Max Wisniewski , Alexander Steen

Tutor: Ansgar Schneider

Aufgabe 1 α - *Konversion*

Wenn man für die α - Reduktion $\lambda x.t \underset{\alpha}{\rightarrow} \lambda y \$^x_y$ auf die Bedingung $y \notin Var(t)$ verzichtet, kann eine solche Reduktion die Semantik verändern. Geben Sie dafür ein Beispiel an.

Lösung:
Seien $y, a \in A_\lambda$ mit $y \neq a$ und $(\lambda x.y)a$ λ-Ausdruck.
Nun können wir die 2 folgenden Ausführungsreihenfolgen angeben:

1:

$$\begin{aligned}(\lambda x.y)a &\xrightarrow{\beta} \$^x_a\, y \\ &\longrightarrow y\end{aligned}$$

2:

$$\begin{aligned}(\lambda x.y)a &\xrightarrow{\alpha} (\lambda y.\$^x_y\, y)a \\ &\longrightarrow (\lambda y.y)a \\ &\xrightarrow{\beta} \$^y_a y \\ &\longrightarrow a\end{aligned}$$

Wir haben nun ein beiden Fällen eine Normalform erreicht. Wir können nicht einmal eine α - Konversion anwenden, da es keine gebundenen Variablen mehr gibt.
Nun sagt uns aber das Church - Rosser - Theorem, dass wir durch 2 verschiedene Anwendungs von Reduktionen, die Ergbnisse auf die gleiche Syntaktische Form bringen können müssen.
Da $a \neq y$ gilt, sind bei Ausdrücke nicht äquivalent.

Aufgabe 2 β - *Konversion*

Wenn man für die β - Reduktion $(\lambda x.t)s \rightarrow \$^x_s t$ auf die Forederung $Fr(s) \cap Geb(t) = \emptyset$ verzichtet, kann eine solche Reduktion die Semantik verändern.
Geben Sie dafür ein Beispiel an.

Lösung:
Seien $y, t \in A_\lambda$ mit $y \neq t$ und $(\lambda xy.x)\ y\ t$ λ-Ausdruck. Nun können wir die folgenden beiden Reduktionen angeben:

1:

$$\begin{aligned}(\lambda xy.x)\ y\ t &\xrightarrow{\alpha} (\lambda xa.x)\ y\ t \\ &\xrightarrow{\beta} (\lambda a.y)t \\ &\xrightarrow{\beta} y\end{aligned}$$

2:

$$\begin{aligned}(\lambda xy.x)\ y\ t &\xrightarrow{\beta} (\lambda y.\$^x_y x)t \\ &\longrightarrow (\lambda y.y)t \\ &\xrightarrow{\beta} t\end{aligned}$$

Wie in der ersten Aufgabe haben wir 2 Ausdrücke in Normalform, deren freie Variablen nicht gleich sind. Damit können sie nicht in einander überführt werden und die Semantik hat sich verändert.

Aufgabe 3 *Nicht-endliche Reduktion*

Konstruieren Sie einen λ-Ausdruck t, der keine Normalform besitzt und dessen Reduktion zu immer größeren Ausrücken führt.

Lösung:
Wir benutzen die sich reproduzierenden Teil aus dem Fixpunktkombinator $Y \equiv (\lambda f.(\lambda xy.y(xx))(\lambda xy.y(xx))f)$, nämlig den inneren Teil $(\lambda x.a(xx))(\lambda x.a(xx))$. Wir führen hier ein paar Schritte der Reduktion aus:

$$\begin{array}{rcl}(\lambda x.a(xx))(\lambda x.a(xx)) & \xrightarrow{\beta} & a((\lambda x.a(xx))(\lambda x.a(xx))) \\ & \xrightarrow{\beta} & a(a((\lambda x.a(xx))(\lambda x.a(xx)))) \\ & ... & \end{array}$$

Wie wir sehen, reproduziert sich der Ursprüngliche Ausdruck immer wieder und wir schachteln die Konstante a immer weiter ineinander.

Aufgabe 4 *Getypter λ - Kalkül*

Schreiben Sie je einen getypten λ - Ausdruck für die folgenden Aufgaben:

(a) Eine symmetrische Funktion soll dreifach auf ein Argument angewendet werden.
Lösung:
Wir konstruieren zunächst die allgemeine Version der Funktionen, die wir brauchen und wenden sie im Spezialfall an.
$\underline{\text{iterate}} : \mathbb{N} \to [D \to D] \to [D \to D]$ um diese Funktion zu bauen, beachte zunächst, das durch die Church - Notation der Natürlichen Zahlen gilt: $nfx : D$ mit $n \in X^{\mathbb{N}}$, $f \in x^{[D->D]}$ und $x \in X^D$.
(Iterativ gilt $0fx = (\lambda xy.y)fx\overset{\beta}{rightarrow}x$,
$(n+1)fx = (\lambda xy.x(nxy))fx\overset{\beta}{f}(nfx) \overset{I.V.}{\to} f^{n+1}x)$.
Nun bauen wir $\underline{\text{iterate}} = \lambda fn.nf \ : \mathbb{N} \to [D \to D]$, das die Signatur nach konstruktion der anwendung von natürlichen Zahlen auf Funktionen erfüllt.

Wir betrachten nun eine symmetrische Addition, die als Ergebniss wieder einen Tupel liefern muss die Zahl 2 mal wiederholt. Andernfalls könnten wir die Funktion nicht mehrmals anwenden.

$\underline{sym} = (\lambda x.\underline{tupel}(\underline{fst}x + \underline{snd}x)(\underline{fst}x + \underline{snd}x) : D_1 \times D_2 \to D_1 \times D_2$.

