists E\ (*)$$

- (*) Eigenvariablenbedingung:
  x nicht frei in ψ, oder einer offenenen Vorbedingung außer ϕ
- (†) Ggf. Umbenennung durch Substitution

Zusammenfassung

- Prädikatenlogik: Erweiterung der Aussagenlogik um
  - Konstanten- und Prädikatensymbole
  - Gleichheit
  - Quantoren
- Das natürliche Schließen mit Quantoren
  - Variablenbindungen — Umbenennungen bei Substitution
  - Eigenvariablenbedingung
- Das nächste Mal: Gleichungen und natürliche Zahlen

Formale Methoden der Softwaretechnik 1
Vorlesung vom 11.11.09:
Gleichungslogik und natürliche Zahlen

Christoph Lüth, Lutz Schröder

Universität Bremen

Wintersemester 2009/10

Das Tagesmenü

- Gleichheit in Logik 1. Stufe
- Die natürlichen Zahlen
- Eigenschaften der natürlichen Zahlen

Fahrplan

- Teil I: Grundlagen der Formalen Logik
  - Einführung
  - Natürliches Schließen, Aussagenlogik
  - Prädikatenlogik 1. Stufe
  - Gleichungslogik und natürliche Zahlen
- Teil II: Arbeiten mit Isabelle
- Teil III: Modellierung imperative Programme

Regeln für die Gleichheit

- Reflexivität, Symmetrie, Transitivität:

$$\frac{}{x = x}\ \text{refl} \qquad \frac{x = y}{y = x}\ \text{sym} \qquad \frac{x = y \quad y = z}{x = z}\ \text{trans}$$

- Kongruenz:

$$\frac{x_1 = y_1, \ldots, x_n = y_n}{f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)}\ \text{conf}$$

- Substitutivität:

$$\frac{x_1 = y_1, \ldots, x_m = y_m \quad P(x_1, \ldots, x_n)}{P(y_1, \ldots, y_m)}\ \text{subst}$$

Die natürlichen Zahlen

- Verschiedene Axiomatisierungen:
- Presburger-Arithmetik
  - 5 Axiome
  - Konsistent und vollständig
  - Entscheidbar (Aufwand $2^{2^{cn}}$, n Länge der Aussage)
  - Enthält Nichtstandardmodelle
- Peano-Arithmetik
  - 8 Axiome
  - Konsistent
  - Unvollständig (bzgl. Standard-Modellen)
  - Nicht entscheidbar

Zusammenfassung

- Gleichungslogik in natürlichem Schließen:
  - möglich
  - aber umständlich
- Entwicklung natürlicher Zahlen benötigt:
  - Zusätzliche Axiome
  - Konzepte höherer Ordnung (Induktion!)
- Deshalb nächstes Mal: Logik höherer Stufe

Formale Methoden der Softwaretechnik 1
Vorlesung vom 16.11.09:
Grundlage von Isabelle

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
  - Isabelle/HOL
  - Beweise über funktionale Programme in Isabelle/HOL
  - Beweisen mit Isabelle: Simplifikation, Taktiken
- Teil III: Modellierung imperative Programme

Fahrplan

- Grundlagen:
  - der getypte λ-Kalkül
  - Unifikation und Resolution
- Beweisen in Isabelle

Der λ-Kalkül

- In den 30'ern von Alonzo Church als theoretisches Berechhnungsmodell erfunden.
- In den 60'ern Basis von Lisp (John McCarthy)
- Mathematische Basis von funktionalen Sprachen (Haskell etc.)
- Hier: Grundlage der Syntax ("higher-order abstract syntax")
  - Typisierung
  - Gebundene Variablen
  - Warum?

