Teil II

Theoretische Grundlagen

Kalküle

Kalküle sind

- minimalistische Programmiersprachen zur Beschreibung von Berechnungen,
- mathematische Objekte, über die Beweise geführt werden können.

In dieser Vorlesung:

λ-Kalkül	(Church, Landin) für sequentielle (funktionale/imperative) Sprachen

Beispiele weiterer Kalküle:

CSP	(Hoare) Communicating Sequential Processes - für nebenläufige Programme mit Nachrichtenaustausch
π-Kalkül	(Milner) für nebenläufige, mobile Programme

Der untypisierte λ-Kalkül

Alonzo Church

* 1903; † 1995

Der untypisierte λ-Kalkül

- Turing-mächtiges Modell funktionaler Programme
- Auch: Beschreibung sequentieller imperativer Konstrukte

λ-Terme

Bezeichnung	Notation	Beispiele	
Variablen	x	$\mathtt{x}$	$\mathtt{y}$
Abstraktion	$\lambda x.\ t$	$\lambda \mathtt{y}.\ 0$	$\lambda \mathtt{f}.\ \lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \mathtt{f}\ \mathtt{y}\ \mathtt{x}$
Funktionsanwendung	$t_1\ t_2$	$\mathtt{f}\ 42$	$(\lambda \mathtt{x}.\ \mathtt{x} + 5)\ 7$
(weitere primitive Operationen nach Bedarf)		17, *True*, $+$, $\cdot$, $\ldots$	

Variablenkonvention:

$\mathtt{x}$, $\mathtt{y}$, $\mathtt{f}$	sind konkrete Programmvariablen
x, y, z	sind Meta-Variablen für Programmvariablen
t, t', t_1, t_2, ...	bezeichnen immer einen λ-Term

Funktionsanwendung linksassoziativ, bindet stärker als Abstraktion

$$\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}\ \mathtt{y} = \lambda \mathtt{x}.\ ((\mathtt{f}\ \mathtt{x})\ \mathtt{y})$$

Variablenbindung bei Abstraktion

Variablenbindung in Haskell:

Anonyme Funktion: `\ x -> (\ y -> y + 5) (x + 3)`

`let`-Ausdruck: `let x = 5 in x + y`

Analog bei λ-Abstraktionen: $\lambda x.\ t$ bindet die Variable x im Ausdruck t

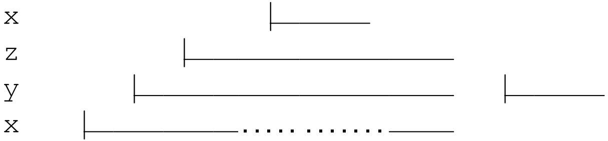
Beispiele:

$\lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \mathtt{f\ y\ x}$ bindet $\mathtt{x}$ in $\lambda \mathtt{y}.\ \mathtt{f\ y\ x}$, das selbst $\mathtt{y}$ in $\mathtt{f\ y\ x}$ bindet.

$\mathtt{f}$ ist frei in $\lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \mathtt{f\ y\ x}$.

Innere Abstraktionen können äußere Variablen verdecken:

$$(\lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \lambda \mathtt{z}.\ \mathtt{f}\ (\lambda \mathtt{x}.\ \mathtt{z} + \mathtt{x})\ (\mathtt{y\ x}))\ (\lambda \mathtt{y}.\ \mathtt{y} + \mathtt{x})$$



α-Äquivalenz

Namen gebundener Variablen

- dienen letztlich nur der Dokumentation
- entscheidend sind die Bindungen

α-Äquivalenz

t_1 und t_2 heißen α-äquivalent ($t_1 \stackrel{\alpha}{=} t_2$), wenn t_1 in t_2 durch konsistente Umbenennung der λ-gebundenen Variablen überführt werden kann.

Beispiele:

$$\lambda \mathtt{x}.\ \mathtt{x} \stackrel{\alpha}{=} \lambda \mathtt{y}.\ \mathtt{y}$$

$$\lambda \mathtt{x}.\ (\lambda \mathtt{z}.\ \mathtt{f}\ (\lambda \mathtt{y}.\ \mathtt{z}\ \mathtt{y})\ \mathtt{x}) \stackrel{\alpha}{=} \lambda \mathtt{y}.\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\lambda \mathtt{z}.\ \mathtt{x}\ \mathtt{z})\ \mathtt{y})$$

aber

$$\lambda \mathtt{x}.\ (\lambda \mathtt{z}.\ \mathtt{f}\ (\lambda \mathtt{y}.\ \mathtt{z}\ \mathtt{y})\ \mathtt{x}) \stackrel{\alpha}{\neq} \lambda \mathtt{x}.\ (\lambda \mathtt{z}.\ \mathtt{g}\ (\lambda \mathtt{y}.\ \mathtt{z}\ \mathtt{y})\ \mathtt{x})$$

$$\lambda \mathtt{x}.\ (\lambda \mathtt{z}.\ \mathtt{f}\ (\lambda \mathtt{y}.\ \mathtt{z}\ \mathtt{y})\ \mathtt{x}) \stackrel{\alpha}{\neq} \lambda \mathtt{z}.\ (\lambda \mathtt{z}.\ \mathtt{f}\ (\lambda \mathtt{y}.\ \mathtt{z}\ \mathtt{y})\ \mathtt{z})$$

η-Äquivalenz

Extensionalitäts-Prinzip:

- Zwei Funktionen sind gleich, falls Ergebnis gleich für alle Argumente

η-Äquivalenz

Terme $\lambda \mathtt{x}.\ f\ \mathtt{x}$ und f heißen η-äquivalent ($\lambda \mathtt{x}.\ f\ \mathtt{x} \stackrel{\eta}{=} f$) falls $\mathtt{x}$ nicht freie Variable von f

Beispiele:

$$\lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \underline{\mathtt{f\ z\ x}}\ \mathtt{y} \stackrel{\eta}{=} \lambda \mathtt{x}.\ \mathtt{f\ z\ x}$$

$$\mathtt{f\ z} \stackrel{\eta}{=} \lambda \mathtt{x}.\ \underline{\mathtt{f\ z}}\ \mathtt{x}$$

$$\lambda \mathtt{x}.\ \mathtt{x} \stackrel{\eta}{=} \lambda \mathtt{x}.\ \underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ \mathtt{x}$$

aber

$$\lambda \mathtt{x}.\ \underline{\mathtt{f\ x}}\ \mathtt{x} \stackrel{\eta}{\neq} \mathtt{f\ x}$$

Ausführung von λ-Termen

Redex Ein λ-Term der Form $(\lambda x.\ t_1)\ t_2$ heißt Redex.

β-Reduktion β-Reduktion entspricht der Ausführung der Funktionsanwendung auf einem Redex:

$$(\lambda x.\ t_1)\ t_2 \Rightarrow t_1\ [x \mapsto t_2]$$

Substitution $t_1\ [x \mapsto t_2]$ erhält man aus dem Term t_1, wenn man alle freien Vorkommen von x durch t_2 ersetzt.

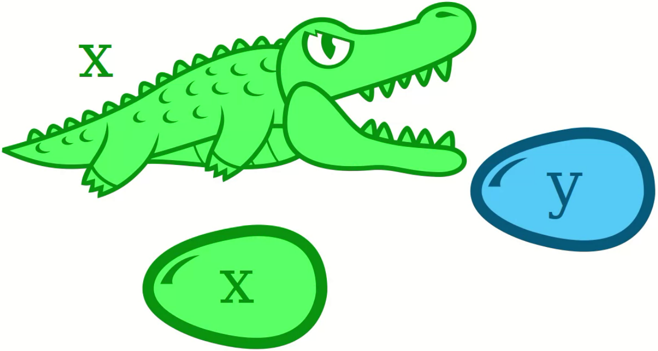
Normalform Ein Term, der nicht weiter reduziert werden kann, heißt in Normalform.

