Mathematik und Logik

Franz Binder

Institut für Algebra
Johannes Kepler Universität Linz

Vorlesung im 2005W
`http://www.algebra.uni-linz.ac.at/Students/Win/ml05w`

Inhalt

Implikation, $\Rightarrow$

- Schlußregeln:

$$\dfrac{\begin{array}{c}x : P\\ \boxed{\begin{array}{c}\vdots\\ t[x] : Q\end{array}}\end{array}}{x \mapsto t[x] : P \Rightarrow Q}\Rightarrow\mathcal{I} \qquad\qquad \dfrac{f : P \Rightarrow Q \quad x : P}{fx : Q}\Rightarrow\mathcal{E}$$

- Der Beweis einer Implikation ist ein Algorithmus,
- der für jeden Input vom Typ P einen Output vom Typ Q liefert.
- Funktionsdatentyp: Schreibweise: $P \to Q$ oder Q^P.
- Konstruktor: Abstraktion: $(\mapsto)$;
- Selektor: Funktionsanwendung: apply.

Logische Konjunktion, $\wedge$

- Introduktion und Elimination

$$\dfrac{x : P \quad y : Q}{(x, y) : P \wedge Q}\wedge\mathcal{I} \qquad \dfrac{z : P \wedge Q}{\mathsf{fst}\, z : P}\wedge\mathcal{E}_0 \qquad \dfrac{z : P \wedge Q}{\mathsf{snd}\, z : Q}\wedge\mathcal{E}_1$$

- Ein Beweis der Konjunktion $P \wedge Q$ ist ein Paar,
- dessen Komponenten die Typen P bzw. Q haben.
- Verbunddatentyp (Direktes Produkt): $P \times Q$.
- Konstruktor: $(,) : P \to Q \to P \times Q$;
- Selektoren: $\mathsf{fst} : P \times Q \to P$, $\mathsf{snd} : P \times Q \to Q$.

Kommutativität der Konjunktion

Satz

$A \wedge B \implies B \wedge A.$

- Beweis:

$$\dfrac{\dfrac{\boxed{\dfrac{\dfrac{c\colon A \wedge B}{\mathsf{snd}\, c\colon B}\wedge\mathcal{E}_1 \qquad \dfrac{c\colon A \wedge B}{\mathsf{fst}\, c\colon A}\wedge\mathcal{E}_0}{(\mathsf{snd}\, c, \mathsf{fst}\, c)\colon B \wedge A}\wedge\mathcal{I}}}{c \mapsto (\mathsf{snd}\, c, \mathsf{fst}\, c)\colon A \wedge B \implies B \wedge A}\Rightarrow\mathcal{I}}{}$$

- $\mathsf{commute}\colon A \times B \to B \times A,\ c \mapsto (\mathsf{snd}\, c, \mathsf{fst}\, c)$,
- Intuitiver: $(a, b) \mapsto (b, a)$.
- Äquivalenz:
  $\big(c \mapsto (\mathsf{snd}\, c, \mathsf{fst}\, c), c \mapsto (\mathsf{snd}\, c, \mathsf{fst}\, c)\big)\colon A \wedge B \iff B \wedge A.$

Logische Disjunktion, $\vee$

- Introduktion und Elimination

$$\dfrac{x\colon P}{\mathsf{Left}\, x\colon P \vee Q}\vee\mathcal{I}_0 \qquad \dfrac{y\colon Q}{\mathsf{Right}\, y\colon P \vee Q}\vee\mathcal{I}_1$$

$$\dfrac{f\colon P \Rightarrow R \quad g\colon Q \Rightarrow R}{\mathsf{either}\, f\, g\colon P \vee Q \Rightarrow R}\vee\mathcal{E}$$

- Ein Beweis der Disjunktion $P \vee Q$ ist einer von P oder von Q, *und als solcher gekennzeichnet.*
- Disjunkte Vereinigung (Direkte Summe): $P + Q$.
- Konstruktoren: $\mathsf{Left}\colon P \to P + Q$, $\mathsf{Right}\colon Q \to P + Q$;
- Selektor: $\mathsf{either}\colon (P \to R) \to (Q \to R) \to (P + Q \to R)$.

Beispiel: $A \vee (B \wedge C) \implies A \vee B$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a : A}}{\mathsf{Left}\, a : A \vee B}\vee\mathcal{I}_0}{a \mapsto \mathsf{Left}\, a : A \Rightarrow A \vee B}\Rightarrow\mathcal{I} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y : B \wedge C}}{y : B}\wedge\mathcal{E}_1}{\mathsf{fst}\, y : B}\vee\mathcal{I}_1}{y \mapsto \mathsf{Right}(\mathsf{fst}\, y) : B \wedge C \Rightarrow A \vee B}\Rightarrow\mathcal{I}}{\mathsf{either}(a \mapsto \mathsf{Left}\, a)\,(y \mapsto \mathsf{Right}(\mathsf{fst}\, y)) : A \vee (B \wedge C) \Rightarrow A \vee B}\vee\mathcal{E}$$

Mit
$$f : A \vee (B \wedge C) \Rightarrow A \vee B$$
$$g : A \vee (B \wedge C) \Rightarrow A \vee C$$

erhalten wir auch:

$$(f, g) : A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)$$

Es gibt auch: $h : (A \vee B) \wedge (A \vee C) \Rightarrow A \vee (B \wedge C)$

Curry-Howard-Isomorphismus

- Eine Aussage legt den Datentyp ihrer Beweise fest.
- Ein Datentyp entspricht der Aussage, daß es ein Objekt dieses Typs gibt.
- Jeder Algorithmus, der ein Objekt eines bestimmten Datentyps konstruiert, ist ein Beweis, daß es ein solches gibt.
- Aussagen entsprechen Programmspezifikationen.
- Beweise entsprechen Programmen.
- Man kann Aussagen beweisen, indem man ein Objekt vom passenden Typ konstruiert.
- Aus mathematischen Beweisen lassen sich verifizierte Programme extrahieren.
- *Fehlerfreie Software beliebiger Komplexität ist möglich.*