Vorgängerfunktion

Idee: der Vorgänger von n wird definiert, indem in der Repräsentation von $\underline{n} = (\lambda f\ x\ .\ f^n\ x)$ die erste Anwendung von f in f^n entfernt wird.

Hilfsfunktion prefn:
„n Anwendungen von prefn auf (true, x) sollen n-1 Anwendungen von f auf x ergeben, also $(\underline{\text{prefn}}\ f)^n\ (\underline{\text{true}}, x) = (\underline{\text{false}}, f^{n-1}x)$ für n>0"

$$\text{LET } \underline{\text{prefn}} = \lambda f\ p.(\underline{\text{false}}, (\underline{\text{fst}}\ p \to \underline{\text{snd}}\ p \mid (f\ (\underline{\text{snd}}\ p)\ )))$$

Vorgängerfunktion pre:

$$\text{LET } \underline{\text{pre}} = \lambda n\ f\ x.\underline{\text{snd}}\ (n\ (\underline{\text{prefn}}\ f)\ (\underline{\text{true}}, x))$$

$$\begin{aligned} \underline{\text{pre}}\ \underline{n}\ f\ x &= \underline{\text{snd}}\ (\underline{n}\ (\underline{\text{prefn}}\ f)\ (\underline{\text{true}}, x)) && \text{(*definition von pre*)}\\ &= \underline{\text{snd}}\ ((\underline{\text{prefn}}\ f)^n\ (\underline{\text{true}}, x)) && \text{(*definition von } \underline{n}\text{*)}\\ &= \underline{\text{snd}}\ (\underline{\text{false}}, f^{n-1}\ x) && \text{(* n -anwendungen von prefn *)}\\ &= f^{n-1}\ x \end{aligned}$$

Rekursion

Idee: $\underline{\text{mult}}\ m\ n = \underline{\text{add}}\ n\ (\underline{\text{add}}\ n\ (...(\underline{\text{add}}\ n\ \underline{0})...))$

$\underline{\text{mult}}\ m\ n = (\underline{\text{iszero}}\ m \to \underline{0} \mid \underline{\text{add}}\ n\ (\underline{\text{mult}}\ (\underline{\text{pre}}\ m)\ n))$ bzw.

$\underline{\text{mult}} \quad = \lambda m\ n\ .(\underline{\text{iszero}}\ m \to \underline{0} \mid \underline{\text{add}}\ n\ (\underline{\text{mult}}\ (\underline{\text{pre}}\ m)\ n))$

!!! ungebunden !!!

Lösung: Definiere einen zunächst ein $\underline{Y}$ mit der folgenden Eigenschaft:

$\underline{Y}\ E = E\ (\underline{Y}\ E)$

Def. E' ist ein Fixpunkt von E, wenn E' = E E'

Def. Fix heißt Fixpunktoperator, wenn
Fix E = E (Fix E) $\forall$ E

$$\text{LET } \underline{Y} = \lambda f.\ (\lambda x.\ f(x\ x))\ (\lambda x.\ f(x\ x))$$

LET $\underline{Y} = \lambda f.\ (\lambda x.\ f(x\ x))\ (\lambda x.\ f(x\ x))$

Die rekursiv definierte Multiplikationsfunktion:

LET $\underline{\text{mult}} = \underline{Y}\ \underline{\text{multfn}}$

wobei

LET $\underline{\text{multfn}} = \lambda \mathbf{f}\ m\ n\ .(\underline{\text{iszero}}\ m \to \underline{0} \mid \underline{\text{add}}\ n\ (\mathbf{f}\ (\underline{\text{pre}}\ m)\ n))$