Beispiele:

$$\underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ \mathtt{y} \Rightarrow \mathtt{x}\ [\mathtt{x} \mapsto \mathtt{y}] = \mathtt{y}$$

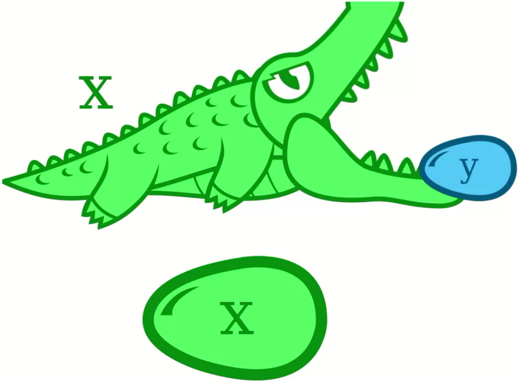
$$\underline{(\lambda \mathtt{x}.\ \mathtt{x}\ (\lambda \mathtt{x}.\ \mathtt{x}))}\ (\mathtt{y}\ \mathtt{z}) \Rightarrow (\mathtt{x}\ (\lambda \mathtt{x}.\ \mathtt{x}))\ [\mathtt{x} \mapsto \mathtt{y}\ \mathtt{z}] = (\mathtt{y}\ \mathtt{z})\ (\lambda \mathtt{x}.\ \mathtt{x})$$

Ausführung von λ-Termen (informell)



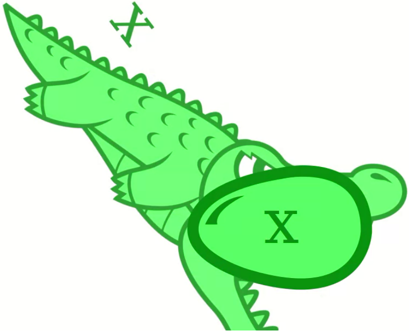
$$\underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{y}$$

Ausführung von λ-Termen (informell)



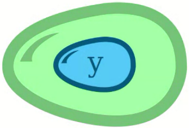
$$x\,[x \mapsto y]$$

Ausführung von λ-Termen (informell)



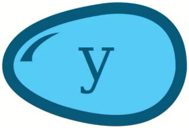
$$x\,[x \mapsto y]$$

Ausführung von λ-Termen (informell)



$$x\,[x \mapsto y]$$

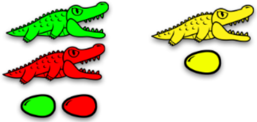
Ausführung von λ-Termen (informell)



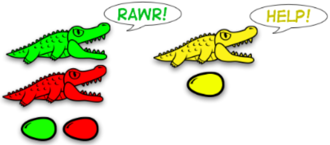
y

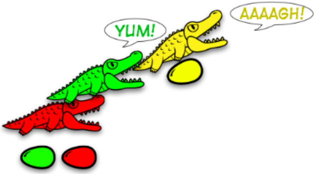
http://metatoys.org/alligator/#!/(%CE%BBx.x)%20y

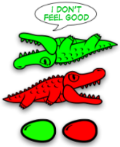
Ausführung von λ-Termen (informell)

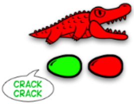


$(\lambda x.\lambda y.\ x\ y)\ \lambda z.z$

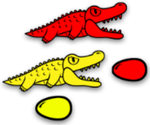




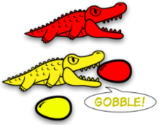


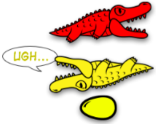


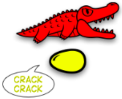
$(\lambda y.\ x\ y)[x \longmapsto \lambda z.z]$



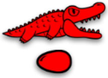
$\lambda y.\ (\lambda z.z)\ y$







$\lambda y.\ (z[z \longmapsto y])$



$\lambda y.\ y$

Beispielreduktionen

$$\underline{(\lambda x.\, x)}\ (\lambda x.\, x) \Rightarrow (\lambda x.\, x)$$ in Normalform.

$$\begin{aligned}
\underline{(\lambda x.\, x\ x)}\ (\lambda x.\, x\ x) &\Rightarrow \underline{(\lambda x.\, x\ x)}\ (\lambda x.\, x\ x)\\
&\Rightarrow \underline{(\lambda x.\, x\ x)}\ (\lambda x.\, x\ x)\\
&\Rightarrow \ldots
\end{aligned}$$

Beachte: Funktionsanwendung ist linksassoziativ!

$$\begin{aligned}
\underline{(\lambda x.\, x\ x\ x)}\ (\lambda x.\, x\ x\ x) &\Rightarrow \underline{(\lambda x.\, x\ x\ x)}\ (\lambda x.\, x\ x\ x)(\lambda x.\, x\ x\ x)\\
&\Rightarrow \underline{(\lambda x.\, x\ x\ x)}\ (\lambda x.\, x\ x\ x)(\lambda x.\, x\ x\ x)(\lambda x.\, x\ x\ x)\\
&\Rightarrow \ldots
\end{aligned}$$

Beispielreduktion SKK

$S = \lambda \mathtt{x\ y\ z.\ (x\ z)\ (y\ z)} = \lambda \mathtt{x.} \lambda \mathtt{y.} \lambda \mathtt{z.\ ((x\ z)\ (y\ z))}$

$K = \lambda \mathtt{x\ y.\ x} = \lambda \mathtt{x.} \lambda \mathtt{y.\ x}$

$$\begin{aligned} \underline{S}\ K\ K &\Rightarrow \underline{(\lambda \mathtt{y.} \lambda \mathtt{z.\ (}K\ \mathtt{z)\ (y\ z))}}\ K \\ &\Rightarrow \lambda \mathtt{z.\ ((}\underline{K}\ \mathtt{z)\ (}K\ \mathtt{z))} \\ &\Rightarrow \lambda \mathtt{z.\ (}\underline{\mathtt{(}\lambda \mathtt{y.\ z)}}\ \mathtt{(}K\ \mathtt{z))} \\ &\Rightarrow \lambda \mathtt{z.\ (z)} \end{aligned}$$

Hinweis: Die „Kombinatorische Logik“ ist ein Vorläufer des λ-Kalküls ohne Variablen. Es sind nur die Kombinatoren *S*, *K* und Applikationen von Kombinatoren notwendig.

