Typentheorie und Logik

Typen und Terme

Typen und Terme

- Typsystem
  - Funktionstypen
  - Produkttypen
  - Listentypen
- Terme
  - λ-Terme
  - bool-Terme
  - Binder
  - Mengen

Typen höherer Ordnung

- Syntax für ein polymorphes Typsystem
  - Konstante Typen
    - bool, nat, …
  - Typvariablen
    - α, β, …
  - Typkonstruktoren: Sind τ, τ_1, τ_2 Typen, dann auch
    - $\tau_1 \to \tau_2$ Funktionstyp
    - $\tau_1 \times \tau_2$ Produkttyp
    - τ list Listentyp
  - Beispiel: $(\alpha \to \beta) \to (\alpha \text{ list} \to \beta \text{ list})$ (Der Typ von „map“)

Terme der Typen

- Signatur Σ definiert Terme wie bisher
- Zusätzliche Terme:

$$\frac{t_1 : \tau_1 \to \tau_2, \quad t_2 : \tau_1}{t_1(t_2) : \tau_2}$$

$$\frac{x : \tau_1, \quad t : \tau_2}{\lambda x.t : \tau_1 \to \tau_2}$$

Produktterme

- Terme für Produkttypen

$$\frac{t_1 : \tau_1, \quad t_2 : \tau_2}{(t_1, t_2) : \tau_1 \times \tau_2}$$

$$\frac{t : \tau_1 \times \tau_2}{\pi_1(t) : \tau_1 \ , \ \pi_2(t) : \tau_2}$$

Listenterme

- Listenterme

$$\frac{}{nil : \tau\ list}$$

nil
= null
= leere Liste

$$\frac{s:\tau,\ t:\tau\ list}{cons(s,t) : \tau\ list}$$

construct

$$\frac{t : \tau\ list\ ,\quad t \neq nil}{car(t):\tau,\quad cdr(t) : \tau\ list}$$

car = head
cdr = tail

bool-Terme

- Aussagenlogische Operatoren sind Terme
  - true , false : bool
  - $\vee$,$\wedge$: bool $\times$ bool $\rightarrow$ bool
  - $\wedge$: bool $\times$ bool $\rightarrow$ bool
  - $\neg$: bool $\rightarrow$ bool
- **Quantoren sind polymorphe Terme**
  - $\forall$: $(\alpha \rightarrow \text{bool}) \rightarrow \text{bool}$
  - $\exists$: $(\alpha \rightarrow \text{bool}) \rightarrow \text{bool}$
- Definition:

  $\forall (f : \tau \rightarrow \text{bool})$: bool = **if** $(f = \lambda x{:}\tau.\text{true})$ **then** true **else** false
  $\exists (f : \tau \rightarrow \text{bool})$: bool = **if** $(f = \lambda x{:}\tau.\text{false})$ **then** false **else** true
  d.h.
  - $\forall (\lambda x{:}\tau.\phi)$ = **if** $(\lambda x{:}\tau.\phi = \lambda x{:}\tau.\text{true})$ **then** true **else** false

    = **if** $(\forall x{:}\tau.\phi)$ **then** true **else** false

    = $(\forall x{:}\tau.\phi)$
  - $\exists x{:}\tau.\phi$ = $\exists (\lambda x{:}\tau.\ \phi)$
- $\forall x{:}\text{nat}.\ \exists y{:}\text{nat}.\ x < y = \forall (\lambda x{:}\text{nat}.\ \exists\ (\lambda y{:}\text{nat}.\ x < y)$

Binder

- Terme der Funktionalität
  $(\alpha \to \tau) \to \sigma$
  heißen Binder, weil sie eine Variable vom Typ α binden

- Beispiele:
  - Sei a: nat $\to \mathbb{R}$ und $\Sigma : (\text{nat} \to \mathbb{R}) \to \mathbb{R}$. Statt
    $\Sigma(a) = \Sigma\ (\lambda\ n{:}\text{nat}.a(n))$
    schreibt man $\Sigma_{n \in N}\ a(n)$

