Institut für Programmstrukturen und Datenorganisation
Prof. Dr.-Ing. Gregor Snelting
gregor.snelting@kit.edu
Prof. Dr. Ralf Reussner
reussner@kit.edu

Programmierparadigmen – WS 2021/22

https://pp.ipd.kit.edu/lehre/WS202122/paradigmen/uebung

Blatt 5: Rekursionsoperatoren, Typprüfung

Abgabe: 26.11.2021, 14:00
Besprechung: 29.11. – 30.11.2021

Reichen Sie Ihre Abgabe bis zum 26.11.2021 um 14:00 in unserer Praktomat-Instanz unter https://praktomat.cs.kit.edu/pp_2021_WS ein.

Hinweis: Im Rahmen von PSE wurde in einem vorherigen Semester ein Lerntool entwickelt, welches λ-Ausdrücke reduzieren und strukturell darstellen kann. Dieses kann Ihnen beim Lösen dieser Aufgaben behilflich sein.[1]

1 Klammerung im λ-Kalkül [größtenteils aus "To Mock a Mockingbird" von Raymond Smullyan]

1. Fügen Sie in die folgenden Ausdrücke alle impliziten Klammern ein!

   (a) $c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5$
   (b) $(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5)$
   (c) $c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6)$
   (d) $c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6$
   (e) $c_0\ (c_1\ (c_2\ c_3\ c_4))\ c_5\ c_6$
   (f) $(\lambda y.\ c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5)$
   (g) $(\lambda y.\ c_0\ (\lambda z.\ c_1\ c_2))\ (c_3\ c_4\ c_5)$

   Beispiellösung:

   (a) $((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5$
   (b) $((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5)$
   (c) $((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6)$
   (d) $(((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5)\ c_6$
   (e) $((c_0\ (c_1\ ((c_2\ c_3)\ c_4)))\ c_5)\ c_6$
   (f) $(\lambda y.\ ((c_0\ c_1)\ c_2))\ ((c_3\ c_4)\ c_5)$
   (g) $(\lambda y.\ (c_0\ (\lambda z.\ (c_1\ c_2))))\ ((c_3\ c_4)\ c_5)$

2. Welcher dieser beiden λ-Terme repräsentiert den gleichen λ-Term wie $\lambda y.\ y\ c_0$?

   (a) $(\lambda y.\ y)\ c_0$

[1] Siehe https://pp.ipd.kit.edu/lehre/WS201718/paradigmen/lambda-ide/Wavelength.html. Das Tool ist zudem auf der Webseite der Übung verlinkt

(b) $\lambda y.\ (y\ c_0)$

Beispiellösung: Der zweite.

(a) Führen Sie in folgenden Termen Substitution durch:

i. $x = \ \lambda y.\ y$ in den Term $(x)\ c_0$

ii. $x = (\lambda y.\ y)$ in den Term $x\ \ c_0$

(b) Gilt folgende Aussage im λ-Kalkül?

"Für beliebiges t repräsentieren t und (t) den gleichen λ-Term"

(c) Führen Sie in folgendem Term Substitution durch:[2]

iii. $x = \ \lambda y.\ y$ in den Term $x\ \ c_0$

Beispiellösung: Substitution in λ-Ausdrücken ist eine *strukturelle* Operation, keine lexikalische. Manchmal müssen Klammern ergänzt werden, um die Struktur des resultierenden Ausdrucks richtig darzustellen, andere Male können Klammern weggelassen werden, weil sie schon aus den Assoziativitätsregeln der Terme folgen.
Dementsprechend stimmt die zitierte Aussage, und alle drei Substitutionen resultieren im gleichen Term, nämlich $(\lambda y.\ y)\ c_0$.

3. Angenommen, $x = c_0\ c_1$. Welche der folgenden Aussagen gelten?

(a) $c_0\ c_1\ c_2 = x\ c_2$

(b) $c_2\ c_0\ c_1 = c_2\ x$

(c) $c_2\ (c_3\ c_4)\ c_0\ c_1 = c_2\ (c_3\ c_4)\ x$

(d) $c_2\ (c_0\ c_1\ c_3)\ c_4 = c_2\ (x\ c_3)\ c_4$

Beispiellösung: Es gelten die Aussagen 1 und 4.