$$\begin{aligned} \underline{\text{mult}}\ m\ n &= (\underline{Y}\ \underline{\text{multfn}})\ m\ n && \text{(*Def. von } \underline{\text{mult}} \text{ *)}\\ &= \underline{\text{multfn}}\ (\underline{Y}\ \underline{\text{multfn}})\ m\ n && \text{(*Fix-Punkt-Eigenschaft von Y*)}\\ &= \underline{\text{multfn}}\ \underline{\text{mult}}\ m\ n && \text{(*Def. von } \underline{\text{mult}} \text{ *)}\\ &= (\lambda \mathbf{f}\ m\ n\ .(\underline{\text{iszero}}\ m \to \underline{0} \mid \underline{\text{add}}\ n\ (\mathbf{f}\ (\underline{\text{pre}}\ m)\ n)))\ \underline{\text{mult}}\ m\ n\\ &= (\underline{\text{iszero}}\ m \to \underline{0} \mid \underline{\text{add}}\ n\ (\text{mult}\ (\underline{\text{pre}}\ m)\ n)) && \text{(*}\beta\text{-Regel*)} \end{aligned}$$

Eine Gleichung der Form $\underline{\text{fktn}}\ x_1 \ldots x_n = E$ wird rekursiv genannt, wenn $\underline{\text{fktn}}$ in E als freie Variable vorkommt.

a) Schreibe die gewünschte Funktion als Gleichung der Form:
$\underline{\text{fktn}}\ x_1 \ldots x_n = \ldots..\ \underline{\text{fktn}}\ \ldots....$

b) Definiere die Funktion durch: LET $\underline{\text{fktn}} = \underline{Y}\ (\lambda f\ x_1 \ldots x_n\ .\ \ldots..\ f\ \ldots....)$

Repräsentation von Funktionen

Im λ-Kalkül gibt es zwei Varianten zur Repräsentation:
a) (f x ,... x_n) (* gecurryt *) Bsp. add m n = m+n
b) f (x_1,...,x_n) sum (m, n) = m+n

LET curry = λf x_1 x_2. f (x_1, x_2)
LET uncurry = λf p. f (fst p) (snd p)

nun gilt: sum = uncurry add

sum (m, n) = uncurry add (m, n)
= (λf p. f (fst p)(snd p)) add (m, n)
= add (fst (m, n)) (snd (m, n))
= add m n
= m+n

Rekursive Funktionen

Zwei Funktionsklassen:
a) primitiv rekursive Funktionen
b) rekursive Funktionen

Eine Funktion heißt **primitiv rekursiv**, wenn sie ausgehend von
- der konstanten Funktion 0,
- einer Nachfolgerfunktion S(x) = x+1
- einer Projektionsfunktion U_n^i (x_1,x_2, ... , x_n) = x_i
- unter Verwendung von Substitution und
- des primitiven Rekursionschemas

konstruiert werden kann.

Substitution und primitive Rekursion

Substitution:
Sei g eine Funktion mit r Argumenten und h_1, ..., h_r seien n-stellige Funktionen.
Wir sagen f ensteht aus g und h durch Substitution, wenn:

$$f(x_1,...,x_n) = g(h_1(x_1,...,x_n), ... , h_n(x_1,...,x_n))$$

Primitive Rekursion:
Sei g eine Funktion mit n-1 Argumenten und h eine Funktion mit n+1 Argumenten. Wir sagen, f wird aus g und h durch primitive Rekursion definiert, wenn:

$$f(0, x_2, ..., x_n) = g(x_2, ... , x_n)$$
$$f(S(x_1),x_2,...,x_n) = h(f(x_1,x_2,...,x_n), x_1,x_2,...,x_n)$$

Bsp.
Beh. sum ist primitiv rekursiv
Bew. $sum(0, x_2) = x_2$
$sum(S(x_1), x_2) = S(sum(x_1, x_2))$

Jede primitiv-rekursive Funktionen ist im λ-Kalkül darstellbar!