Der polymorph getypte λ-Kalkül

- Typen $\mathcal{T}ype$ gegeben durch
  - Typkonstanten: $c \in \mathcal{C}_{\mathcal{T}ype}$
  - Typvariablen: $\alpha \in \mathcal{V}_{\mathcal{T}ype}$ (Menge $\mathcal{V}_{\mathcal{T}ype}$ gegeben)
  - Funktionen: $s, t \in \mathcal{T}ype$ dann $s \Rightarrow t$ in $\mathcal{T}ype$
- Terme $\mathcal{T}erm$ gegeben durch
  - Konstanten: $c \in \mathcal{C}$
  - Variablen: $v \in \mathcal{V}$
  - Applikation: $s, t \in \mathcal{T}erm$ dann $s\ t \in \mathcal{T}erm$
  - Abstraktion: $x \in \mathcal{V}, \tau \in \mathcal{T}ype, t \in \mathcal{T}erm$ dann $\lambda x^{\tau}.t \in \mathcal{T}erm$
    - Typ τ kann manchmal berechnet werden.
- Signatur enthält $\mathcal{C}_{\mathcal{T}ype}$, $\mathcal{C}$

Typisierung

- Signatur: $\Sigma = \langle \mathcal{C}_{\mathcal{T}ype}, \mathcal{C}, ar \rangle, ar : \mathcal{C} \to \mathcal{T}ype$
  - Typisierung der Konstanten
  - Notation: $\Sigma = \{c_1 : \tau_1, \ldots, c_n : \tau_n\}$ für $ar(c_i) = \tau_i$
- Term t hat Typ τ in einem Kontext Γ und einer Signatur Σ:

$$\Gamma \vdash_{\Sigma} t : \tau$$

- Kontext: $x_1 : \tau_1, \ldots, x_n : \tau_n$ $(x_i \in \mathcal{V})$
  - Typisierung von Variablen
- Vergleiche Haskell etc.

Typisierung

- Die Regeln:

$$\frac{c : \tau \in \Sigma}{\Gamma \vdash_{\Sigma} c : \tau}\ CONST$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_{\Sigma} x : \tau}\ VAR$$

$$\frac{\Gamma \vdash_{\Sigma} s : \sigma \Rightarrow \tau \quad \Gamma \vdash_{\Sigma} t : \sigma}{\Gamma \vdash_{\Sigma} s\ t : \tau}\ APP$$

$$\frac{\Gamma, x : \sigma \vdash_{\Sigma} t : \tau}{\Gamma \vdash_{\Sigma} \lambda x^{\sigma}.t : \sigma \Rightarrow \tau}\ ABST$$

Einbettungen

- Beispiel: Einbettung der Prädikatenlogik mit PA
- Basistypen:

$$\mathcal{C}_{\mathcal{T}ype} = \{i, o\}$$

- Konstanten:

$$\begin{array}{rcl} \Sigma_T & = & \{0 : i, s : i \Rightarrow i, plus : i \Rightarrow i \Rightarrow i\} \\ \Sigma_A & = & \{eq : i \Rightarrow i \Rightarrow o, false : o, and : o \Rightarrow o \Rightarrow o, \ldots, \\ & & all : (i \Rightarrow o) \Rightarrow o, ex : (i \Rightarrow o) \Rightarrow o\} \\ \Sigma & = & \Sigma_T \cup \Sigma_A \end{array}$$

- $\phi \in \mathcal{F}orm \longleftrightarrow \vdash_{\Sigma} t : o$
- Beispiel: $\forall x.\exists y.(x = y) \longleftrightarrow all\ (\lambda x.ex\ (\lambda y.(eq\ x)\ y))$

Substitution

- $s\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right]$ ist Ersetzung von x durch t in s
- Definiert durch strukturelle Induktion:

$$
\begin{aligned}
c\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] &\stackrel{def}{=} c \\
y\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] &\stackrel{def}{=} \begin{cases} t & x = y \\ y & x \neq y \end{cases} \\
(r\ s)\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] &\stackrel{def}{=} r\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] s\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] \\
(\lambda y.s)\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] &\stackrel{def}{=} \begin{cases} \lambda y.s & x = y \\ \lambda y.s\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] & x \neq y, y \notin FV(t) \\ \lambda z.(s\left[\begin{smallmatrix}z\\y\end{smallmatrix}\right])\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] & x \neq y, y \in FV(t) \\ & \text{mit } z \notin FV(t)\ (z \text{ frisch}) \end{cases}
\end{aligned}
$$