4. Unterstreichen Sie alle *linken Seiten* der Redexe (also die "sofort anwendbaren Funktionen") in folgendem Term:

$$(\lambda a.a)\ (\lambda b.b)\ ((\lambda c.c)\ ((\lambda d.d)\ (\lambda e.e)\ (\lambda f.f)))\ (\lambda g.g)\ ((\lambda h.h)\ (\lambda i.i)).$$

Führen Sie dann, jeweils ausgehend von obigem Term, je einen Reduktionsschritt pro Redex durch.

Statt zu unterstreichen können Sie auch einfach den Variablennamen der zugehörigen Lambda-Abstraktion angeben.

Beispiellösung:

$$\underline{(\lambda a.a)}\ (\lambda b.b)\ (\underline{(\lambda c.c)}\ (\underline{(\lambda d.d)}\ (\lambda e.e)\ (\lambda f.f)))\ (\lambda g.g)\ (\underline{(\lambda h.h)}\ (\lambda i.i)).$$

Warum $(\lambda g.g)$ nicht unterstrichen gehört ist vielleicht leichter zu sehen, wenn man die großen geklammerten Subterme weglässt:

$$\underline{(\lambda a.a)}\ (\lambda b.b)\ (...)\ (\lambda g.g)\ (...).$$

Da Applikation linksassoziativ ist, ist $(\lambda g.g)$ also die *rechte* Seite einer Applikation.

(a) $(\lambda b.b)\ ((\lambda c.c)\ ((\lambda d.d)\ (\lambda e.e)\ (\lambda f.f)))\ (\lambda g.g)\ ((\lambda h.h)\ (\lambda i.i))$

(b) $(\lambda a.a)\ (\lambda b.b)\ ((\lambda d.d)\ (\lambda e.e)\ (\lambda f.f))\ (\lambda g.g)\ ((\lambda h.h)\ (\lambda i.i))$

(c) $(\lambda a.a)\ (\lambda b.b)\ ((\lambda c.c)\ ((\lambda e.e)\ (\lambda f.f)))\ (\lambda g.g)\ ((\lambda h.h)\ (\lambda i.i))$

(d) $(\lambda a.a)\ (\lambda b.b)\ ((\lambda c.c)\ ((\lambda d.d)\ (\lambda e.e)\ (\lambda f.f)))\ (\lambda g.g)\ (\lambda i.i)$

[2] Betrachten Sie noch einmal die vorherigen Teilaufgaben. Haben Sie konsistent geantwortet?

2 Redexe, Auswertungsstrategie

Markieren Sie in den folgenden λ-Termen alle vorkommenden Redexe. Welcher Redex wird unter den Auswertungsstrategien *Normalreihenfolge*, *Call-By-Name* bzw. *Call-By-Value* jeweils als nächstes reduziert?

$t_1 =$ (λt. λf. f) ((λy. (λx. x x) (λx. x x)) ((λx. x) (λx. x))) (λt. λf. f)

$t_2 =$ λy. (λz. (λx. x) (λx. x) z) y

Sie können diese Aufgabe eingescannt, abfotografiert oder z.B. als ASCII-Art abgeben:

```
(\x. x) ((\y. y) z)
         ------- ~  CBV
------- ~~~~~~~~~~ NRF, CBN
```

In den meisten PDF-Readern sollten Sie die Terme auch 1:1 aus den Aufgaben herauskopieren können.

