Übungsblatt 6

Abgabe der Lösungen: Do, 13.06., 12:00 im Briefkasten am blauen Hochhaus

PRÄSENZAUFGABEN

Church-Numerale

Wir betrachten erneut die Church-Kodierung natürlicher Zahlen vom vorigen Übungsblatt:

$$\lceil n \rceil := \lambda f a. \underbrace{f(f(f(\ldots f}_{n} a))))$$

mit einheitlicher Kodierung

$$\begin{array}{ll} \mathit{zero} & = \lambda f\, a\ .\ a \\ \mathit{succ}\ n & = \lambda f\, a\ .\ f\ (n\ f\ a) \end{array} \qquad \begin{array}{ll} \mathit{one} & = \mathit{succ}\ \mathit{zero} \\ \mathit{two} & = \mathit{succ}\ \mathit{one} \\ \mathit{three} & = \mathit{succ}\ \mathit{two} \\ \mathit{four} & = \mathit{succ}\ \mathit{three} \end{array}$$

sowie den Operationen *add* und *mult* mit der entsprechenden Semantik.

Notation: Von nun an schreiben wir $s\ +\ t$ und $s\ *\ t$ anstelle von $\mathit{add}\ s\ t$ und $\mathit{mult}\ s\ t$ und verwenden die $\beta\delta$-Regeln

$$\lceil n \rceil\ +\ \lceil m \rceil \to^*_{\beta\delta} \lceil n + m \rceil \qquad\qquad \lceil n \rceil\ *\ \lceil m \rceil \to^*_{\beta\delta} \lceil n \cdot m \rceil .$$

Übung 1 Church-Kodierung von Booleans

Boolesche Wahrheitswerte werden als λ-Terme wie folgt definiert:

$$\begin{array}{ll} \mathit{true} & = \lambda x\, y.\ x \\ \mathit{false} & = \lambda x\, y.\ y \\ \mathit{ite} & = \lambda\, b\, x\, y.\ b\ x\ y \end{array}$$

1. Zeigen Sie, dass für alle λ-Terme s und t gilt:

$$\mathit{ite}\ \ \mathit{true}\ \ s\ \ t \to^*_{\beta\delta} s \qquad\qquad \mathit{ite}\ \ \mathit{false}\ \ s\ \ t \to^*_{\beta\delta} t$$

2. Vervollständigen Sie die folgenden Funktionsdefinitionen, so dass sie (unter normaler Reduktion) boolesche Negation, exklusives Oder und Implikation berechnen:

$$\begin{array}{ll} \mathit{not}\ b & = \ldots \\ \mathit{xor}\ b1\ b2 & = \ldots \\ \mathit{imp}\ b1\ b2 & = \ldots \end{array}$$

Notation: Von nun an schreiben wir "**if** s **then** t **else** u" anstelle von "$\mathit{ite}\ \ s\ \ t\ \ u$".

Übung 2 Rekursive Definitionen

In den meisten funktionalen Programmiersprachen sind *rekursive* Funktionsdefinitionen zulässig, das heißt, die definierte Funktion darf auf der rechten Seite einer solchen Funktionsdefinition vorkommen. Rekursive Funktionsdefinitionen entsprechen – wie in Übung 2, Blatt 4 – δ-Reduktionen.

Hinweis. Nehmen Sie an, dass die Subtraktion von natürlichen Zahlen (in Form von Church-Numeralen) im λ-Kalkül darstellbar ist, d.h. für $n \geq 1$ gilt $\lceil n \rceil - \lceil 1 \rceil \to_\beta^* \lceil n-1 \rceil$, und dass ebenso die üblichen Vergleichsoperationen möglich sind, d.h. $\lceil n \rceil \leq \lceil 1 \rceil \leftrightarrow_{\beta\delta}^* \mathit{true}$ wenn $n \leq 1$ usw. Siehe dazu Übung 4.