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ (\lambda \mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$\underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ (\lambda \mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathtt{x}.\ \mathtt{x})\ (\underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ (\lambda \mathtt{z}.\ (\lambda \mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{z}))$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda\mathtt{x}.\ \mathtt{x})\ ((\lambda\mathtt{x}.\ \mathtt{x})\ (\lambda\mathtt{z}.\ (\lambda\mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

$$(\lambda\mathtt{x}.\ \mathtt{x})\ ((\lambda\mathtt{x}.\ \mathtt{x})\ (\lambda\mathtt{z}.\ \underline{(\lambda\mathtt{x}.\ \mathtt{x})}\ \mathtt{z}))$$

Auswertungsstrategien

Karlsruher Institut für Technologie

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ (\lambda \mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

$$\begin{aligned} & (\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ \underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ \mathtt{z})) \\ \Rightarrow\ & (\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ \mathtt{z})) \end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

$$\begin{aligned} & (\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{z})) \\ \Rightarrow\ & (\lambda \mathrm{x}.\ \mathrm{x})\ (\underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ \mathrm{z})) \end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$

Volle β-Reduktion   Jeder Redex kann jederzeit reduziert werden.

$$\begin{aligned}
& (\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ \underline{(\lambda x.\ x)\ z}))\\
\Rightarrow\ & (\lambda x.\ x)\ (\underline{(\lambda x.\ x)\ (\lambda z.\ z)})\\
\Rightarrow\ & (\lambda x.\ x)\ (\lambda z.\ z)
\end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ (\lambda \mathtt{x}.\ \mathtt{x})\ \mathtt{z}))$$

Volle β-Reduktion   Jeder Redex kann jederzeit reduziert werden.

$$\begin{aligned} & (\lambda \mathtt{x}.\ \mathtt{x})\ ((\lambda \mathtt{x}.\ \mathtt{x})\ (\lambda \mathtt{z}.\ \underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ \mathtt{z})) \\ \Rightarrow\ & (\lambda \mathtt{x}.\ \mathtt{x})\ (\underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ (\lambda \mathtt{z}.\ \mathtt{z})) \\ \Rightarrow\ & \underline{(\lambda \mathtt{x}.\ \mathtt{x})}\ (\lambda \mathtt{z}.\ \mathtt{z}) \end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

$$
\begin{aligned}
& (\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{z})) \\
\Rightarrow\ & (\lambda \mathrm{x}.\ \mathrm{x})\ (\underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ \mathrm{z})) \\
\Rightarrow\ & \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ \mathrm{z}) \\
\Rightarrow\ & \lambda \mathrm{z}.\ \mathrm{z}
\end{aligned}
$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

$$\begin{aligned}
& (\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{z})) \\
\Rightarrow\ & (\lambda \mathrm{x}.\ \mathrm{x})\ (\underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ \mathrm{z})) \\
\Rightarrow\ & \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ \mathrm{z}) \\
\Rightarrow\ & \lambda \mathrm{z}.\ \mathrm{z} \not\Rightarrow
\end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\underline{(\lambda x.\ x)}\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$
$$\Rightarrow (\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z)$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\begin{aligned} & \underline{(\lambda x.\ x)}\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z)) \\ \Rightarrow\ & \underline{(\lambda x.\ x)}\ (\lambda z.\ (\lambda x.\ x)\ z) \end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\begin{aligned}
& \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z})) \\
\Rightarrow\ & \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}) \\
\Rightarrow\ & \lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}
\end{aligned}$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\underline{(\lambda x.\ x)}\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$
$$\Rightarrow \underline{(\lambda x.\ x)}\ (\lambda z.\ (\lambda x.\ x)\ z)$$
$$\Rightarrow \lambda z.\ \underline{(\lambda x.\ x)}\ z$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda \mathrm{x}.\ \mathrm{x})\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$
\begin{aligned}
& \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ ((\lambda \mathrm{x}.\ \mathrm{x})\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z})) \\
\Rightarrow\ & \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ (\lambda \mathrm{z}.\ (\lambda \mathrm{x}.\ \mathrm{x})\ \mathrm{z}) \\
\Rightarrow\ & \lambda \mathrm{z}.\ \underline{(\lambda \mathrm{x}.\ \mathrm{x})}\ \mathrm{z} \\
\Rightarrow\ & \lambda \mathrm{z}.\ \mathrm{z}
\end{aligned}
$$

Auswertungsstrategien

Wenn es in einem Term mehrere Redexe gibt, welchen reduziert man?

$$(\lambda x.\ x)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))$$

Volle β-Reduktion Jeder Redex kann jederzeit reduziert werden.

Normalreihenfolge Immer der linkeste äußerste Redex wird reduziert

$$\begin{aligned}
&\underline{(\lambda x.\ x)}\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda x.\ x)\ z))\\
\Rightarrow\ &\underline{(\lambda x.\ x)}\ (\lambda z.\ (\lambda x.\ x)\ z)\\
\Rightarrow\ &\lambda z.\ \underline{(\lambda x.\ x)}\ z\\
\Rightarrow\ &\lambda z.\ z \not\Rightarrow
\end{aligned}$$

Braucht man primitive Operationen?

Nicht unbedingt! Kodierung mit Funktionen höherer Ordnung:

Beispiel: `let`

$$\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \quad \text{wird zu} \quad (\lambda x.\ t_2)\ t_1$$

Beispiel: $\texttt{let x = g y in f x}$ berechnet $\texttt{f (g y)}$

$$(\lambda \texttt{x. f x})\ (\texttt{g y})$$

Braucht man primitive Operationen?

Nicht unbedingt! Kodierung mit Funktionen höherer Ordnung:

Beispiel: **let**

	$\texttt{let}\ x = t_1\ \texttt{in}\ t_2$	wird zu	$(\lambda x.\ t_2)\ t_1$
Beispiel:	$\texttt{let x = g y in f x}$	berechnet	$\texttt{f (g y)}$
	$\underline{(\lambda \texttt{x}.\ \texttt{f x})}\ (\texttt{g y})$		

Braucht man primitive Operationen?

Nicht unbedingt! Kodierung mit Funktionen höherer Ordnung:

Beispiel: `let`

	$\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$	wird zu	$(\lambda x.\, t_2)\ t_1$
Beispiel:	$\mathtt{let\ x = g\ y\ in\ f\ x}$	berechnet	$\mathtt{f\ (g\ y)}$
	$\underline{(\lambda \mathtt{x}.\ \mathtt{f\ x})}\ (\mathtt{g\ y})$	$\Rightarrow$	$\mathtt{f\ (g\ y)}$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$c_0 = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{z}$$

$$c_1 = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}\ \mathtt{z}$$

$$c_2 = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$$

$$c_3 = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\vdots$$

$$c_n = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}^n\ \mathtt{z}$$

Nachfolgerfunktion:

$$succ = \lambda \mathtt{n}.\, \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \ldots$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$c_0 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{z}$$

$$c_1 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ \mathtt{z}$$

$$c_2 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$$

$$c_3 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\vdots$$

$$c_n = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^n\ \mathtt{z}$$

Nachfolgerfunktion:

$$succ = \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \ldots$

$$succ(c_2) = (\lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$c_0 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{z}$$

$$c_1 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ \mathtt{z}$$

$$c_2 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$$

$$c_3 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\vdots$$

$$c_n = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^n\ \mathtt{z}$$

Nachfolgerfunktion:

$$succ = \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \ldots$

$$succ(c_2) = \underline{(\lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))}\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \mathtt{s}\ \mathtt{z})$$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$c_0 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{z}$$

$$c_1 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ \mathtt{z}$$

$$c_2 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$$

$$c_3 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\vdots$$

$$c_n = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^n\ \mathtt{z}$$

Nachfolgerfunktion:

$$\mathit{succ} = \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \ldots$

$$\mathit{succ}(c_2) = \underline{(\lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))}\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\underline{(\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))}\ \mathtt{s}\ \mathtt{z})$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ ((\lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \mathtt{z})$$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$
\begin{aligned}
c_0 &= \lambda s.\ \lambda z.\ z \\
c_1 &= \lambda s.\ \lambda z.\ s\ z \\
c_2 &= \lambda s.\ \lambda z.\ s\ (s\ z) \\
c_3 &= \lambda s.\ \lambda z.\ s\ (s\ (s\ z)) \\
&\ \ \vdots \\
c_n &= \lambda s.\ \lambda z.\ s^n\ z
\end{aligned}
$$