Reduktion und Äquivalenzen

- β-Reduktion: $(\lambda x.s)t \rightarrow_\beta s\left[\begin{smallmatrix}t\\x\end{smallmatrix}\right]$
- β-Äquivalenz: $=_\beta \stackrel{def}{=} (\rightarrow_\beta^* \cup {}_\beta^* \leftarrow)^*$
  - $s =_\beta t$ iff. $\exists u. s \rightarrow_\beta^* u, t \rightarrow_\beta^* u$ (Church-Rosser)
- α-Äquivalenz: $\lambda x.t =_\alpha \lambda y.t\left[\begin{smallmatrix}y\\x\end{smallmatrix}\right]$, $y \notin FV(t)$
  - Name von gebundenen Variablen unerheblich
  - In Isabelle Implizit (deBruijn-Indizes)
- η-Äquivalenz: $\lambda x.tx =_\eta t$, $x \notin FV(t)$
  - "punktfreie" Notation

Unifikation

- Für $s, t \in \mathcal{T}erm$ oder $s, t \in \mathcal{T}ype$ ist Substitution σ Unifikator wenn $s\sigma = t\sigma$
  - Allgemeinster Unifikator τ: alle anderen Unifikatoren sind Instanzen von τ
  - Allgemeinster Unifikator eindeutig, wenn er existiert
- Unifikationsalgorithmus:

$$
\begin{aligned}
\tau(f\ t, g\ s) &= \mathit{false} \\
\tau(f\ t, f\ s) &= \tau(t, s) \\
\tau(t, x) &= \left[\begin{smallmatrix}t\\x\end{smallmatrix}\right] \quad x \notin FVt \\
\tau(y, s) &= \left[\begin{smallmatrix}y\\x\end{smallmatrix}\right] \quad y \notin FVs
\end{aligned}
$$

- Matching: einseitige Unifikation (σ mit $s\sigma = t$)

Grundlagen von Isabelle

- Grundlage: getypter Λ-Kalkül
- Unifikation und Matching (`apply (rule r)`)
- Äquivalenzen:
  - β-Reduktion automatisch
  - α-Äquivalenz eingebaut (deBruijn-Indexing)
  - η-Äquivalenz automatisch
- Variablen, Formeln, Resolution

Variablen

- Meta-Variablen: können unifiziert und beliebig instantiiert werden
- freie Variablen (fixed): beliebig aber fest
- Gebundene Variablen: Name beliebig (α-Äquivalenz)
- Meta-Quantoren: Isabelles Eigenvariablen

$$\frac{\bigwedge x.P(x)}{\forall x.P(x)}\ \mathit{allI} \qquad \texttt{!!x. P x ==> ALL x. P x}$$

  - Beliebig instantiierbar
  - Gültigkeit auf diese (Teil)-Formel begrenzt

Formeln

- Formeln in Isabelle:

$$\frac{\phi_1, \ldots, \phi_n}{\psi} \qquad [\![\phi_1, \ldots, \phi_n]\!] \Longrightarrow \psi$$

  - $\phi_1, \ldots, \phi_n$ Formeln, ψ atomar
  - Theoreme: ableitbare Formeln
  - Ableitung von Formeln: Resolution, Instantiierung, Gleichheit
- Randbemerkung:
  - $\Longrightarrow$, $\bigwedge$, $\equiv$ formen Meta-Logik
  - Einbettung anderer Logiken als HOL möglich — generischer Theorembeweiser

Resolution

- Einfache Resolution (`rule`)
  - Achtung: Lifting von Meta-Quantoren und Bedingungen
  - Randbemerkung: Unifikation höherer Stufe unentscheidbar
- Eliminationsresolution (`erule`)
- Destruktionsresolution (`drule`)

Rückwärtsbeweis

- Ausgehend von Beweisziel ψ
- Beweiszustand ist $[\![\phi_1, \ldots, \phi_n]\!] \Longrightarrow \psi$
- $\phi_1, \ldots, \phi_n$: subgoals
- Beweisverfahren: Resolution, Termersetzung, Beweissuche