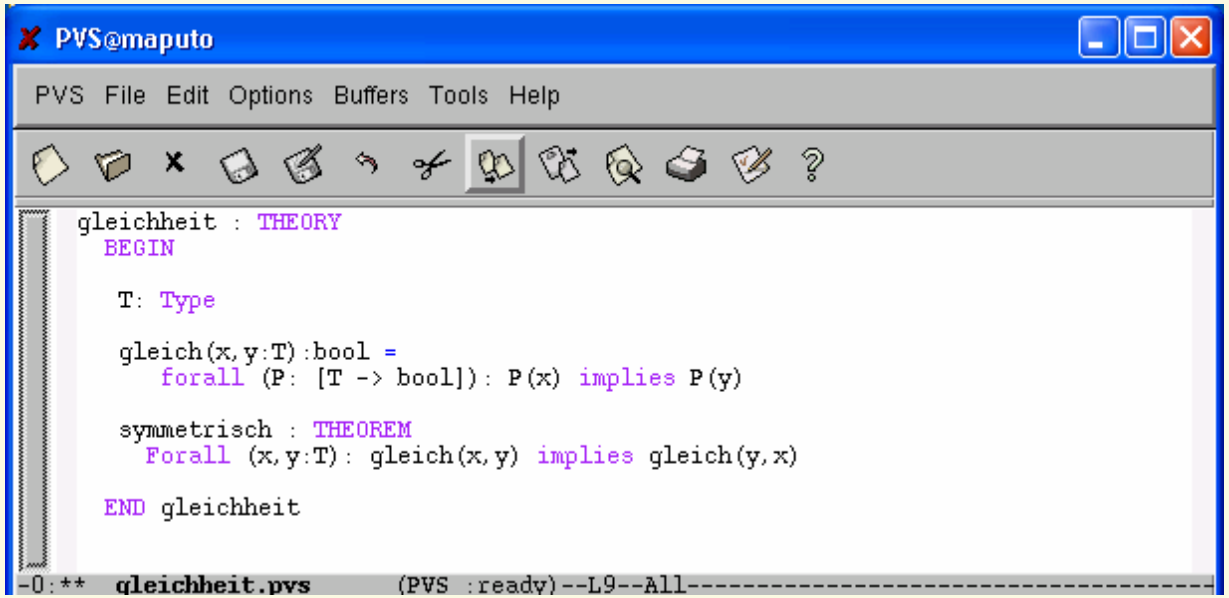
  - Sei $e : \mathbb{R} \to \mathbb{R}$ und $\int : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R}$. Statt
    $\int e = \int (\ \lambda\ x{:}\mathbb{R}.e(x)\ )$
    schreibt man $\int e\ dx$

- Typische Binder der Logik $\forall$, $\exists$, sowie :
  - $\mu : \mathcal{L} \to \mathcal{L} \to \mathcal{L}$ kleinster Fixpunkt
  - $\nu : \mathcal{L} \to \mathcal{L} \to \mathcal{L}$ größter Fixpunkt

- Typischer Binder der Informatik : **fix**
  - fact = fix λ f: nat $\to$ nat. λ n:nat. if n=0 then 1 else n*f(n-1)

Definierbarkeit

- In der Logik höherer Stufe ist die Gleichheit definierbar:
  - $(x=y) :\Leftrightarrow \forall P{:}\alpha\rightarrow \text{bool}.\ P(x) \rightarrow P(y)$.
- Definieren Sie die Gleichheit auf diese Weise in PVS und leiten Sie die Axiome her:
  - Reflexivität, Symmetrie, Transitivität
  - Zeigen Sie in PVS: $(x=y) :\Leftrightarrow \forall P{:}\alpha\rightarrow \text{bool}.\ P(x) \rightarrow P(y)$.



Mengen

- Mengennotation ist syntaktischer Zucker für λ-Terme:

Menge		charakteristische Funktion
– $\{ n : \tau \mid \phi \}$	$:\Leftrightarrow$	$\lambda\, n{:}\tau\, .\, \phi$

- Beispiele
  - $\{ n : nat \mid n \bmod 2 = 0 \}$ $:\Leftrightarrow$ $\lambda\, n{:}nat.\ n \bmod 2 = 0$
  - $\{ x,y : \mathbb{R} \mid x^2+y^2=1 \}$ $:\Leftrightarrow$ $\lambda\, x,y{:}\mathbb{R}.\ x^2+y^2=1$

- Übung beschreiben Sie durch einen λ-Term:
  - $\{ x^2+y^2 \mid x,y{:}\mathbb{R},\ x*y \leq 0 \}$
  - für eine beliebige Abbildung $f{:}\tau \to \sigma$ und $P{:}M \to bool$