Nachfolgerfunktion:

$$succ = \lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z)$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda s.\ \lambda z.\ \ldots$

$$
\begin{aligned}
succ(c_2) &= \underline{(\lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z))}\ (\lambda s.\ \lambda z.\ s\ (s\ z)) \\
&\Rightarrow \lambda s.\ \lambda z.\ s\ (\underline{(\lambda s.\ \lambda z.\ s\ (s\ z))}\ s\ z) \\
&\Rightarrow \lambda s.\ \lambda z.\ s\ (\underline{(\lambda z.\ s\ (s\ z))}\ z) \\
&\Rightarrow \lambda s.\ \lambda z.\ s\ (s\ (s\ z))
\end{aligned}
$$

Kodierung natürlicher Zahlen

Church-Zahlen

Eine (natürliche) Zahl drückt aus, wie oft die Funktion $\mathtt{s}$ angewendet wird.

$$c_0 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{z}$$

$$c_1 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ \mathtt{z}$$

$$c_2 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$$

$$c_3 = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\vdots$$

$$c_n = \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^n\ \mathtt{z}$$

Nachfolgerfunktion:

$$succ = \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$$

$\mathtt{n}$ Church-Zahl,
d.h. von der Form $\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \ldots$

$$succ(c_2) = \underline{(\lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))}\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\underline{(\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))}\ \mathtt{s}\ \mathtt{z})$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\underline{(\lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))}\ \mathtt{z})$$

$$\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z})) = c_3$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition: *plus* $= \lambda \mathtt{m}.\, \lambda \mathtt{n}.\, \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{m\ s}\ (\mathtt{n\ s\ z})$

Multiplikation: *times* $= \lambda \mathtt{m}.\, \lambda \mathtt{n}.\, \lambda \mathtt{s}.\, \mathtt{n}\ (\mathtt{m\ s})$

$\stackrel{\eta}{=} \lambda \mathtt{m}.\, \lambda \mathtt{n}.\, \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{n}\ (\mathtt{m\ s})\ \mathtt{z}$

Potenzieren: *exp* $= \lambda \mathtt{m}.\, \lambda \mathtt{n}.\, \mathtt{n\ m}$

$\stackrel{\eta}{=} \lambda \mathtt{m}.\, \lambda \mathtt{n}.\, \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{n\ m\ s\ z}$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda m.\ \lambda n.\ \lambda s.\ \lambda z.\ m\ s\ (n\ s\ z)$
Multiplikation:	*times*	$= \lambda m.\ \lambda n.\ \lambda s.\ n\ (m\ s)$
		$\stackrel{\eta}{=} \lambda m.\ \lambda n.\ \lambda s.\ \lambda z.\ n\ (m\ s)\ z$
Potenzieren:	*exp*	$= \lambda m.\ \lambda n.\ n\ m$
		$\stackrel{\eta}{=} \lambda m.\ \lambda n.\ \lambda s.\ \lambda z.\ n\ m\ s\ z$

$$
\begin{aligned}
\mathit{plus}\ c_2\ c_3 &= (\lambda m.\ \lambda n.\ \lambda s.\ \lambda z.\ m\ s\ (n\ s\ z))\ c_2\ c_3 \\
&\stackrel{2}{\Rightarrow} \lambda s.\ \lambda z.\ c_2\ s\ (c_3\ s\ z)
\end{aligned}
$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m}\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$
Multiplikation:	*times*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})\ \mathtt{z}$
Potenzieren:	*exp*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n}\ \mathtt{m}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ \mathtt{m}\ \mathtt{s}\ \mathtt{z}$

$$\mathit{plus}\ c_2\ c_3 = (\lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m}\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))\ c_2\ c_3$$

$$\stackrel{2}{\Rightarrow} \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \mathtt{s}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z})))\ \mathtt{s}\ \mathtt{z})$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, m\ s\ (n\ s\ z)$
Multiplikation:	*times*	$= \lambda m.\, \lambda n.\, \lambda s.\, n\ (m\ s)$
		$\stackrel{\eta}{=} \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, n\ (m\ s)\ z$
Potenzieren:	*exp*	$= \lambda m.\, \lambda n.\, n\ m$
		$\stackrel{\eta}{=} \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, n\ m\ s\ z$

$$
\begin{aligned}
\mathit{plus}\ c_2\ c_3 &= (\lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, m\ s\ (n\ s\ z))\ c_2\ c_3 \\
&\stackrel{2}{\Rightarrow} \lambda s.\, \lambda z.\, \underline{(\lambda s.\, \lambda z.\, s\ (s\ z))\ s}\ ((\lambda s.\, \lambda z.\, s\ (s\ (s\ z)))\ s\ z) \\
&\Rightarrow \lambda s.\, \lambda z.\, (\lambda z.\, s\ (s\ z))\ ((\lambda s.\, \lambda z.\, s\ (s\ (s\ z)))\ s\ z)
\end{aligned}
$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m\ s\ (n\ s\ z)}$
Multiplikation:	*times*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \mathtt{n\ (m\ s)}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n\ (m\ s)\ z}$
Potenzieren:	*exp*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n\ m}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n\ m\ s\ z}$

$$
\begin{aligned}
\mathit{plus}\ c_2\ c_3 &= (\lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m\ s\ (n\ s\ z)})\ c_2\ c_3 \\
&\stackrel{2}{\Rightarrow} \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \underline{(\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s\ (s\ z)})}\ \mathtt{s}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s\ (s\ (s\ z))})\ \mathtt{s\ z}) \\
&\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \underline{(\lambda \mathtt{z}.\ \mathtt{s\ (s\ z)})}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s\ (s\ (s\ z))})\ \mathtt{s\ z}) \\
&\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s\ (s\ ((}\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s\ (s\ (s\ z)))\ s\ z))}
\end{aligned}
$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m}\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$
Multiplikation:	*times*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})\ \mathtt{z}$
Potenzieren:	*exp*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n}\ \mathtt{m}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ \mathtt{m}\ \mathtt{s}\ \mathtt{z}$

$$
\begin{aligned}
\mathit{plus}\ c_2\ c_3 &= (\lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m}\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z}))\ c_2\ c_3 \\
&\stackrel{2}{\Rightarrow} \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \underline{(\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \mathtt{s}}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z})))\ \mathtt{s}\ \mathtt{z}) \\
&\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \underline{(\lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z})))\ \mathtt{s}\ \mathtt{z}) \\
&\Rightarrow \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z})))\ \mathtt{s}\ \mathtt{z})) \\
&\stackrel{2}{\Rightarrow} \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ \mathtt{z}))))
\end{aligned}
$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, m\ s\ (n\ s\ z)$
Multiplikation:	*times*	$= \lambda m.\, \lambda n.\, \lambda s.\, n\ (m\ s)$
		$\stackrel{\eta}{=} \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, n\ (m\ s)\ z$
Potenzieren:	*exp*	$= \lambda m.\, \lambda n.\, n\ m$
		$\stackrel{\eta}{=} \lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, n\ m\ s\ z$

$$
\begin{aligned}
\textit{plus}\ c_2\ c_3 &= (\lambda m.\, \lambda n.\, \lambda s.\, \lambda z.\, m\ s\ (n\ s\ z))\ c_2\ c_3 \\
&\stackrel{2}{\Rightarrow} \lambda s.\, \lambda z.\, \underline{(\lambda s.\, \lambda z.\, s\ (s\ z))\ s}\ ((\lambda s.\, \lambda z.\, s\ (s\ (s\ z)))\ s\ z) \\
&\Rightarrow \lambda s.\, \lambda z.\, \underline{(\lambda z.\, s\ (s\ z))}\ ((\lambda s.\, \lambda z.\, s\ (s\ (s\ z)))\ s\ z) \\
&\Rightarrow \lambda s.\, \lambda z.\, s\ (s\ ((\lambda s.\, \lambda z.\, s\ (s\ (s\ z)))\ s\ z)) \\
&\stackrel{2}{\Rightarrow} \lambda s.\, \lambda z.\, s\ (s\ (s\ (s\ (s\ z)))) = c_5
\end{aligned}
$$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition: *plus* $= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m}\ \mathtt{s}\ (\mathtt{n}\ \mathtt{s}\ \mathtt{z})$

