Interaktives Beweisen

Gert Smolka
Universität des Saarlandes
Perspektiven der Informatik
9. Januar 2012

Beweisassistenten

- Unterstützen die mathematische Formulierung von Theorien
  - Definitionen
  - Aussagen
  - Beweise
- Garantieren Wohlgeformtheit und Korrektheit
- Grundlage ist ein logischer Kalkül
- Coq und Isabelle sind vielbenutzte Beweisassistenten
- Für diesen Vortrag benutzen wir Coq

Coq

- Wird seit 1985 von INRIA in Frankreich entwickelt
- Wird bei uns in Projekten und den Vorlesungen Introduction to Computational Logic und Semantics verwendet
- Die zugrundeliegende Logik ist der Calculus of Inductive Constructions
- Coq ist mit der Programmiersprache OCAML realisiert
- ML wurde 1974 von Robin Milner für den Beweisassistenten LCF entwickelt

Grundlegende Fragen

- Was sind Definitionen?
- Was sind Aussagen?
- Was sind Beweise?

- Wie sieht eine Sprachen aus, mit der diese Objekte beschrieben und verarbeitet werden können?

- Was sind die natürlichen Zahlen?

- Diese Themen gehören zum Wissensgebiet der Logik, das seit 50 Jahren als Teilgebiet der Informatik weiter entwickelt wird

Dedekind-Peano Axiome

- Axiomatisierung der natürlichen Zahlen, 1889
- Hauptererkenntnis: Die natürlichen Zahlen sind ein Konstruktortyp mit den Konstruktoren

  O : nat
  S : nat → nat

- Alle natürlichen Zahlen können mit O und S beschrieben werden:

  O, S O, S (S O), . . .

- O und S sind disjunkt
- S ist injektiv
- Induktionsaxiom

- Arithmetische Operationen können rekursiv definiert werden

Coq Demo

- Definiere Konstruktortyp nat
- Definiere Addition als rekursive Prozedur

$$O + y = y$$
$$S\,x + y = S(x+y)$$

- Beweise Kommutativität der Addition mit Induktion

$$x + y = y + x$$

- Dazu sind 2 Lemmata erforderlich
  - $x + O = x$
  - $x + Sy = S(x + y)$

```
Inductive nat :=
| O : nat
| S : nat -> nat.

Fixpoint add (x y : nat) : nat :=
match x with
| O => y
| S x' => S (add x' y)
end.

Lemma Add1 :
add O (S O) = add (S O) O.

Proof.
simpl. reflexivity.
Qed.

Lemma AddO :
forall x : nat,
add x O = x.

Proof.
induction x.
simpl. reflexivity.
simpl. rewrite IHx. reflexivity.
Qed.
```

```
Lemma AddS :
forall x y : nat,
add x (S y) = S (add x y).

Proof.
intros x y.
induction x.
simpl. reflexivity.
simpl. rewrite IHx. reflexivity.
Qed.

Lemma AddCom :
forall x y : nat,
add x y = add y x.

Proof.
intros x y.
induction x.
simpl. rewrite AddO. reflexivity.
simpl. rewrite IHx. rewrite AddS. reflexivity.
Qed.
```

Implikation $A \rightarrow B$

- Wenn A, dann B
- A impliziert B
- Wenn A beweisbar ist, dann ist B beweisbar
- Ein Beweis einer Implikation $A \rightarrow B$ ist eine Funktion, die zu jedem Beweis von A einen Beweis von B liefert
- Brouwer-Heyting-Kolmogorov-Interpretation

Falschkeit $\bot$ und Negation $\neg$

- Falschheit $\bot$ ist eine Aussage, die keinen Beweis hat
- Negation ist definiert

  $\neg A := A \to \bot$

  Wenn $\neg A$ beweisbar ist,
  dann gibt es keinen Beweis für A

- Deduktion unter Annahmen

Explosionsregel für Negation

- Aus einem Beweis für Falschkeit kann man einen Beweis für jede Aussage erhalten

$$\frac{\perp}{A}$$

Quantifizierung $\forall x{:}X.\ A$

- Für alle x in X gilt Ax
- Ein Beweis von $\forall x{:}X.A$ ist eine Funktion, die zu jedem Wert x:X einen Beweis von Ax liefert

Regeln des natürlichen Schließens
Gentzen 1935

- Ein Beweis folgert eine Aussage aus $n \geq 0$ Aussagen

$$A_1, \ldots, A_n \Rightarrow A$$

- Annahmen $\Rightarrow$ Behauptung
- Separate Regeln für jede logische Operation
- Regeln für Implikation

$$\frac{A \Rightarrow B}{A \rightarrow B} \qquad\qquad \frac{A \rightarrow B \quad A}{B}$$