Zusammenfassung

- Getypter λ-Kalkül als Grundlage der Metalogik, Modellierung von Logiken durch Einbettung
- Isabelle: Korrektheit durch Systemarchitektur
- Beweis: Rückwärts, Resolution
  - Anmerkung: auch Vorwärtsbeweis möglich
- Donnerstag: Logik höherer Ordnung in Isabelle

Formale Methoden der Softwaretechnik 1
Vorlesung vom 18.11.09:
Logik Höherer Stufe in Isabelle

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

Fahrplan

- Alles über Logik höherer Stufe (Higher-order Logic, HOL):
  - Typen und Terme
  - Die Basis-Axiome
  - Definierte Operatoren

Logik höherer Stufe

- Ziel: Formalisierung von Mathematik
  - "Logik für Erwachsene"
- Problem: Mögliche Inkonsistenz (Russel's Paradox)
- Lösung: Restriktion vs. Ausdrucksstärke
- Alternative Grundlagen:
  - Andere Typtheorien (Martin-Löf, Calculus of Constructions)
  - Ungetypte Mengenlehre (ZFC)
- HOL: guter Kompromiss, weit verbreitet.
  - Klassische Logik höherer Stufe nach Church
  - Schwächer als ZFC, stärker als Typtheorien

Warum Logik höherer Stufe?

- Aussagenlogik: keine Quantoren
- Logik 1. Stufe: Quantoren über Terme

$$\forall x\, y.\, x = y \longrightarrow y = x$$

- Logik 2. Stufe: Quantoren über Prädikaten und Funktionen

$$\forall P.(P\,0 \wedge \forall x.P\,x \longrightarrow P\,(S\,x)) \longrightarrow \forall x.P\,x$$

- Logik 3. Stufe: Quantoren über Argumenten von Prädikaten
- Logik höherer Stufe (HOL): alle endlichen Quantoren
  - Keine wesentlichen Vorteile von Logik 2. Ordnung

Vermeidung von Inkonsistenzen

- Russell's Paradox
  - $R = \{X \mid X \notin X\}$
  - Abhilfe: Typen
- Gödel's 2. Unvollständigkeitssatz:
  - Jede Logik, die ihre eigene Konsistenz beweist, ist inkonsistent.
- Unterscheidung zwischen Termen und Aussagen
  - Dadurch **in** HOL keine Aussage **über** HOL

Typen

- Typen $\mathcal{T}ype$ gegeben durch
  - Typkonstanten: $c \in \mathcal{C}_{\mathcal{T}ype}$ (Menge $\mathcal{C}_{\mathcal{T}ype}$ durch Signatur gegeben)
    - $Prop, Bool \in \mathcal{C}_{\mathcal{T}ype}$: *Prop* alle Terme, *Bool* alle Aussagen
  - Typvariablen: $\alpha \in \mathcal{V}_{\mathcal{T}ype}$ (Menge $\mathcal{V}_{\mathcal{T}ype}$ fest)
  - Funktionen: $s, t \in \mathcal{T}ype$ dann $s \Rightarrow t$ in $\mathcal{T}ype$
- Konvention: Funktionsraum nach rechts geklammert
  $\alpha \Rightarrow \beta \Rightarrow \gamma$ für $\alpha \Rightarrow (\beta \Rightarrow \gamma)$

Terme

- Terme $\mathcal{T}erm$ gegeben durch
  - Konstanten: $c \in \mathcal{C}$ (Menge $\mathcal{C}$ durch Signatur gegeben)
  - Variablen: $v \in \mathcal{V}$
  - Applikation: $s, t \in \mathcal{T}erm$ dann $s\,t \in \mathcal{T}erm$
  - Abstraktion: $x \in \mathcal{V}, t \in \mathcal{T}erm$ dann $\lambda x.\, t \in \mathcal{T}erm$
- Konventionen: Applikation links geklammert, mehrfache Abstraktion
  $\lambda x\, y\, z.\, f\, x\, y\, z$ für $\lambda x.\, \lambda y.\, \lambda z.\, ((f\, x)\, y)\, z$