Multiplikation: *times* $= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})$

$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ (\mathtt{m}\ \mathtt{s})\ \mathtt{z}$

Potenzieren: *exp* $= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n}\ \mathtt{m}$

$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n}\ \mathtt{m}\ \mathtt{s}\ \mathtt{z}$

Rechnen mit Church-Zahlen

Arithmetische Operationen

Addition:	*plus*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{m\ s\ (n\ s\ z)}$
Multiplikation:	*times*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \mathtt{n\ (m\ s)}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n\ (m\ s)\ z}$
Potenzieren:	*exp*	$= \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n\ m}$
		$\stackrel{\eta}{=} \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{n\ m\ s\ z}$

Idee zu *exp*:

$$exp\ c_m\ c_n \stackrel{2}{\Rightarrow} c_n\ c_m = \underline{(\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^n\ \mathtt{z})}\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^m\ \mathtt{z})$$

$$\Rightarrow \lambda \mathtt{z}.\ (\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^m\ \mathtt{z})^n\ \mathtt{z}$$

$$\text{(Induktion über } n\text{)} \stackrel{\alpha\beta\eta}{=} \lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}^{m^n}\ \mathtt{z} = c_{m^n}$$

Kodierung boolescher Werte

Church-Booleans

True wird zu $c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$

False wird zu $c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  if True then x **else** y ergibt:

  $$(\lambda \mathtt{a}.\, \mathtt{a})\; (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\; x\; y$$

Kodierung boolescher Werte

Church-Booleans

`True`	wird zu	$c_{\mathrm{true}} = \lambda \mathtt{t}.\,\lambda \mathtt{f}.\,\mathtt{t}$
`False`	wird zu	$c_{\mathrm{false}} = \lambda \mathtt{t}.\,\lambda \mathtt{f}.\,\mathtt{f}$

- **`if _ then _ else _`** wird zu $\lambda \mathtt{a}.\,\mathtt{a}$

  `if True then` x **`else`** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\,\mathtt{a})}\ (\lambda \mathtt{t}.\,\lambda \mathtt{f}.\,\mathtt{t})\ x\ y \Rightarrow (\lambda \mathtt{t}.\,\lambda \mathtt{f}.\,\mathtt{t})\ x\ y$$

Kodierung boolescher Werte

Church-Booleans

$$\texttt{True} \quad \text{wird zu} \quad c_{\text{true}} = \lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{t}$$
$$\texttt{False} \quad \text{wird zu} \quad c_{\text{false}} = \lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{f}$$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\ \mathtt{a}$

  if True then x **else** y ergibt:

$$\underline{(\lambda \mathtt{a}.\ \mathtt{a})}\ (\lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{t})\ x\ y \Rightarrow \underline{(\lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{t})}\ x\ y \Rightarrow (\lambda \mathtt{f}.\ x)\ y$$

Kodierung boolescher Werte

Church-Booleans

`True`	wird zu	$c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$
`False`	wird zu	$c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **`if _ then _ else _`** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  `if True then` x **`else`** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\; x\; y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\; x\; y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)}\; y \Rightarrow x$$

Kodierung boolescher Werte

Church-Booleans

True	wird zu	$c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$
False	wird zu	$c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  if True then x **else** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\; x\; y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\; x\; y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)}\; y \Rightarrow x$$

- $\mathtt{b}_1$ && $\mathtt{b}_2$ ist äquivalent zu **if** $\mathtt{b}_1$ **then** $\mathtt{b}_2$ **else False**

$\Rightarrow$ $\mathtt{b}_1$ && $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\; \mathtt{b}_1\; \mathtt{b}_2\; c_{\text{false}}$

Kodierung boolescher Werte

Church-Booleans

True wird zu $c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$

False wird zu $c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  if True then x **else** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\, x\, y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\, x}\, y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)\, y} \Rightarrow x$$

- $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ ist äquivalent zu **if** $\mathtt{b}_1$ **then** $\mathtt{b}_2$ **else False**

$\Rightarrow$ $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\, \mathtt{b}_1\, \mathtt{b}_2\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$

Kodierung boolescher Werte

Church-Booleans

True	wird zu	$c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$
False	wird zu	$c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  if True then x **else** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\; x\; y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\; x}\; y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)\; y} \Rightarrow x$$

- $\mathtt{b}_1$ && $\mathtt{b}_2$ ist äquivalent zu **if** $\mathtt{b}_1$ **then** $\mathtt{b}_2$ **else False**

⇒ $\mathtt{b}_1$ && $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\; \mathtt{b}_1\; \mathtt{b}_2\; (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$

True && **True** ergibt:

$$(\lambda \mathtt{a}.\, \mathtt{a})\; c_{\text{true}}\; c_{\text{true}}\; (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

Kodierung boolescher Werte

Church-Booleans

`True`	wird zu	$c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$
`False`	wird zu	$c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- `if _ then _ else _` wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  `if True then` x `else` y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\, x\, y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\, x}\, y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)\, y} \Rightarrow x$$

- $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ ist äquivalent zu `if` $\mathtt{b}_1$ `then` $\mathtt{b}_2$ `else False`

$\Rightarrow$ $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\, \mathtt{b}_1\, \mathtt{b}_2\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$

`True && True` ergibt:

$$\underline{(\lambda \mathtt{a}.\, \mathtt{a})\, c_{\text{true}}}\, c_{\text{true}}\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

Kodierung boolescher Werte

Church-Booleans

True wird zu $c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$

False wird zu $c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **if _ then _ else _** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  if True then x **else** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\ (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\ x\ y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\ x\ y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)}\ y \Rightarrow x$$

- $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ ist äquivalent zu **if** $\mathtt{b}_1$ **then** $\mathtt{b}_2$ **else False**

$\Rightarrow$ $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\ \mathtt{b}_1\ \mathtt{b}_2\ (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$

True `&&` **True** ergibt:

$$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\ c_{\text{true}}\ c_{\text{true}}\ (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

$$\Rightarrow c_{\text{true}}\ c_{\text{true}}\ (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

Kodierung boolescher Werte

Church-Booleans

`True` wird zu $c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$

`False` wird zu $c_{\text{false}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f}$

- **`if _ then _ else _`** wird zu $\lambda \mathtt{a}.\, \mathtt{a}$

  `if True then` x **`else`** y ergibt:

  $$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\, x\, y \Rightarrow \underline{(\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})}\, x\, y \Rightarrow \underline{(\lambda \mathtt{f}.\, x)}\, y \Rightarrow x$$

- $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ ist äquivalent zu **`if`** $\mathtt{b}_1$ **`then`** $\mathtt{b}_2$ **`else False`**

$\Rightarrow$ $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ wird zu $(\lambda \mathtt{a}.\, \mathtt{a})\, \mathtt{b}_1\, \mathtt{b}_2\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$