1. Wir betrachten die folgende rekursive Funktion:

   $$\mathit{fact}\ \ n = \textbf{if}\ n \leq \lceil 1 \rceil\ \textbf{then}\ \lceil 1 \rceil\ \textbf{else}\ n * (\mathit{fact}\ \ (n - \lceil 1 \rceil))$$

   Zeigen Sie, dass $\mathit{fact}\ \ \lceil 3 \rceil \to_{\beta\delta}^* \lceil 6 \rceil$.

2. Schreiben Sie eine rekursive Funktion $\mathit{odd}\ \lceil n \rceil$, die *true* als Ergebnis liefert, wenn n ungerade ist, und *false* andernfalls.

3. Schreiben Sie eine rekursive Funktion *halve*, so dass:

   $(\lceil 2 \rceil * \mathit{halve}\ \ \lceil n \rceil) + \textbf{if}\ \mathit{odd}\ \lceil n \rceil\ \textbf{then}\ \lceil 1 \rceil\ \textbf{else}\ \lceil 0 \rceil\ \ \to_{\beta\delta}^* \lceil n \rceil$.

Übung 3 Auswertungsstrategien

In der Vorlesung haben Sie verschiedene Reduktionsstrategien für den ungetypten λ-Kalkül kennengelernt. Diese unterscheiden sich hauptsächlich in den Zeitpunkten, zu denen β-Redexe *kontrahiert* werden, also wann in einem Term die β-Regel angewandt wird.

1. Welcher Redex im λ-Term

   (a) $(\lambda x.\lambda y.y(\lambda z.x))\ \ (u\ u)\ \ (\lambda v.v((\lambda w.w)\ (\lambda w.w)))$

   (b) $(\lambda u.u\ (\lambda y.z))\ \ (\lambda x.x\ \ ((\lambda v.v)(\lambda v.v)))$

   muss nicht kontrahiert werden, um die Normalform zu erreichen? Reduzieren Sie den Term durch $\beta\delta$-Reduktion zur Normalform, ohne diesen Redex zu kontrahieren.

2. Wir schreiben wie aus der Vorlesung bekannt $\Omega = (\lambda x.\, x\, x)$. Reduzieren Sie den Term $(\lambda f.\ f\ \ I\ \ (\Omega\, \Omega))(\lambda x\, y.\ x\ \ x)$ mittels

   (a) applikativer Reduktion,

   (b) normaler Reduktion.

   Unterstreichen Sie in jedem Schritt den zu reduzierenden Redex. Betrachten Sie in dieser Aufgabe die δ-Reduktionen als explizite Schritte!

3. Man erinnere sich an folgende auf Church-Kodierungen definierten Funktionen:

$$
\begin{array}{l}
\text{-- Allgemein} \\
twice \;=\; \lambda f\, x.\;\; f\;\; (f\;\; x) \\
\text{-- Church-Booleans} \\
true \;=\; \lambda x\, y.\; x \\
false \;=\; \lambda x\, y.\; y \\
\text{-- Church-Paare} \\
pair \;=\; \lambda a\; b\;\; select\;.\;\;\; select\;\;\; a\;\; b \\
fst \;=\; \lambda p.\; p\;\; (\lambda x\, y.\;\; x) \\
snd \;=\; \lambda p.\; p\;\; (\lambda x\, y.\;\; y)
\end{array}
$$

   Geben sie die ersten fünf $\beta\delta$-Reduktionsschritte des Terms

$$twice\;\;\; fst\;\;\; (pair\;\;\; (pair\;\;\; true\;\;\; false\,)\;\;\; true)$$

   unter a) normaler und b) applikativer Reduktion an. Markieren Sie (durch Unterstreichen) in jedem Schritt den zu reduzierenden Redex.