Ein Tupel hat die Form
$(\lambda x.xab) : [D_1 \to D_2 \to D] \to D = D_1 \times D_2$
mit $x \in X^{[D_1 \to D_2 \to D]}$, $a \in X^{D_1}$ und $b \in X^{D_2}$.

Nun ist $(\lambda xy.x) : D_1 \to D_2 \to D_1$ eine Abbildung auf das erste Element und
$(\lambda xy.x) : D_1 \to D_2 \to D_2$ eine Abbildung auf das zweite Element.

Als nächstes betrachten wir die Addition, dass wir nicht aus dem Bereich heraus laufen, den wir betrachten: $\underline{+} = (\lambda mnxy.mx(nxy)) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ Dieser Beweis funktioniert analog zur Iteration.

Nun ist die endgülitge Funktion
$\underline{sym} = (\lambda x.\lambda y((\lambda mnab.ma(nab))((\lambda ab.a)x)((\lambda ab.b)))((\lambda mnab.ma(nab))((\lambda ab.a)x)((\lambda ab.b)x)))$

Unsere Funktion $\lambda x.\underline{iterate}\ 3\ \underline{sym}\ x : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ ist eine Funkion, die eine symmetrische Funktion 3 mal auf ein Tupel von natürlichen Zahlen anwendet.

(b) Gebeben sei eine Liste der Länge 4 von Elementen des Typs D und eine Funktion vom Typ $[D \to D]$, berechnen Sie die Anwendung dieser Funktion auf alle Listenelemente.
Lösung:
Wir konstruieren in diesem Fall zunächst die Funktion $\underline{map} : [D_1 \to D_2] \to D_1^* \to D_2^*$.
Was wir als erstes betrachten ist die Listen konstruktion im λ - Kalkül:
$[] = (\lambda x.(\lambda yz.z))$ mit $x \in X^{D \times D^* \to D'}$
$(x : xs) = (\lambda y.yxxs)$ mit $y \in X^{D \times D^* \to D'}$, $x \in X^D$ und $xs \in X^{D^*}$. Da wir eine Tupelkonstruktion haben, können wir $fst = head$ und $snd = tail$ benutzen. $\underline{empty} = (\lambda a.a(\lambda bcde.d))$. Eine leere Liste schmeißt die parameter weg und ist selber false und jede andere Liste sorgt dafür, dass head und tail weggeworfen werden und einzig und allein true stehen bleibt.

Nun machen wir uns an die übliche konstruktion der Map implementierung.

```
map f xs =
    if !empty xs then
        []
    else
        cons (f (head xs)) (map f (tail xs))
```

.

Wir brauchen für die Implementierung nun nur noch das bekannt zu Übetragen:

$\underline{map} =$	
$(\lambda f\ l.$	f : Funktion, l : list
$(\lambda i.(\lambda xy.y(xx))(\lambda xy.y(xx))i)$	Fixpunktkombinator
$(\lambda rx.$	Map funktion
$(\lambda a.a(\lambda bcde.d))x$	if !empty(x)
$(\lambda a.a(f((\lambda bc.b)x))(r((\lambda bc.c)x)))$	f(head(x)):map f tail x
$(\lambda a.\lambda yz.z))$	else []
$l)$	initial ganz l

Wenn wir nun eine Liste vom Typ D und der Länge 4 mit einer Funktion $f : [D \to D]$ an die Funktion map geben, so wird das Ergebnis heraus kommen.

Sei $\underline{xs} : D$ eine solche Liste und $f : [D \to D]$ eine solche Funkion.
Dann erfüllt:
$map\ f\ \underline{xs} : D^*$ die Vorraussetzungen.

(c) Beschreiben Sie den uncurry-Operator im getypten λ - Kalkül, der angewandt auf eine Funktion vom Typ $[D_1 \to [D_2 \to D_3]]$ eine Funktion des Typs $[(D_1 \times D_2) \to D_3]$ liefert, wobei für alle f, a und b

$$(uncurry\ f)\ < a, b >= f\ a\ b$$

gelten soll.
Lösung:
Da wir uns nun schon zuvor mit Tupel beschäftigt haben, ist die Funktion nicht weiter schwierig zu konstruieren: $\underline{uncurry} : [D_1 \to [D_2 \to D_3]]$ nimmt im folgenden eine Funktion dieses Types entgegen und formatiert die eingabe, die danach kommt um: $\underline{uncurry} = (\lambda fx.f(x(\lambda ab.a))(x(\lambda ab.b)))$, mit $f \in X^{D_1 \to D_2 \to D_3}$, $x \in X^{D_1 \times D_2}$.

Wie schon zuvor gezeigt, ist $fst = (\lambda ab.a) : D_1 \times D_2 \to D_1$ und $snd = D_1 \times D_2 \to D_2$.

Wir zeigen nun, dass für alle f,a und b gilt $(uncurry\ f) < a, b >= f\ a\ b$

$$\begin{aligned}
(uncurry f) < a, b > &= f(fst < a, b >)(snd < a, b >) \\
&= f((\lambda x.xab)(\lambda rs.r))((\lambda x.xab)(\lambda rs.s)) \\
&\overset{\beta}{\to} f((\lambda rs.r)ab)((\lambda rs.s)ab) \\
&\overset{\beta}{\to} f((\lambda s.a)b)((\lambda s.s)b) \\
&\overset{\beta}{\to} f\ a\ b
\end{aligned}$$

Damit ist unser *uncurry* erstens Typkorrekt und liefert zweitens das richtige Ergebnis.