    $\{ f(n) \mid n \in N,\ P(n) \}$

Der Satz von Cantor

- Cantor :
  - Für keine Menge X existiert surjektive Abbildung $f: X \to \mathbb{P}(X)$.
- Äquivalent:
  - Für keine Menge X existiert injektive Abbildung $f: \mathbb{P}(X) \to X$
- „$\mathbb{P}(X)$ hat mehr Elemente als X“
  - Schreibweise: $|X| < |\mathbb{P}(X)|$
  - das gilt für endliche und für unendliche Mengen folglich auch:
    - $0 < 1 < 2 < \ldots < |\mathbb{N}| < |\mathbb{P}(\mathbb{N})| < |\mathbb{P}(\mathbb{P}(\mathbb{N}))| < \ldots$
    - *Allgemeine Continuumshypothese*:
      „Es gibt keine Menge M mit $|\mathbb{N}| < |M| < |\mathbb{P}(\mathbb{N})|$“

Beweis des Satzes von Cantor

- Angenommen, $f : X \to \mathbb{P}(X)$ wäre surjektiv.
- Für $u := \{ x \in X \mid x \notin f(x) \}$ gilt:
  $u = f(w)$ für ein $w \in X$

Es folgt der Widerspruch:

$$w \in f(w) \leftrightarrow w \notin f(w).$$

Spezifikation

```
cantor  [ T : Type ]: THEORY
  BEGIN

    cantor: theorem
      forall (f:[T -> setof[T]]):
        exists (u:setof[T]):
          forall (t:T): not f(t)=u
  END cantor
-0:**  cantor.pvs          (PVS :ready)--L8--All
```

instanziieren

```
cantor :

  |-------
{1}   EXISTS (u: setof[T]): FORALL (t: T): NOT f(t) = u


Rule? (inst 1 "{ x:T | not f(x)(x) }")
Instantiating the top quantifier in 1 with the terms:
 { x:T | not f(x)(x) },
this simplifies to:
cantor :

  |-------
{1}   FORALL (t: T): NOT f(t) = ({x: T | NOT f(x)(x)})
```

β–Regel

```
cantor.1 :

[-1]  f(a)(a)
[-2]  ({x: T | NOT f(x)(x)})(a)
  |------


Rule? (beta)
Applying beta-reduction,
this simplifies to:
cantor.1 :

[-1]  f(a)(a)
  |-------
{1}   f(a)(a)

which is trivially true.
```

Fehlt noch was ?

- Unmöglich in simple type theory higher order auszudrücken :
  - Ein bekanntes Lemma von Dickson besagt:

$$\forall k \in \mathbb{N}.\ \forall f\colon \mathbb{N} \to \mathbb{N}^k.\ \ \exists a,b\colon \mathbb{N}.\ a < b\ \wedge \forall i<k.\ \pi_i(f(a)) \leq \pi_i(f(b)).$$

    Hierbei seien $\pi_i : \mathbb{N}^k \to \mathbb{N}$ die Projektionen. In diesem Falle ist der Typ $\mathbb{N}^k$ abhängig von der Variablen k. Die obige Aussage lässt sich daher nicht in der bisherigen Logik höherer Stufe ausdrücken.

- **Dependent Types**
- PVS erlaubt dependent Types, also Typen der Form

  Πx:M N

  Jedem Element x aus M wird ein Typ N(x) zugeordnet.

  Ein Beispiel in PVS:

```
k : VAR nat
upto(k) : Type = {x :nat | x < k}
```

Dependent types

- Argumenttypen und Resultattyp von Funktionen dürfen von früheren Argumenten abhängen.

```
take(x:list[nat], n:upto(length(x)))
                                        :list[nat]
```

oder

```
take(x:list[nat], n:upto(length(x)))
                                :{l:list[nat]|length(l)<n}
```

- Wir gehen hier nicht näher darauf ein.

Folgerung

- Mit Funktionsobjekten und Funktionstypen lassen sich alle Begriffe der Logik höherer Stufe einheitlich repräsentieren.
- Viele Begriffe sind syntaktischer Zucker für besondere λ-Terme:
  - Quantoren
  - Mengennotation
  - Arrays
- Die Implementierung von PVS baut auf diesem Typsystem auf.