`True` `&&` **`True`** ergibt:

$$\underline{(\lambda \mathtt{a}.\, \mathtt{a})}\, c_{\text{true}}\, c_{\text{true}}\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

$$\Rightarrow (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t})\, (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$$

Kodierung boolescher Werte

Church-Booleans

`True`	wird zu	$c_{\text{true}} = \lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t}$
`False`	wird zu	$c_{\text{false}} = \lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f}$

- **`if _ then _ else _`** wird zu $\lambda\mathtt{a}.\,\mathtt{a}$

  `if True then` x **`else`** y ergibt:

  $$\underline{(\lambda\mathtt{a}.\,\mathtt{a})}\,(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})\; x\; y \Rightarrow \underline{(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})}\; x\; y \Rightarrow \underline{(\lambda\mathtt{f}.\,x)}\; y \Rightarrow x$$

- $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ ist äquivalent zu **`if`** $\mathtt{b}_1$ **`then`** $\mathtt{b}_2$ **`else False`**

⇒ $\mathtt{b}_1$ `&&` $\mathtt{b}_2$ wird zu $(\lambda\mathtt{a}.\,\mathtt{a})\;\mathtt{b}_1\;\mathtt{b}_2\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})$

`True` `&&` **`True`** ergibt:

$$\underline{(\lambda\mathtt{a}.\,\mathtt{a})}\; c_{\text{true}}\; c_{\text{true}}\; (\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})$$
$$\Rightarrow \underline{(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})}\; (\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})\; (\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})$$
$$\Rightarrow (\lambda\mathtt{f}.\,(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t}))\; (\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})$$

Kodierung boolescher Werte

Church-Booleans

True wird zu $c_{\text{true}} = \lambda t.\, \lambda f.\, t$

False wird zu $c_{\text{false}} = \lambda t.\, \lambda f.\, f$

- **if _ then _ else _** wird zu $\lambda a.\, a$

  if True then x **else** y ergibt:

  $$\underline{(\lambda a.\, a)}\, (\lambda t.\, \lambda f.\, t)\, x\, y \Rightarrow \underline{(\lambda t.\, \lambda f.\, t)}\, x\, y \Rightarrow \underline{(\lambda f.\, x)}\, y \Rightarrow x$$

- b_1 && b_2 ist äquivalent zu **if** b_1 **then** b_2 **else False**

$\Rightarrow$ b_1 && b_2 wird zu $(\lambda a.\, a)\, b_1\, b_2\, (\lambda t.\, \lambda f.\, f)$

True && **True** ergibt:

$$\underline{(\lambda a.\, a)}\, c_{\text{true}}\, c_{\text{true}}\, (\lambda t.\, \lambda f.\, f)$$
$$\Rightarrow \underline{(\lambda t.\, \lambda f.\, t)}\, (\lambda t.\, \lambda f.\, t)\, (\lambda t.\, \lambda f.\, f)$$
$$\Rightarrow \underline{(\lambda f.\, (\lambda t.\, \lambda f.\, t))}\, (\lambda t.\, \lambda f.\, f) \Rightarrow \lambda t.\, \lambda f.\, t$$

Kodierung boolescher Werte

Church-Booleans

$\texttt{True}$ wird zu $c_{\text{true}} = \lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t}$

$\texttt{False}$ wird zu $c_{\text{false}} = \lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f}$

- **if _ then _ else _** wird zu $\lambda\mathtt{a}.\,\mathtt{a}$

  if True then x **else** y ergibt:

  $$\underline{(\lambda\mathtt{a}.\,\mathtt{a})\,(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})}\;x\;y \Rightarrow \underline{(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})\;x}\;y \Rightarrow \underline{(\lambda\mathtt{f}.\,x)\;y} \Rightarrow x$$

- $\mathtt{b}_1$ && $\mathtt{b}_2$ ist äquivalent zu **if** $\mathtt{b}_1$ **then** $\mathtt{b}_2$ **else False**

$\Rightarrow$ $\mathtt{b}_1$ && $\mathtt{b}_2$ wird zu $(\lambda\mathtt{a}.\,\mathtt{a})\;\mathtt{b}_1\;\mathtt{b}_2\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})$

True && **True** ergibt:

$$\begin{aligned}
&\underline{(\lambda\mathtt{a}.\,\mathtt{a})\;c_{\text{true}}}\;c_{\text{true}}\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})\\
\Rightarrow\;&\underline{(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t})}\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})\\
\Rightarrow\;&\underline{(\lambda\mathtt{f}.\,(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t}))\;(\lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{f})} \Rightarrow \lambda\mathtt{t}.\,\lambda\mathtt{f}.\,\mathtt{t} = c_{\text{true}}
\end{aligned}$$

Divergenz

Bisherige Beispiele werten zu einer Normalform aus. Aber:

$$\omega = (\lambda \mathrm{x}.\ \mathrm{x}\ \mathrm{x})\ (\lambda \mathrm{x}.\ \mathrm{x}\ \mathrm{x}) \Rightarrow (\lambda \mathrm{x}.\ \mathrm{x}\ \mathrm{x})\ (\lambda \mathrm{x}.\ \mathrm{x}\ \mathrm{x}) \Rightarrow \ldots$$

$\lambda \mathrm{x}.\ \mathrm{x}\ \mathrm{x}$ wendet sein Argument auf das Argument selbst an
$\Rightarrow$ dadurch reproduziert ω sich selbst.

Divergenz

Terme, die nicht zu einer Normalform auswerten, divergieren.
Diese modellieren unendliche Ausführungen.

Rekursive Funktionen sind Fixpunkte

Rekursive Definition von g:

$$g = \lambda \mathtt{n}. \ldots g \ldots n \ldots \text{ Rumpf verwendet } g$$

Daraus gewinnt man das Funktional

$$G = \lambda \mathtt{g}.\, \lambda \mathtt{n}. \ldots g \ldots n \ldots$$

Falls G einen Fixpunkt g^* hat, d.h. $G(g^*) = g^*$, so

$$\begin{array}{lcl} g^* & = G(g^*) = & \lambda \mathtt{n}. \ldots g^* \ldots n \ldots \\ \text{Vergleiche:} \quad g & = & \lambda \mathtt{n}. \ldots g \ \ldots n \ldots \end{array}$$

Rekursive Definition $\Leftrightarrow$ Fixpunkt des Funktionals

Beispiel: Fakultät

```
g =          \ n -> if isZero n then 1 else n * g (n - 1)   -- rekursiv
G = \ g -> \ n -> if isZero n then 1 else n * g (n - 1)   -- Funktional
```

Rekursionsoperator

Rekursionsoperator

$$\mathsf{Y} = \lambda \mathtt{f}.\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$$

$$\begin{aligned}
\mathsf{Y}\ f &= \underline{(\lambda \mathtt{f}.\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x}\ \mathtt{x})))}\ f \\
&\Rightarrow \underline{(\lambda \mathtt{x}.\ f\ (\mathtt{x}\ \mathtt{x}))}\ (\lambda \mathtt{x}.\ f\ (\mathtt{x}\ \mathtt{x})) \\
&\Rightarrow f\ ((\lambda \mathtt{x}.\ f\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\ f\ (\mathtt{x}\ \mathtt{x}))) \Leftarrow f\ (\mathsf{Y}\ f)
\end{aligned}$$

also $\quad f\ (\mathsf{Y}\ f) \stackrel{\beta}{=} \mathsf{Y}\ f$

d.h. $\quad$ **Y *f* ist Fixpunkt von *f*.**

Turing-Mächtigkeit

Der untypisierte λ-Kalkül ist turing-mächtig.

