Kapitel 2

Prädikatenlogik und Typentheorie

2.1 Konstruktive Prädikatenlogik

Als Sprache verwenden wir den einfach getypten Lambda-Kalkül mit dem Typisierungsurteil $\Gamma \vdash t : \tau$.

Prädikatenlogik erweitert die Aussagenlogik um zwei neue logische Konnektive, $\forall$ und $\exists$.

BHK-Interpretation:

- Ein Beweis von $\forall x\!:\!\tau.\, A$ ist ein Verfahren, dass für ein beliebiges Element $a : \tau$ einen Beweis von $A[a/x]$ liefert.
- Ein Beweis von $\exists x\!:\!\tau.\, A$ ist ein Paar (w, p), bestehend aus einem Zeugen $w : \tau$ und einem Beweis p von $A[w/x]$.

Beispiel 30 $((\exists x\!:\!\tau.P) \Rightarrow Q) \Rightarrow \forall x\!:\!tau.P \Rightarrow Q$ entspricht dem *currying.*

Die Aussage $(\forall x : \tau.\, P \vee Q(x)) \Rightarrow P \vee \forall x : \tau.\, Q(x)$ (wobei x nicht frei in P vorkomme) ist konstruktiv nicht gültig. Um zu entscheiden, ob P gilt oder $\forall x : \tau.\, Q(x)$, müßte man erst für alle x in τ testen, ob P oder $Q(x)$ gilt. Im allgemeinen kann man aber das nicht in endlicher Zeit tun.

Beispiel 31 (Induktion) Das Induktionsaxiom für das Prädikat P kann formuliert werden als $P\,\mathsf{zero} \Rightarrow (\forall n\!:\!\mathsf{nat}.P\,n \Rightarrow P\,(\mathsf{succ}\,n)) \Rightarrow \forall n\!:\!\mathsf{nat}.\,P\,n$.

Regeln des natürlichen Schliessens:

$$\frac{\Gamma, a\!:\!\tau \vdash A[a/x]}{\Gamma \vdash \forall x\!:\!\tau.\,A}\,\forall\mathsf{I}\ a \text{ neu} \qquad \frac{\Gamma \vdash \forall x\!:\!\tau.\,A \qquad \Gamma \vdash t : \tau}{\Gamma \vdash A[t/x]}\,\forall\mathsf{E}$$

$$\frac{\Gamma \vdash A[t/x] \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \exists x\!:\!\tau.\,A}\,\exists\mathsf{I} \qquad \frac{\Gamma \vdash \exists x\!:\!\tau.\,A \qquad \Gamma, a\!:\!\tau, A[a/x] \vdash C}{\Gamma \vdash C}\,\exists\mathsf{E}$$

Beweisterme und Typisierung: Für $\forall\mathsf{I}$ verwenden wir die Abstraktion (wie für $\Rightarrow\mathsf{I}$), und für $\forall\mathsf{E}$ die Applikation (wie für $\Rightarrow\mathsf{E}$).

$$\frac{\Gamma, a\!:\!\tau \vdash p[a/y] : A[a/x]}{\Gamma \vdash \lambda y.p : \forall x\!:\!\tau.\, A}\ \forall\mathsf{I}\ a \text{ neu} \qquad \frac{\Gamma \vdash p : \forall x\!:\!\tau.\, A \qquad \Gamma \vdash t : \tau}{\Gamma \vdash p\, t : A[t/x]}\ \forall\mathsf{E}$$

In Coq wird $\lambda y.p$ als `fun y => p` geschrieben.

Ein Beweis einer existentiellen Aussage $\exists x\!:\!\tau.A$ ist ein Paar (t, p) bestehend aus einem Zeugen $t : \tau$ und einem Beweis p von $A[t/x]$.

$$\frac{\Gamma \vdash p : A[t/x] \qquad \Gamma \vdash t : \tau}{\Gamma \vdash (t, p) : \exists x\!:\!\tau.\, A}\ \exists\mathsf{I} \qquad \frac{\Gamma \vdash p : \exists x\!:\!\tau.\, A \qquad \Gamma, a\!:\!\tau, h : A[a/x] \vdash q : C}{\Gamma \vdash \mathsf{let}\ (a, h) = p\ \mathsf{in}\ q : C}\ \exists\mathsf{E}$$

In Coq wird das Paar mit `ex_intro _ t p` gebildet und mit

```
match p with ex_intro a h => q
```

eliminiert.