Coq Demo

- Drei logische Gesetze
  - Modus Ponens
  - De Morgan
  - Russel (Barbierformulierung)

```
Lemma ModusPonens :
forall X Y : Prop,
((X -> Y) /\ X) -> Y.

Proof.
intros X Y.
intros A.
destruct A as [f x].
apply f.
apply x.
Qed.

Lemma DeMorgan :
forall X Y : Prop,
(~X \/ ~Y) -> ~(X /\ Y).

Proof.
intros X Y.
cbv.
intros A.
intros B.
destruct B as [x y].
destruct A as [A1|A2].
apply A1.
apply x.
apply (A2 y).
Qed.
```

```
Lemma Hilf :
forall X : Prop, ~(X <-> ~X).

Proof.
intros X.
red. intros A. red in A. destruct A as [f g]. red in f.
cut X.
intros x. apply (f x x).
apply g. red. intros x. apply (f x x).
Qed.

Lemma Barbier :
forall M : Type, forall r : M -> M -> Prop,
~ exists b : M, forall x : M,
r b x <-> ~ r x x.

Proof.
intros M r.
red.
intros A.
destruct A as [b B].
specialize (B b).
apply (Hilf (r b b)). apply B.
Qed.
```

Konstruktive Logik

- Unterscheidet zwischen Beweisregeln (ND) und zusätzlichen mathematischen Annahmen
- Eine klassische Annahme, die nicht aus den Beweisregel folgt, ist tertium non datur oder excluded middle:

$$\forall A{:}\mathrm{Prop}.\ \ A \lor \neg A$$

- Klassische Logik: 2 Wahrheitswerte

Excluded Middle

Die folgenden Aussagen sind beweisbar äquivalent

- $\forall$ A:Prop. $A \vee \neg A$ — Excluded Middle
- $\forall$ A:Prop. $\neg\neg A \rightarrow A$ — Double Negation
- $\forall$ A B : Prop. $((A \rightarrow B) \rightarrow A) \rightarrow A$ — Peirce‘s Law

Gödelscher Unvollständigkeitssatz

In jedem ausdrucksstarken Beweissystem

gibt es Aussagen A,

für die weder A noch ¬A beweisbar ist

Logische Operationen als Funktionen

$\neg : \mathrm{Prop} \to \mathrm{Prop}$

$\wedge : \mathrm{Prop} \to \mathrm{Prop} \to \mathrm{Prop}$

$\exists : \forall X{:}\mathrm{Type}.\ (X \to \mathrm{Prop}) \to \mathrm{Prop}$

Konjunktionen und Disjunktionen als Konstruktortypen

and : Prop → Prop → Prop

And : ∀ A B : Prop. A → B → and A B

or : Prop → Prop → Prop

OrLeft : ∀ A B : Prop. A → or A B

OrRight : ∀ A B : Prop. B → or A B

Induktionsregel als Lemma

- Induktionsregel

$$\frac{pO \quad \forall x.px \to p(Sx)}{\forall x.px}$$

- Realisierung durch das Lemma

$$\forall p.pO \to (\forall x.px \to p(Sx)) \to \forall x.px$$

Coq Demo

Beweis des Induktionslemmas mithilfe von Rekursion

```
Lemma Induction :
forall p : nat -> Prop,
p O -> (forall x, p x -> p (S x)) -> forall x, p x.

Proof.
intros p base step.
fix f 1.
intros x.
destruct x.
apply base.
apply step. apply f.
Qed.
```

Typtheorie

- Funktionale Programmiersprache mit sehr ausdrucksstarkem Typsystem (Typen als Werte)
- Aussagen werden als Typen dargestellt, z.B. Implikation als Funktionstyp
- Beweise werden als Ausdrücke dargestellt
- a:A bedeutet, das a ein Beweis der Aussage A ist
- Beweisprüfung als Typprüfung
- In Coq werden Beweise durch Skripte synthetisiert
- Wichtige Beiträge kamen von
  B. Russell (1903), A. Church (1940),
  P. Martin-Löf (1971), J.Y. Girard (1972), Curry 1958,
  N.G. de Bruijn 1980, Coquand und Huet (1985)

Wenn Sie mehr wissen wollen

- Webseite der Vorlesung ICL
  Introduction to Computational Logic
  www.ps.uni-saarland.de/courses/cl-ss11
  unter Gert Smolka / Teaching
- Webseiten des Coq Systems: coq.inria.fr