Beispiel: Fakultät im λ-Kalkül

```
fak =          \ n -> if isZero n then 1 else n * fak (n - 1)
G   = \ fak -> \ n -> if isZero n then 1 else n * fak (n - 1)
```

$$\mathsf{Y} = \lambda \mathtt{f}.\, (\lambda \mathtt{x}.\, \mathtt{f}\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, \mathtt{f}\,(\mathtt{x}\,\mathtt{x}))$$

$$G = \lambda \mathtt{fak}.\, \lambda \mathtt{n}.\, (\lambda \mathtt{a}.\, \mathtt{a})\,(\mathit{isZero}\,\mathtt{n})\; c_1\; (\mathit{times}\,\mathtt{n}\,(\mathtt{fak}\,(\mathit{sub}\,\mathtt{n}\,c_1)))$$

$$\mathit{fak} = \mathsf{Y}\, G$$

$$\begin{aligned}
\mathit{fak}\; c_2 = \mathsf{Y}\, G\, c_2 &\Rightarrow (\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\; c_2 \\
&\Rightarrow G\,((\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x})))\; c_2 \\
&\overset{2}{\Rightarrow} (\lambda \mathtt{a}.\, \mathtt{a})\,(\mathit{isZero}\; c_2)\; c_1\; (\mathit{times}\; c_2\; ((\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\mathit{sub}\; c_2\; c_1))) \\
&\overset{*}{\Rightarrow} \mathit{times}\; c_2\; ((\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\mathit{sub}\; c_2\; c_1)) \\
&\overset{*}{\Rightarrow} \mathit{times}\; c_2\; (\overbrace{(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))}^{\mathsf{Y}\, G \Rightarrow}\; c_1) \\
&\overset{3}{\Rightarrow} \mathit{times}\; c_2\, ((\lambda \mathtt{a}.\, \mathtt{a})\,(\mathit{isZero}\; c_1)\, c_1\, (\mathit{times}\; c_1\, ((\lambda \mathtt{x}.\, G(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G(\mathtt{x}\,\mathtt{x}))\;(\mathit{sub}\; c_1\; c_1)))) \\
&\overset{*}{\Rightarrow} \mathit{times}\; c_2\; (\mathit{times}\; c_1\; ((\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\lambda \mathtt{x}.\, G\,(\mathtt{x}\,\mathtt{x}))\,(\mathit{sub}\; c_1\; c_1))) \\
&\overset{*}{\Rightarrow} \mathit{times}\; c_2\; (\mathit{times}\; c_1\; ((\mathit{is_zero}\; c_0)\; c_1 \ldots)) \overset{*}{\Rightarrow} c_2
\end{aligned}$$

Church-Rosser-Eigenschaft

Satz (Church-Rosser)

Der untypisierte λ-Kalkül ist konfluent:
Wenn $t \overset{*}{\Rightarrow} t_1$ und $t \overset{*}{\Rightarrow} t_2$,
dann gibt es ein t' mit $t_1 \overset{*}{\Rightarrow} t'$ und $t_2 \overset{*}{\Rightarrow} t'$.

$$
\begin{tikzcd}
 & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 \arrow[dr, dashed, "*"'] & & t_2 \arrow[dl, dashed, "*"] \\
 & t' &
\end{tikzcd}
$$

Beispiel

<u>überlappende Redexe</u>

$$
\begin{tikzcd}
 & \underline{(\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}\ \mathtt{x})}\ ((\lambda \mathtt{y}.\ \mathtt{y})\ \mathtt{z}) \arrow[dl] \arrow[dr] & \\
\mathtt{f}\ (\underline{(\lambda \mathtt{y}.\ \mathtt{y})}\ \mathtt{z})\ ((\lambda \mathtt{y}.\ \mathtt{y})\ \mathtt{z}) \arrow[d] & & (\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}\ \mathtt{x})\ \mathtt{z} \arrow[ddl] \\
\mathtt{f}\ \mathtt{z}\ (\underline{(\lambda \mathtt{y}.\ \mathtt{y})}\ \mathtt{z}) \arrow[dr] & & \\
 & \mathtt{f}\ \mathtt{z}\ \mathtt{z} &
\end{tikzcd}
$$

Church-Rosser-Eigenschaft

Satz (Church-Rosser)

Der untypisierte λ-Kalkül ist konfluent:
Wenn $t \overset{*}{\Rightarrow} t_1$ und $t \overset{*}{\Rightarrow} t_2$,
dann gibt es ein t' mit $t_1 \overset{*}{\Rightarrow} t'$ und $t_2 \overset{*}{\Rightarrow} t'$.

$$
\begin{tikzcd}
 & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 \arrow[dr, dashed, "*"'] & & t_2 \arrow[dl, dashed, "*"] \\
 & t' &
\end{tikzcd}
$$

Beispiel

<u>überlappende Redexe</u>

$$
\begin{tikzcd}[column sep=tiny]
 & \texttt{(}\lambda\texttt{x. f x x) (}\underline{\texttt{(}\lambda\texttt{y. y)}}\texttt{ z)} \arrow[dl] \arrow[dr] & \\
\texttt{f ((}\lambda\texttt{y. y) z) ((}\lambda\texttt{y. y) z)} \arrow[d] & & \underline{\texttt{(}\lambda\texttt{x. f x x)}}\texttt{ z} \arrow[ddl] \\
\texttt{f z ((}\lambda\texttt{y. y) z)} \arrow[dr] & & \\
 & \texttt{f z z} &
\end{tikzcd}
$$

Church-Rosser-Eigenschaft

Satz (Church-Rosser)

Der untypisierte λ-Kalkül ist konfluent:
Wenn $t \overset{*}{\Rightarrow} t_1$ und $t \overset{*}{\Rightarrow} t_2$,
dann gibt es ein t' mit $t_1 \overset{*}{\Rightarrow} t'$ und $t_2 \overset{*}{\Rightarrow} t'$.

$$\begin{tikzcd} & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\ t_1 \arrow[dr, dashed, "*"'] & & t_2 \arrow[dl, dashed, "*"] \\ & t' & \end{tikzcd}$$

Beispiel

überlappende Redexe

$$\begin{tikzcd} & \texttt{(}\lambda\texttt{x. f x x) (}\underline{\texttt{(}\lambda\texttt{y. y)}}\texttt{ z)} \arrow[dl] \arrow[dr] & \\ \texttt{f ((}\lambda\texttt{y. y) z) ((}\lambda\texttt{y. y) z)} \arrow[d] & & \underline{\texttt{(}\lambda\texttt{x. f x x)}}\texttt{ z} \arrow[ddl] \\ \texttt{f z ((}\lambda\texttt{y. y) z)} \arrow[dr] & & \\ & \texttt{f z z} & \end{tikzcd}$$

Beweisidee: Definiere $\twoheadrightarrow$ als „parallele“ β-Reduktion.

- Es gilt: $\Rightarrow \subseteq \twoheadrightarrow \subseteq \overset{*}{\Rightarrow}$.
- Zeige Diamant-Eigenschaft für $\twoheadrightarrow$.

$$\begin{tikzcd} & t \arrow[dl, two heads] \arrow[dr, two heads] & \\ t_1 \arrow[dr, dashed, two heads] & & t_2 \arrow[dl, dashed, two heads] \\ & t' & \end{tikzcd}$$

Eindeutigkeit der Normalform

Korollar (Eindeutigkeit der Normalform)

Die Normalform eines λ-Terms t ist – sofern sie existiert – eindeutig.