Beispiellösung:

(λt. λf. f) ((λy. (λx. x x) (λx. x x)) ((λx. x) (λx. x))) (λt. λf. f) Normalenreihenfolge und Call-By-Name

(λt. λf. f) ((λy. (λx. x x) (λx. x x)) ((λx. x) (λx. x))) (λt. λf. f)

(λt. λf. f) ((λy. (λx. x x) (λx. x x)) ((λx. x) (λx. x))) (λt. λf. f)

(λt. λf. f) ((λy. (λx. x x) (λx. x x)) ((λx. x) (λx. x))) (λt. λf. f) Call-By-Value

Bemerkung: t_1 entspricht ungefähr dem Haskell-Term

```
t1 = False && ((\y -> loop) (id id))
  where loop = loop
        id x = x
```

Wie unter Call-By-Name wird hier das zweite Argument von `&&` nie ausgewertet.

λy. (λz. (λx. x) (λx. x) z) y Normalenreihenfolge

λy. (λz. (λx. x) (λx. x) z) y

Bemerkung: t_2 entspricht einer Funktion (*Wert!*), nur Normalenreihenfolge „vereinfacht“ diese weiter

3 Church-Zahlen in Haskell [Klausuraufgabe vom WS 2010/11] [10 Punkte]

Reichen Sie Ihre Lösung als Modul `ChurchNumbers` ein.

Bekanntlich kann man die natürlichen Zahlen im λ-Kalkül codieren als:

$$c_n \equiv \lambda s.\ \lambda z.\ \underbrace{s\ (\ldots\ s\ (s\ z))}_{\text{n mal}}$$

Diese Church-Zahlen haben den (polymorphen) Typ

```
type Church t = (t -> t) -> t -> t.
```

Geben Sie zwei Haskellfunktionen

```
int2church :: Integer -> Church t
church2int :: Church Integer -> Integer
```

an, die gewöhnliche natürliche Zahlen in Church-Zahlen verwandeln und umgekehrt.
Hinweis: `church2int` lässt sich ohne Rekursion definieren.

Beispiellösung:

```
int2church :: Integer -> Church t
int2church 0 = \s z -> z
int2church n = \s z -> int2church (n - 1) s (s z)

church2int :: Church Integer -> Integer
church2int n = n (+1) 0
```

Alternativlösung für `int2church`:

```
int2church' n = \s z -> iterate s z !! fromIntegral n
```

4 Church-Paare [weitgehend aus alten Klausuraufgaben]

Aus der Vorlesung kennen Sie die Modellierung von Church-Zahlen und Church-Booleans. Auf ähnliche Weise kann man auch Paare von Elementen im λ-Kalkül darstellen. Beispielsweise lässt sich das Paar $(3, 5)$ (unter Verwendung von Church-Zahlen c_n) darstellen als

$$\lambda \mathtt{f}.\ \mathtt{f}\ c_3\ c_5$$

Der Paar-Konstruktor *pair* ist damit:

$$\mathit{pair} = \lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \lambda \mathtt{f}.\ \mathtt{f}\ \mathtt{a}\ \mathtt{b}$$

Wichtige Funktionen auf Paaren sind die Destruktoren *fst* und *snd*, die jeweils das erste bzw. zweite Element aus dem Paar extrahieren. Die Definition von *fst* ist:

$$\mathit{fst} = \lambda \mathtt{p}.\ \mathtt{p}\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a})$$

1. Definieren Sie *snd*. [1 Punkt]

   Beispiellösung:

   $$\mathit{snd} = \lambda \mathtt{p}.\ \mathtt{p}\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{b})$$

2. Zeigen Sie mit Beta-Reduktion unter Verwendung von Call-By-Name: [4 Punkte]

$$\mathit{fst}\ (\mathit{pair}\ a\ b) \Rightarrow^* a$$

Beispiellösung:

$$\begin{array}{rcl}
\mathit{fst}\ (\mathit{pair}\ a\ b) & = & (\lambda \mathtt{p}.\ \mathtt{p}\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a}))\ ((\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \lambda \mathtt{f}.\ \mathtt{f}\ \mathtt{a}\ \mathtt{b})\ a\ b) \\
& \Rightarrow & (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \lambda \mathtt{f}.\ \mathtt{f}\ \mathtt{a}\ \mathtt{b})\ a\ b\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a}) \\
& \Rightarrow & (\lambda \mathtt{b}.\ \lambda \mathtt{f}.\ \mathtt{f}\ a\ \mathtt{b})\ b\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a}) \\
& \Rightarrow & (\lambda \mathtt{f}.\ \mathtt{f}\ a\ b)\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a}) \\
& \Rightarrow & (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{a})\ a\ b \\
& \Rightarrow & (\lambda \mathtt{b}.\ a)\ b \\
& \Rightarrow & a
\end{array}$$