2.2 Kripke-Modelle für Prädikatenlogik

Ein Kripke-Modell für die Prädikatenlogik ist ein Quadrupel $(K, \leq, \Vdash, \{[\![-]\!]^k\}_{k\in K})$, wobei $(K, \leq, \Vdash)$ ein aussagenlogisches Kripke-Modell ist. Für jede Welt $k \in K$ ist $[\![-]\!]^k$ eine Interpretation der Typen und Funktions- und Prädikatensymbole. Die Interpretationen erfüllen folgende Monotoniebedingungen:

Falls $k' \leq k$, so gilt:

- Für alle Typen τ: $[\![\tau]\!]^k \subseteq [\![\tau]\!]^{k'}$ (neue Elemente dürfen hinzukommen, alte Elemente dürfen nicht verschwinden).
- Für alle Funktionssymbole $f : \tau_1 \times \cdots \times \tau_n \to \tau$, $[\![f]\!]^k(x_1, \ldots, x_n) = [\![f]\!]^{k'}(x_1, \ldots, x_n)$ für alle $x_i \in [\![\tau_i]\!]^k$. (Interpretation von Funktionen kann sich nicht ändern, nur auf neue Elemente erweitern).
- Für alle Prädikatensymbole $P : \tau_1 \times \cdots \times \tau_n \to \mathsf{Prop}$, $[\![P]\!]^k(x_1, \ldots, x_n)$ impliziert $[\![P]\!]^{k'}(x_1, \ldots, x_n)$ für alle $x_i \in [\![\tau_i]\!]^k$ (Prädikate dürfen nur "wahrer" werden).

Eine Umgebung η ist eine partielle Abbildung von Variablen und Parametern auf Elemente der Typinterpretationen einer Welt.

Die Interpretation $[\![t]\!]^k_\eta$ eines Terms t in der Umgebung ist definiert durch:

$$\begin{array}{lcl} [\![x]\!]^k_\eta & = & \eta(x) \\ [\![f(t_1, \ldots, t_n)]\!]^k_\eta & = & [\![f]\!]^k([\![t_1]\!]^k_\eta, \ldots [\![t_n]\!]^k_\eta) \end{array}$$

Lemma 32 (Korrektheit der Term-Interpretation) *Wenn $\Gamma \vdash t : \tau$ und $\eta(a) \in [\![\Gamma(a)]\!]^k$ für alle $a \in \mathsf{FV}(t)$, dann $[\![t]\!]^k_\eta \in [\![\tau]\!]^k$.*

Die Erfüllbarkeit $k \Vdash A\ \eta$ ist eine Relation zwischen einer Welt k, einer Formel A und einer Umgebung η. Folgende Bedingungen müssen erfüllt sein:

- $k \Vdash P(\vec{t})\ \eta$ gdw. $[\![P]\!]^k([\![\vec{t}]\!]^k_\eta)$.
- $k \Vdash (\forall x\!:\!\tau.A)\ \eta$ gdw. $k' \Vdash A\ \eta[x\mapsto o]$ für alle $k' \leq k$ und alle $o \in [\![\tau]\!]^{k'}$.
- $k \Vdash (\exists x\!:\!\tau.A)\ \eta$ gdw. es ein $o \in [\![\tau]\!]^k$ gibt, so dass $k \Vdash A\ \eta[x\mapsto o]$.
- $k \Vdash (A \Rightarrow B)\ \eta$ gdw. $k' \Vdash A\ \eta$ impliziert $k' \Vdash B\ \eta$ für alle $k' \leq k$.
- Klauseln für die übrigen aussagenlogischen Konnektive, analog zu Kripke-Modellen für Aussagenlogik.

Beispiel 33 (Markov-Prinzip) Wir konstruieren ein Gegenmodell zu der klassisch gültigen Formel $\neg(\forall x : \tau.\, \neg P(x)) \Rightarrow \exists x : \tau.\, P(x)$ [**?**]. Das Modell besteht aus der Gegenwart und einer Zukunft. Der Typ τ besteht jeweils aus genau einem Element o. $P(o)$ gelte in der Zukunft, aber nicht in der Gegenwart. Damit ist die Folgerung $\exists x\!:\!\tau.\, P(x)$ falsch. Wir müssen zeigen, dass die Prämisse $\neg(\forall x : \tau.\, \neg P(x))$ wahr ist. Dazu muss in allen Welten $\forall x : \tau.\, \neg P(x)$ falsch sein. Es genügt, dass $\neg P(x)$ in der Zukunft falsch ist, was auch der Fall ist.