Beweis:

- t_1 und t_2 Normalformen von t, d.h. $t \stackrel{*}{\Rightarrow} t_1 \not\Rightarrow$ und $t \stackrel{*}{\Rightarrow} t_2 \not\Rightarrow$.
- Nach Church-Rosser gibt es t' mit $t_1 \stackrel{*}{\Rightarrow} t'$ und $t_2 \stackrel{*}{\Rightarrow} t'$.
- Nach Annahme $t_1 \not\Rightarrow$ und $t_2 \not\Rightarrow$, also $t_1 = t' = t_2$.

Bei β-Reduktionen ist irrelevant, welchen Redex man zuerst reduziert.

Auswertung in Programmiersprachen

Werte in Haskell:

- Primitive Werte: `2`, **`True`**
- Funktionen: `(\x -> x)`, `(&&)`, `(\x -> (\y -> y+y) x)`

Werte im λ-Kalkül:

- Abstraktionen: $c_2 = \lambda \mathtt{s}.\, \lambda \mathtt{z}.\, \mathtt{s}\ (\mathtt{s}\ \mathtt{z})$, $c_{\text{true}} = \lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{t}$
  $\lambda \mathtt{x}.\, \mathtt{x}$, $\lambda \mathtt{b}_1.\, \lambda \mathtt{b}_2.\, \mathtt{b}_1\ \mathtt{b}_2\ (\lambda \mathtt{t}.\, \lambda \mathtt{f}.\, \mathtt{f})$,
  $\lambda \mathtt{x}.\, (\lambda \mathtt{y}.\, \mathit{plus}\ \mathtt{y}\ \mathtt{y})\ \mathtt{x}$

Auswertungsstrategie: Keine weitere Reduzierung von Werten

$\Rightarrow$ Reduziere keine Redexe unter Abstraktionen (umgeben von λ): call-by-name, call-by-value

Call-By-Name

Call-by-name Reduziere linkesten äußersten Redex

- Aber nicht falls von einem λ umgeben

$$\underline{(\lambda y.\,(\lambda x.\,y\;(\lambda z.\,z)\;x))}\;((\lambda x.\,x)\;(\lambda y.\,y))$$
$$\Rightarrow (\lambda x.\,(\underline{(\lambda x.\,x)\;(\lambda y.\,y)})\;(\lambda z.\,z)\;x) \not\Rightarrow$$

Intuition: Reduziere Argumente erst, wenn benötigt

Auswertung in Haskell: Lazy-Evaluation = call-by-name + sharing

- Standard-Auswertungsstrategie für Funktionen/Konstruktoren

```
listOf x = x : listOf x
3 : listOf 3 ⇏

      (div 1 0):(6:[])    ⇏
tail ((div 1 0):(6:[]))   ⇒ 6:[] ⇏
```

Call-By-Value

Call-by-value Reduziere linkesten Redex

- der nicht von einem λ umgeben
- und dessen Argument ein Wert ist

$$\underline{(\lambda y.\,(\lambda x.\,y\,(\lambda z.\,z)\,x))}\,(\underline{(\lambda x.\,x)}\,(\lambda y.\,y))$$
$$\Rightarrow \underline{(\lambda y.\,(\lambda x.\,y\,(\lambda z.\,z)\,x))}\,(\lambda y.\,y)$$
$$\Rightarrow (\lambda x.\,\underline{(\lambda y.\,y)}\,(\lambda z.\,z)\,x) \not\Rightarrow$$

Intuition: Argumente vor Funktionsaufruf auswerten
Auswertungsstrategie vieler Sprachen: Java, C, Scheme, ML, ...

Arithmetik in Haskell: Auswertung by-value

```
prodOf x = x * prodOf x
3 + prodOf 3 ⇒ 3 + (3 + prodOf 3) ⇒ 3 + (3 + (3 + prodOf 3)) ⇒ ...

((div 1 0)*6)*0  ⇒ ⊥
((div 2 2)*6)*0  ⇒ (1*6)*0 ⇒ 6*0 ⇒ 0
```

Vergleich der Auswertungsstrategien

call-by-name und call-by-value:

- Werten nicht immer zur Normalform aus: $\lambda x.\ (\lambda y.\ y)\ x$
- Gibt es Normalform, dann darauf β-reduzierbar (Church-Rosser)
- Call-by-name terminiert öfter

$$
\begin{aligned}
\mathsf{Y}\ (\lambda y.\ z) &= \underline{\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))}\ (\lambda y.\ z) \\
&\Rightarrow \underline{\lambda x.\ (\lambda y.\ z)\ (x\ x)}\ (\lambda x.\ (\lambda y.\ z)\ (x\ x)) \\
&\Rightarrow (\lambda y.\ z)\ ((\lambda x.\ (\lambda y.\ z)\ (x\ x))\ (\lambda x.\ (\lambda y.\ z)\ (x\ x)))
\end{aligned}
$$

Standardisierungssatz

Wenn *t* eine Normalform hat, dann findet Normalreihenfolgenauswertung diese.

Vergleich der Auswertungsstrategien

call-by-name und call-by-value:

- Werten nicht immer zur Normalform aus: $\lambda x.\ (\lambda y.\ y)\ x$
- Gibt es Normalform, dann darauf β-reduzierbar (Church-Rosser)
- Call-by-name terminiert öfter

$$
\begin{aligned}
\mathsf{Y}\ (\lambda y.\ z) &= \underline{\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))}\ (\lambda y.\ z) \\
&\Rightarrow \underline{\lambda x.\ (\lambda y.\ z)\ (x\ x)}\ (\lambda x.\ (\lambda y.\ z)\ (x\ x)) \\
&\Rightarrow \underline{(\lambda y.\ z)}\ ((\lambda x.\ (\lambda y.\ z)\ (x\ x))\ (\lambda x.\ (\lambda y.\ z)\ (x\ x))) \overset{cbn}{\Rightarrow} z
\end{aligned}
$$

Standardisierungsatz

Wenn *t* eine Normalform hat, dann findet Normalreihenfolgenauswertung diese.

Vergleich der Auswertungsstrategien

call-by-name und call-by-value:

- Werten nicht immer zur Normalform aus: $\lambda x.\ (\lambda y.\ y)\ x$
- Gibt es Normalform, dann darauf β-reduzierbar (Church-Rosser)
- Call-by-name terminiert öfter

$$
\begin{aligned}
\mathsf{Y}\ (\lambda y.\ z) = {} & \underline{\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))}\ (\lambda y.\ z) \\
\Rightarrow {} & \underline{\lambda x.\ (\lambda y.\ z)\ (x\ x)}\ (\lambda x.\ (\lambda y.\ z)\ (x\ x)) \\
\Rightarrow {} & (\lambda y.\ z)\ (\underline{(\lambda x.\ (\lambda y.\ z)\ (x\ x))}\ (\lambda x.\ (\lambda y.\ z)\ (x\ x))) \\
\overset{\text{cbv}}{\Rightarrow} {} & (\lambda y.\ z)\ ((\lambda y.\ z)\ (\underline{(\lambda x.\ (\lambda y.\ z)\ (x\ x))}\ (\lambda x.\ (\lambda y.\ z)\ (x\ x)))) \\
\overset{\text{cbv}}{\Rightarrow} {} & \ldots
\end{aligned}
$$

Standardisierungsatz

Wenn *t* eine Normalform hat, dann findet Normalreihenfolgenauswertung diese.