HAUSAUFGABEN

Übung 4 Subtraktion (7 Punkte)

Das Dekrementieren von Church-Numeralen ist wesentlich komplizierter als die Addition, Multiplikation oder Exponentiation; angeblich benötigte Alonzo Church einige Jahre, um herauszubekommen, dass (und auf welche Weise) es möglich ist. Allerdings gab es zu dieser Zeit noch keine Computer, so dass Church auch kein Programmierer im klassischen Sinne war.

$$pred\ n = \lambda f\ a.\ n\ (\lambda\ g\ h.\ h\ (g\ f))\ \ (\lambda\ u.\ a)\ \ (\lambda\ u.\ u)$$

1. Zeigen Sie, dass $pred\ \lceil 2 \rceil \rightarrow^*_{\beta\delta} \lceil 1 \rceil$.

2. Vervollständigen Sie die folgenden Funktionsdefinitionen:

$$\begin{array}{lll} sub\ n\ m = \ldots & // & n - m \\ eq\ \ n\ m = \ldots & // & n == m \\ le\ \ n\ m = \ldots & // & n \leq m \\ lt\ \ n\ m = \ldots & // & n < m \end{array}$$

3. Erläutern Sie in eigenen Worten die Funktionsweise des durch *pred* implementierten Algorithmus.

Übung 5 Wer braucht schon Rekursion? (7 Punkte)

Rekursive Definitionen sind komfortabel, aber nicht notwendig: Bereits im reinen λ-Kalkül ist es möglich, Fixpunkt-Kombinatoren zu definieren (wie z.B. den Y-Kombinator). Es sei *fix* ein beliebiger Fixpunkt-Kombinator, d.h. es gelte $fix\ \ f \leftrightarrow^*_{\beta\delta} f\ (fix\ f)$.

1. Die folgende Definition entspricht der Funktion *fact* aus Übung 2; jedoch wurde die Rekursion durch einen Fixpunkt ersetzt:

$$fact' = fix\ \ (\lambda myself\ n\ .\ \ \textbf{if}\ \ n \leq \lceil 1 \rceil\ \textbf{then}\ \lceil 1 \rceil\ \textbf{else}\ n * (myself\ \ (n - \lceil 1 \rceil)))$$

   Zeigen Sie, dass $fact'\ \ \lceil 3 \rceil \leftrightarrow^*_{\beta\delta} \lceil 6 \rceil$

2. Wir betrachten die folgende rekursive Funktion:

$$fib\ \ n = \textbf{if}\ n \leq \lceil 1 \rceil\ \textbf{then}\ \lceil 1 \rceil\ \textbf{else}\ fib\ \ (n - \lceil 1 \rceil)\ \ + fib\ \ (n - \lceil 2 \rceil)$$

   Geben Sie einen λ-Term t an, so dass für jeden Fixpunkt-Kombinator *fix* der Term $fix\ \ t$ dieselbe Funktion berechnet wie *fib*.

Übung 6 Auswertungsstrategien II (6 Punkte)

Man erinnere sich an folgende auf Church-Kodierungen definierten Funktionen:

$$
\begin{aligned}
&-- \text{ Allgemein} \\
&\mathit{twice} \;=\; \lambda f\, x.\; f\;\; (f\;\; x) \\
&\mathit{const} \;=\; \lambda x\, y.\; x \\
&I \;=\; \lambda x.\; x \\
&-- \text{ Church}-\text{Summen} \\
&\mathit{inl} \;=\; \lambda a\, f\, g.\; f\;\; a \\
&\mathit{inr} \;=\; \lambda b\, f\, g.\; g\;\; b \\
&\mathit{case} \;=\; \lambda f\, g\;\; s.\; s\;\; f\;\; g
\end{aligned}
$$

Geben sie die ersten fünf $\beta\delta$-Reduktionsschritte des Terms

$$\mathit{twice}\;\; ((\lambda x.\;\; \mathit{case}\;\; \Omega\;\; \mathit{const}\;\; x)\;\; (\mathit{inr}\;\; I))\;\; I$$

unter a) normaler und b) applikativer Reduktion an. Markieren Sie (durch Unterstreichen) in jedem Schritt den zu reduzierenden Redex.