Beispiel 34 (Markov-Prinzip Variante) Ein Gegenmodell zu $(\neg\forall x\!:\!\tau.\, P(x)) \Rightarrow \exists x : \tau.\, \neg P(x)$: Die Interpretation der Bedingung $(\neg\forall x : \tau.\, P(x))$ ist: Jede Welt hat eine Zukunft, in der es ein x gibt, so dass nicht $P(x)$. Salopp: immer wieder gibt es ein x, dass P nicht erfüllt. Die Folgerung $\exists x\!:\!\tau.\, \neg P(x)$ liesst sich als: In der Gegenwart gibt es ein x, dass in aller Zukunft nie P erfüllt.

Ein Gegenmodell sieht zum Beispiel so aus: τ wird in allen Welten als $\mathbb{N}$ interpretiert, und $P^k(x)$ gdw. $k > x$. Damit gibt es in jeder Welt k ein Element $x = k$, dass P nicht erfüllt, aber jedes Element erfüllt irgendwann einmal P.

Wachsende Typinterpretationen braucht man, um folgende Formel zu widerlegen:

Beispiel 35 Gegenmodell zu $(\forall x : \tau.\, P \vee Q(x)) \Rightarrow P \vee (\forall x : \tau.\, Q(x))$: Welten $K = 0, 1$. Gegenwart: $[\![\tau]\!]^0 = \{0\}, [\![P]\!]^0 = \emptyset, [\![Q]\!]^0 = \{0\}$. Zukunft: $[\![\tau]\!]^1 = \{0, 1\}, [\![P]\!]^1 = \{()\}, [\![Q]\!]^1 = \{0\}$. Dann ist in jeder Welt P wahr oder $Q(x)$ für alle $x : \tau$, aber weder ist P wahr in der Gegenwart noch in allen Welten $Q(x)$ für alle $x : \tau$.

Theorem 36 In jedem Kripke-Modell, in dem die Typinterpretationen in der Zeit fest bleiben, also $[\![\tau]\!]^k = [\![\tau]\!]^{k'}$ für alle k, k', gilt $(\forall x\!:\!\tau.\, P \vee Q(x)) \Rightarrow P \vee (\forall x\!:\! \tau.\, Q(x))$.

Beweis. Entweder gilt in allen Welten $Q(x)$ für alle $x : \tau$. Oder es gibt eine Welt k und ein $x : \tau$, so dass $Q(x)$ in k nicht gilt. Dann gilt aber $Q(x)$ auch in der Gegenwart nicht. Also muss in der Gegenwart P gelten.

Dieser Beweis funktioniert nicht für wachsende Interpretationen, da das $x : \tau$ der Welt k in der Gegenwart nicht notwendigerweise existiert. $\square$

2.3 Induktive Datentypen

Ein *freier Datentyp* ist gegeben durch eine Menge von *Konstruktoren*, mit denen Elemente diese Typs erzeugt werden können.

Beispiel 37 (Binärbäume) Binärbäume, die an den Knoten mit natürlichen Zahlen beschriftet sind, werden von den folgenden Konstruktoren erzeugt.

$$\begin{array}{lll} \mathsf{leaf} & : & \mathsf{nattree} \\ \mathsf{node} & : & \mathsf{nattree} \to \mathsf{nat} \to \mathsf{nattree} \to \mathsf{nattree} \end{array}$$

Bsp: $\mathsf{node}(\mathsf{node}(\mathsf{leaf}, 1, \mathsf{leaf}), 0, \mathsf{leaf})$.

Moderne funktionale Sprachen, wie Haskell und ML, erlauben die Konstruktion von freien Datentypen mit Datentypdeklarationen, z.B. Haskell:

```
data NatTree where
  Leaf :: NatTree
  Node :: NatTree -> Nat -> NatTree -> NatTree
```

Von *induktiven (Daten)typen* spricht man, wenn man nur die *endlich erzeugten* bzw. *fundierten* Elemente des freien Typs betrachtet. Folgender Baum wird also ausgeschlossen:

Beispiel 38 (Unendlicher Baum) Sei $f(x) = \mathsf{node}(\mathsf{leaf}, 0, x)$. Dann ist $f(f(f(f(\dots))))$ ein unendlicher linearer Baum, der die Gleichung $x = f(x)$ löst.

In Coq kann man eigene induktive Datentypen mit dem `Inductive`-Kommando definieren.

```
Inductive nattree : Set :=
| leaf : nattree
| node : nattree -> nat -> nattree -> nattree.
```

Nichtrekursive und rekursive Funktionen lassen sich wie folgt definieren (Vergleichsoperation `le_gt_dec` auf natürlichen Zahlen wird durch `Require Import Compare_dec` verfügbar):

Beispiel 39 (Einfügen in Suchbaum)

```
Definition singleton (n : nat) : nattree := node leaf n leaf.

(* Insert a number n into the binary search tree t. *)
Fixpoint insert (n : nat) (t : nattree) : nattree :=
```