0 sei $\underline{0}$, S sei $\underline{\text{suc}}$ und U_n^i sei der λ-Ausdruck, der aus der Repräsentation eines n-Tupels, das i-te Element extrahiert.
g sei $\underline{\text{gfktn}}$ und habe r Argumente
h_i seien $\underline{\text{hktn}_i}$ und haben je n Argumente ($1 \leq i \leq r$)

Wenn die n-stellige Funktion f durch Substitution definiert wird als

$$f(x_1,...,x_n) = g(h_1(x_1,...,x_n), \dots , h_r(x_1,...,x_n))$$

dann wird f repräsentiert durch $\underline{\text{ffktn}}$, wobei

$$\underline{\text{ffktn}} = \lambda(x_1,...,x_n) \,.\, g(h_1(x_1,...,x_n), \dots , h_n(x_1,...,x_n))$$

Sei $f(\ 0,\quad x_2, ..., x_n) = g(x_2, \dots , x_n)$
$\quad f(S(x_1),x_2, ..., x_n) = h(f(x_1,x_2,...,x_n), x_1,x_2,...,x_n)$.
Dann repräsentiert $\underline{\text{ffktn}}$ f, wenn gilt:

$$\underline{\text{fktn}}\ (x_1, x_2, \dots , x_n) = (\underline{\text{iszero}}\ x_1 \rightarrow \underline{\text{gfktn}}(x_2,...,x_n) \mid \underline{\text{hfktn}}(\underline{\text{fktn}}(\underline{\text{pre}}\ x_1, x_2, ...,x_n), \underline{\text{pre}}\ x_1, x_2, ...,x_n)))$$

Unter Ausnutzung des Fixpunkt-Trickes folgt:

$$\underline{Y}(\lambda f.\lambda(x_1,x_2,...,x_n).\ (\underline{\text{iszero}}\ x_1 \rightarrow \underline{\text{gfktn}}(x_2,...,x_n) \mid \underline{\text{hfktn}}(\ f\ (\underline{\text{pre}}\ x_1, x_2, ...,x_n), \underline{\text{pre}}\ x_1, x_2, ...,x_n)))$$

xx Rekursive Funktionen

Eine Funktion heißt **rekursiv**, wenn sie ausgehend von der konstanten Funktion 0, einer Nachfolgerfunktion S, einer Projektionsfunktion U_n^i duch eine Folge von Substitutionen, primitive Rekursionen und **Minimierungen** konstruiert werden kann.

Minimierung
Sei g eine n-stellige Funktion. Wir sagen, dass f aus g durch Minimung entsteht, wenn

$$f(x_1,x_2,...x_n) = \text{„das kleinste y mit } g(y,x_2,...,x_n) = x_n\text{“}$$

Als Notation für die Minimierung von f verwenden wir MIN(f).

Funktionen, die durch Minimierung definiert wurden, können für einige Argumente nicht definiert sein.
Bsp. sei one(x) = 1. => $\text{MIN(one)}(x) = \begin{cases} \underline{0} & \text{falls } x=1 \\ \perp & \text{sonst.} \end{cases}$

Jede rekursive Funktion ist im λ-Kalkül darstellbar!

z. Z. ist die Repräsentation der Minimierung einer willkürlichen Funktion.
Sei $\underline{\text{gfktn}}$ die Repräsentation für die n-stellige Funktion g und sei f = MIN(g).
Sei $\underline{\text{min}}$ ein λ-Ausdruck, so dass ($\underline{\text{min}}\ \underline{x}\ \underline{\text{fktn}}\ (\underline{x_1},...,\underline{x_n})$) die kleinste Zahl y repräsentiert, die größer als x ist, so dass

$$f(y,x_2,...,x_n) = x_1$$

dann wird g durch $\underline{\text{gfktn}}$ definiert, wobei

$$\underline{\text{gfktn}} = \lambda(x_1, x_2, \dots , x_n) \,.\, \underline{\text{min}}\ \underline{0}\ \underline{\text{fktn}}\ (x_1, x_2, \dots , x_n)$$

$\underline{\text{min}}$ hat die gewünschte Eigenschaft, wenn

$$\underline{\text{min}}\ x\ f\ (x_1, x_2, \dots , x_n) = (\underline{\text{eq}}\ (f(x, x_2, \dots , x_n))\ x_1) \rightarrow x \mid \underline{\text{min}}\ (\underline{\text{suc}}\ x\ f\ (x_1, x_2,...,x_n))$$

mit $\underline{\text{eq}}\ \underline{m}\ \underline{n}$ ist wahr, wenn m = n.