Basis-Syntax

$= \quad :: \alpha \Rightarrow \alpha \Rightarrow \mathit{Bool}$
$\longrightarrow :: \mathit{Bool} \Rightarrow \mathit{Bool} \Rightarrow \mathit{Bool}$
$\iota \quad :: (\alpha \Rightarrow \mathit{Bool}) \Rightarrow \alpha$

$\neg \quad :: \mathit{Bool} \Rightarrow \mathit{Bool}$
$\mathit{true} :: \mathit{Bool}$
$\mathit{false} :: \mathit{Bool}$
$\mathit{if} \quad :: \mathit{Bool} \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha$
$\forall \quad :: (\alpha \Rightarrow \mathit{Bool}) \Rightarrow \mathit{Bool}$
$\exists \quad :: (\alpha \Rightarrow \mathit{Bool}) \Rightarrow \mathit{Bool}$
$\wedge \quad :: \mathit{Bool} \Rightarrow \mathit{Bool} \Rightarrow \mathit{Bool}$
$\vee \quad :: \mathit{Bool} \Rightarrow \mathit{Bool} \Rightarrow \mathit{Bool}$

- Einbettung (wird weggelassen)
  $\mathit{trueprop} :: \mathit{Bool} \Rightarrow \mathit{Prop}$
- Basis-Operatoren: $=, \longrightarrow, \iota$
- Syntaktische Konventionen:
  - Bindende Operatoren: $\forall$, $\exists$, ι
    $\forall x.P \equiv \forall(\lambda x.P)$
  - Infix-Operatoren: $\wedge, \vee, \longrightarrow, =$
  - Mixfix-Operator:
    $\mathit{if}\ b\ \mathit{then}\ p\ \mathit{else}\ q \equiv \mathit{if}\ b\ p\ q$

Basis-Axiome I: Gleichheit

- Reflexivität:
$$\frac{}{t = t}\ \text{refl}$$
- Substitutivität:
$$\frac{s = t \quad P(s)}{P(t)}\ \text{subst}$$
- Extensionalität:
$$\frac{\forall x.fx = gx}{(\lambda x.fx) = (\lambda x.gx)}\ \text{ext}$$
- Einführungsregel:
$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

Basis-Axiome II: Implikation und Auswahl

- Einführungsregel Implikation:
$$\frac{\begin{array}{c}[P] \\ \vdots \\ Q\end{array}}{P \longrightarrow Q}\ \text{impl}$$
- Eliminationsregel Implikation:
$$\frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$
- Eliminationsregel Auswahloperator:
$$\frac{}{(\iota x.x = a) = a}\ \text{the\_eq}$$
- HOL ist klassisch:
$$\frac{}{(P = \mathit{true}) \vee (P = \mathit{false})}\ \text{true\_or\_false}$$

Die Basis-Axiome (Isabelle-Syntax)

refl : $t = t$
subst : $[\![s = t; P(s)]\!] \Longrightarrow P(t)$
ext : $[\![\bigwedge x.fx = gx]\!] \Longrightarrow (\lambda x.fx) = (\lambda x.gx)$
impl : $[\![P \Longrightarrow Q]\!] \Longrightarrow P \longrightarrow Q$
mp : $[\![P \longrightarrow Q; P]\!] \Longrightarrow Q$
iff : $(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)$
the_eq : $(\iota x.x = a) = a$
true_or_false : $(P = \mathit{true}) \vee (P = \mathit{false})$

Abgeleitete Operatoren

$$\begin{aligned}
\mathit{true} &\equiv (\lambda x.x) = (\lambda x.x) \\
\forall P &\equiv (P = \lambda x.\mathit{true}) \\
\exists P &\equiv \forall Q.(\forall x.Px \longrightarrow Q) \longrightarrow Q \\
\mathit{false} &\equiv \forall P.P \\
\neg P &\equiv P \longrightarrow \mathit{false} \\
P \wedge Q &\equiv \forall R.(P \longrightarrow Q \longrightarrow R) \longrightarrow R \\
P \vee Q &\equiv \forall R.(P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R \\
\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y &\equiv \iota z.(P = \mathit{true} \longrightarrow z = x) \wedge (P = \mathit{false} \longrightarrow z = y)
\end{aligned}$$

