Programmier-Paradigmen

Tutorium – Gruppe 2 & 8

Henning Dieterichs

Rekursion Im Lambda Kalkül

Rekursion

- `fun := (\fun -> term) fun =: term[fun]`
  - `fun ⇒ term[term[term[term[term[term[term[term[term[term[term[…]]]]]]]]]]]`
- Kann es ein `fun` geben, sodass
  `fun = term[fun]` (Gleichheit)?
- Kann es ein `fun` geben, sodass
  `fun ≅ term[fun]` (Ähnliches Verhalten)?
  - `∃X: fun ⇒ X ⇐ (\fun -> term) fun`
  - Verhalten sich irgendwann gleich bzgl. β-Reduktion in Normalreihenfolge
- Dieses `fun` wäre ein Fixpunkt (bzgl. ähnlichem Verhalten) vom Funktional
  `funF := \fun -> term`.
- Der Y Kombinator berechnet solch einen Fixpunkt:
  `fun := Y funF`

Rekursion

- Es bleibt zu zeigen:
  - `∃X: fun ⇒ X ⇐ funF fun`
  - für `fun := Y funF`
  - und `Y := \f . (\x . f (x x))(\x . f (x x))`
- Also
  - `∃X:`
    `(\f . (\x . f (x x))(\x . f (x x))) funF`
    `⇒ X ⇐`
    `funF ((\f . (\x . f (x x))(\x . f (x x))) funF)`

Rekursion

```
(\f . (\x . f (x x))(\x . f (x x))) funF;
```

Run

```
X (λf . (λx . f ( x x ) ) (λx . f ( x x ) ) ) funF
X (λx . funF ( x x ) ) (λx . funF ( x x ) )
X funF ( (λx . funF ( x x ) ) (λx . funF ( x x ) ) )
```

```
funF ((\f . (\x . f (x x))(\x . f (x x))) funF);
```

Run

```
X funF ( (λf . (λx . f ( x x ) ) (λx . f ( x x ) ) ) funF )
X funF ( (λx . funF ( x x ) ) (λx . funF ( x x ) ) )
```

4 Typ-Prüfung

Es sei $\Gamma = \mathtt{a} : \mathsf{int}, \mathtt{b} : \mathsf{bool}, \mathtt{c} : \mathsf{char}$.

Vervollständigen Sie den folgenden Herleitungsbaum:

$$\dfrac{\Gamma \vdash \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x} : \alpha \to \beta \to \alpha \qquad \Gamma, k : \forall\alpha.\forall\beta.\alpha \to \beta \to \alpha \vdash \mathtt{k\ a\ (k\ b\ c)} : \mathsf{int}}{\Gamma \vdash \mathtt{\textbf{let}\ k = \lambda x.\ \lambda y.\ x\ \textbf{in}\ k\ a\ (k\ b\ c)} : \mathsf{int}}\ \mathit{Let}$$

Benutzen Sie (neben trivialen Instanziierungen) die Instanziierungen:

- $\forall\alpha.\forall\beta.\alpha \to \beta \to \alpha \succeq \mathsf{int} \to \mathsf{bool} \to \mathsf{int}$
- $\forall\alpha.\forall\beta.\alpha \to \beta \to \alpha \succeq \mathsf{bool} \to \mathsf{char} \to \mathsf{bool}$

Regelsystem

$$\textsc{Const}: \frac{c \in \mathit{Const}}{\Gamma \vdash c : \tau_c}$$

$$\textsc{Var}: \frac{\Gamma(x) = \tau' \qquad \tau' \succeq \tau}{\Gamma \vdash x : \tau}$$

$$\textsc{App}: \frac{\Gamma \vdash t_1 : \tau_2 \to \tau \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash t_1\ t_2 : \tau}$$

$$\textsc{Abs}: \frac{\Gamma, x : \tau_1 \vdash t : \tau_2 \qquad \tau_1 \text{ kein Typschema}}{\Gamma \vdash \lambda x.\ t : \tau_1 \to \tau_2}$$

$$\textsc{Let}: \frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : ta(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \tau_2}$$

1 Φ-Kombinator [Klausuraufgabe vom SS 2012] [15 Punkte]

In der Vorlesung waren die Kombinatoren S, K, I definiert als:

$$S = \lambda\texttt{x}.\ \lambda\texttt{y}.\ \lambda\texttt{z}.\ \texttt{x z (y z)} \qquad K = \lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{x} \qquad I = \lambda\texttt{x}.\ \texttt{x}$$

Es gilt also: $K\ x\ y \Rightarrow^2 x$ für λ-Terme x, y

Weiterhin werden Kombinatoren Φ, B definiert als:

$$\Phi = \lambda\texttt{o}.\ \lambda\texttt{f}.\ \lambda\texttt{g}.\ \lambda\texttt{x}.\ \texttt{o (f x) (g x)} \qquad B = \lambda\texttt{f}.\ \lambda\texttt{g}.\ \lambda\texttt{x}.\ \texttt{f (g x)}$$

B dient also der Funktionskomposition, Φ der punktweisen Verknüpfung von Funktionen $\texttt{f}, \texttt{g}$ durch Operator $\texttt{o}$. Punktweise Addition zweier Funktionen $\texttt{f}, \texttt{g}$ z.B. wird geschrieben: $(\Phi\ \texttt{(+) f g})$, denn damit gilt dann: $\Phi\ \texttt{(+) f g} \Rightarrow^3 \lambda\texttt{x}.\ \texttt{(+) (f x) (g x)}$

1. Geben Sie einen allgemeinsten Typ von Φ an! [7 Punkte]
   Es ist hierbei keine formale Herleitung erforderlich.

   Hinweis: $\texttt{o}$ muss nicht unbedingt ein arithmetischer Operator sein

Spaß mit SKI-Kalkül

$(\mathbf{S}\ x\ y\ z) = (x\ z\ (y\ z))$

$(\mathbf{K}\ x\ y) = x$

$(\mathbf{I}\ x) = x$

T: Lambda-Term ohne freie Variablen ⇒ SKI-Term

1. $\mathbf{T}[v] \mapsto v$ *für v Variable*
2. $\mathbf{T}[(E1\ E2)] \mapsto (\mathbf{T}[E1]\ \mathbf{T}[E2])$
3. $\mathbf{T}[\lambda x.E] \mapsto (K\ \mathbf{T}\ [E])$ *wenn x nicht frei in E*
4. $\mathbf{T}[\lambda x.x] \mapsto I$
5. $\mathbf{T}[\lambda x.\lambda y.E] \mapsto \mathbf{T}\ [\lambda x.\mathbf{T}[\lambda y.E]]$
6. $\mathbf{T}[\lambda x.(E1\ E2)] \mapsto (S\ \mathbf{T}[\lambda x.E1]\ \mathbf{T}[\lambda x.E2])$

⇒ SKI-Kalkül ist turingvollständig

Spaß mit SKI-Kalkül

- SKK x ⇒ I x
- X := λx.((xS)K)
  - X (X (X X)) ⇒ K
  - X (X (X (X X))) ⇒ S

⇒ X-Kalkül ist turingvollständig