Dieses $\underline{\text{min}}$ lässt sich wie folgt definieren

$$\underline{Y}(\lambda m.\lambda x\ f\ (x_1, x_2, \dots , x_n).\ (\underline{\text{eq}}\ (f(x, x_2, \dots , x_n))\ x_1) \rightarrow x \mid m\ (\underline{\text{suc}}\ x)\ \ f\ (x_1, x_2,...,x_n)))$$

Funktionen höherer Ordnung

betrachte: *Ackermann-Funktion* (rekursiv, aber nicht primitiv-rekursiv):

ack (0, n) = n+1
ack (m+1,0) = ack (m,1)
ack (m+1, n+1) = ack(m, ack (m+1, n))

• keine primitiv-rekursive Funktion als obere Schranke, da starkes Wachstum

Lösungansatz:
Funktionen höherer Ordnung: Funktionen, die (andere) Funktionen als Argumente zulassen oder als Ergebnis ausgeben.

Sei die Hilfsfunktion rec primitiv-rekursiv definiert:

$rec(0,x_2,x_3) = x_2$
$rec(S(x_1),x_2,x_3) = x_3(rec(x_1,x_2,x_3))$

dann kann ack definiert werden durch

ack (m, n) = rec(m, S, f→(x→rec(x,f(1), f))) (n)

Funktionspfeil

Partiell-rekursive Funktionen

Eine Funktion heißt **partiell**, wenn sie nicht für alle Argumente definiert ist.
Bsp. MIN(one), Divisionsfunktion

Eine **partiell rekursive** Funktion ist eine rekursive Funktion, die nicht für alle Argumente definiert ist.

Es kann gezeigt werden, dass alle partiell rekursiven Funktionen im λ-Kalkül in dem Sinne repräsentiert werden können., dass

(i) $\underline{f}(\underline{x_1}, ..., \underline{x_n}) = \underline{y}$, wenn $f(x_1,...,x_n)=y$
(ii) wenn $f(x_1,...,x_n)$ nicht definiert ist, dann hat $\underline{f}(\underline{x_1}, ..., \underline{x_n})$ keine Normalform.

Church-Rosser-Theorem

Wenn $E_1 = E_2$, dann existiert ein E, so dass $E_1 \to E$ und $E_2 \to E$.

1) Folgerung:
Zwei λ-Ausdrücke können in beliebiger Reihenfolge ausgewerten werden.
Beweis:
Gelte $E \to E_1$ und $E \to E_2$ und E_1, E_2 sind Normalformen, dann folgt: $E=E_1$ und $E=E_2$ (* Def. der Gleichheit *)
und damit $E_1=E_2$.
Hieraus folgt: $\exists E'$ mit $E_1 \to E'$ und $E_2 \to E'$. (* Church-Rosser *)
Da E_1 und E_2 Normalformen sind, können höchstens α-Reduktionen durchgeführt werden, um E' zu erhalten. Daher müssen die Ausdrücke E_1 und E_2 bis auf die Namensgebung der gebundenen Variablen identisch sein.

2) Ähnlich kann gezeigt werden, dass mit m≠n auch $\underline{m} \neq \underline{n}$ folgen muss.

Korollar zum Church-Rosser-Theorem

(i) Wenn E eine Normalform hat,
dann ex. E→E‘ mit E‘ ist Normalform

(ii) Wenn E eine Normalform hat und E=E‘ ,
dann hat E‘ eine Normalform

(iii) Wenn E=E‘ und sowohl E als auch E‘ sind Normalformen,
dann sind E und E‘ identisch (außer α-Reduktionen).

Normalisierungs Theorem

Wenn E eine Normalform hat, dann erhält man aus E eine Normalform, indem wiederholend die am weitesten links arbeitende β- oder η-Reduktion angewendet wird (ggf. nach einer vorhergehenden α-Reduktion).