Erweiterungen

- Weitere Operatoren
- Weitere Typen: natürliche Zahlen, Datentypen
- Axiomatisch (vgl. Peano/Presburger in FOL)
  - Mögliche Inkonsistenzen
- Konserverative Erweiterung
  - Logik konsistentzerhaltend erweitern

Zusammenfassung

Logik höherer Stufe (HOL):

- Syntax basiert auf dem einfach getypten λ-Kalkül
- Drei Basis-Operatoren, acht Basis-Axiome
- Rest folgt durch konservative Erweiterung — nächstes Mal

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
- 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: $Thm(\mathcal{T}h) \stackrel{def}{=} \{t \mid \mathcal{T}h \vdash t\}$
- 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:

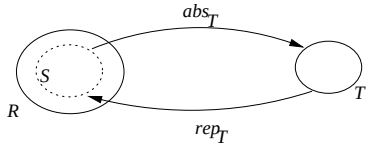
```
theory N
imports T1 T2 ... Tn
begin
```

- 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, $FV(\mathtt{E}) \subseteq FV(\mathtt{f})$
  - 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 $R_{\alpha,\beta} \equiv \alpha \Rightarrow \beta \Rightarrow Bool$
- Neuer Typ $T_{\alpha,\beta} \equiv \alpha \times \beta$
  - Idee: Prädikat $p : \alpha \Rightarrow \beta \Rightarrow Bool$ repräsentiert (a, b)
    $p(x, y) = true \longleftrightarrow x = a \wedge y = b$
- $S\ f = \exists a\ b.(f = \lambda x\ y. x = a \wedge y = b)$
- Abstraktionsfunktion:
  $abs_\times\ p = \iota a\ b.\ p\ a\ b$
- Repräsentationsfunktion:
  $rep_\times(a, b) = p$ mit $p(x, y) = true \longleftrightarrow x = a \wedge y = b$
- 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

Formale Methoden der Softwaretechnik 1
Vorlesung vom 02.12.09:
Beweisprozeduren und Funktionsdefinitionen

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

Überblick

- Automatische Beweisprozeduren im Überblick
- Definition von Funktionen

Simplifikation

- Simplifikation ist Termersetzung:
  - Gegeben Theorem $s = t$, ersetze s durch t.
- Benutzung: `apply (simp)`
- Nutzt Gleichungen und Ungleichungen:
  - Funktionsdefinitionen
  - Vereinfachungsregeln für Datentypen
  - Deklarierte Theoreme
  - Annahmen des lokalen Subgoals
- Benutzt bedingte Gleichungen: $s_1 = t_1, \ldots, s_n = t_n \Longrightarrow s = t$
  - Ersetzt s durch t, wenn Gleichungen $s_1 = t_1 \ldots s_n = t_n$ rekursiv gezeigt werden können.
- Instantiiert keine Meta-Variablen
- Erzeugt keine neuen Subgoals, nur Vereinfachung

Klassische Beweiser

- Beweisplaner: `blast`
  - Konstruiert Beweis durch Suche
  - Gelingt oder schlägt fehl: keine neuen Subgoals
- Klassischer Beweiser: `clarify`
  - Wendet Einführungs- und Eliminationsregeln systematisch an
  - Keine neuen Subgoals, nur Vereinfachung
  - Sicher: keine unbeweisbaren Subgoals
  - `clarsimp`: Kombination mit Simplifikation
- Vollautomatisch: `auto`
  - Kombination verschiedener Beweiser
  - Instantiiert `Meta-Variablen`, erzeugt neue Subgoals
  - Unsicher: kann unbeweisbare Subgoals erzeugen

Definition von Funktionen

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

Einfache Rekursion

fun $f :: \tau$

where

equations

Volle Rekursion

function $f :: \tau$

where

equations

by *completeness-proof*

termination by *termination-proof*

Zusammenfassung

- Automatische Beweisprozeduren: `simp`, `blast`, `clarify`, `auto`
- Funktionsdefinition: `fun` und `function`
  - Fast wie in Haskell, aber immer total.