Zur Erinnerung: λ-Kalkül

- Motivation \o/: minimale universelle Programmiersprache
- Lambda-Abstraktion immer geklammert: $(\lambda x.F)$
- Applikation von Termen ist linksassoziativ: EFG ist implizit geklammert als $(EF)G$
- Kurzschreibweise: $(\lambda xy.F)$ für $(\lambda x.(\lambda y.F))$
- Vor Ausführung einer β-Reduktion $(\lambda x.F)G$ müssen die freien Variablen in G und die gebundenen in F bestimmt werden.
- Sind die Mengen nicht disjunkt (d.h. $FV \cap GV \neq \emptyset$), müssen bei der α-Konversion Variablen umbenannt werden.
- β-Reduktion: In $(\lambda x.F)G$ werden alle Vorkommen der Variablennamen x in F durch G ersetzt und $\lambda x.$ wird entfernt.
- Die Applikation wird so lange wiederholt, bis keine Reduktion mehr möglich ist.
- Nicht für alle Terme existiert eine β-Normalform (vgl. Blatt 4, Übung 4 (b) (4))

Übung 2

$$
\begin{aligned}
& <\text{pow}>\ <2> = (\lambda n f z.n(\lambda g x.g(gx))fz)(\lambda h y.h(hy)) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda h y.h(hy))(\lambda g x.g(gx))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda g x.g(gx))((\lambda g x.g(gx))y))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda x.((\lambda g x.g(gx))y)(((\lambda g x.g(gx))y)x)))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda x.((\lambda g x.g(gx))y)((\lambda x.y(yx))x)))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda x.((\lambda g x.g(gx))y)(y(yx))))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda x.(\lambda x.y(yx))(y(yx))))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.(\lambda x.y(y(y(yx)))))fz) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda x.f(f(f(fx))))z) \\
\Rightarrow_\beta\ & (\lambda f z.f(f(f(fz)))) = <4>
\end{aligned}
$$

Übung 2

$$
\begin{aligned}
& \text{<pow>}\ \text{<0>} = (\lambda n f z.n(\lambda g x.g(g x)) f z)(\lambda h y.y) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda h y.y)(\lambda g x.g(g x)) f z) \\
\Rightarrow_\beta\ & (\lambda f z.(\lambda y.y) f z) \\
\Rightarrow_\beta\ & (\lambda f z.f z) = \text{<1>}
\end{aligned}
$$

Zusatzaufgabe 1

Induktionsanfang (IA)

Sei `t = Leaf x` und `x :: a`.

$$
\begin{aligned}
& \texttt{\textbf{reverse} (yield (Leaf x))} \\
\overset{(9)}{=}\; & \texttt{\textbf{reverse} [x]} \\
\overset{(E1)}{=}\; & \texttt{[x]} \\
\overset{(9)}{=}\; & \texttt{yield (Leaf x)} \\
\overset{(5)}{=}\; & \texttt{yield (mirror (Leaf x))}
\end{aligned}
$$

Induktionsvoraussetzung (IV)

Seien `t1, t2 :: Tree a`, sodass gilt:

$$
\begin{aligned}
&\texttt{\textbf{reverse} (yield t1)} = \texttt{yield (mirror t1)} \qquad \texttt{(IV1)} \\
&\texttt{\textbf{reverse} (yield t2)} = \texttt{yield (mirror t2)} \qquad \texttt{(IV2)}
\end{aligned}
$$

Zusatzaufgabe 1

Induktionsschritt (IS)

Für alle `t = Node x t1 t2` und `x :: a` zeigen wir, dass

`reverse` `(yield t)` = `yield (mirror t)`

gilt:

$$
\begin{aligned}
& \texttt{\textbf{reverse} (yield (Node x t1 t2))} \\
\overset{(8)}{=}\ & \texttt{\textbf{reverse} (yield t1 ++ yield t2)} \\
\overset{(E2)}{=}\ & \texttt{\textbf{reverse} (yield t2) ++ \textbf{reverse} (yield t1)} \\
\overset{(IV1)}{=}\ & \texttt{\textbf{reverse} (yield t2) ++ yield (mirror t1)} \\
\overset{(IV2)}{=}\ & \texttt{yield (mirror t2) ++ yield (mirror t1)} \\
\overset{(8)}{=}\ & \texttt{yield (Node x (mirror t2) (mirror t1))} \\
\overset{(4)}{=}\ & \texttt{yield (mirror (Node x t1 t2))} \qquad \square
\end{aligned}
$$