3. Geben Sie einen Lambdaterm *next* an, der zur Church-Darstellung von (n, m) die Church-Darstellung von $(m, m+1)$ berechnet. [3 Punkte]

Beispiellösung:
$\mathit{next} = \lambda \mathtt{p}.\ \mathit{pair}\ (\mathit{snd}\ \mathtt{p})\ (\mathit{succ}\ (\mathit{snd}\ \mathtt{p}))$

Oder äquivalent z.B.:
$\mathit{next} = \lambda \mathtt{p}.\ (\lambda \mathtt{m}.\ \mathit{pair}\ \mathtt{m}\ (\mathit{succ}\ \mathtt{m}))\ (\mathtt{p}\ (\lambda \mathtt{a}.\ \lambda \mathtt{b}.\ \mathtt{b}))$

4. Zeigen Sie, dass für beliebige Terme n, m gilt: $\mathit{next}\ (\mathit{pair}\ n\ m) \Rightarrow^* \mathit{pair}\ m\ (\mathit{succ}\ m)$ [2 Punkte]

Beispiellösung:

$$\begin{array}{rcl}
\mathit{next}\ (\mathit{pair}\ n\ m) & \Rightarrow & \mathit{pair}\ (\mathit{snd}\ (\mathit{pair}\ n\ m))\ (\mathit{succ}\ (\mathit{snd}\ (\mathit{pair}\ n\ m))) \\
& \Rightarrow^* & \mathit{pair}\ m\ (\mathit{succ}\ (\mathit{snd}\ (\mathit{pair}\ n\ m))) \\
& \Rightarrow^* & \mathit{pair}\ m\ (\mathit{succ}\ m)
\end{array}$$

5. Verwenden Sie *next*, um eine Vorgängerfunktion *pred* für Church-Zahlen anzugeben. [5 Punkte]
Hinweis: n-malige Anwendung von *next* auf $\mathit{pair}\ c_0\ c_0$ ergibt $\mathit{pair}\ c_{n-1}\ c_n$

Beispiellösung:
$\mathit{pred} = \lambda \mathtt{n}.\ \mathit{fst}\ (\mathtt{n}\ \mathit{next}\ (\mathit{pair}\ c_0\ c_0))$

6. Berechnen Sie den Vorgänger von c_2, indem Sie die zugehörige Beta-Reduktion angeben. [5 Punkte]
Beispiellösung:

$$\begin{array}{rcl}
\mathit{pred}\ c_2 & \Rightarrow & \mathit{fst}\ (c_2\ \mathit{next}\ (\mathit{pair}\ c_0\ c_0)) \\
& = & \mathit{fst}\ ((\lambda \mathtt{s}.\ \lambda \mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \mathit{next}\ (\mathit{pair}\ c_0\ c_0)) \\
& \Rightarrow^* & \mathit{fst}\ (\mathit{next}\ (\mathit{next}\ (\mathit{pair}\ c_0\ c_0))) \\
& \Rightarrow^* & \mathit{fst}\ (\mathit{next}\ (\mathit{pair}\ c_0\ (\mathit{succ}\ c_0))) \\
& \Rightarrow^* & \mathit{fst}\ (\mathit{pair}\ (\mathit{succ}\ c_0)\ (\mathit{succ}\ (\mathit{succ}\ c_0))) \\
& \Rightarrow^* & \mathit{succ}\ c_0 \\
& \Rightarrow^* & c_1
\end{array}$$

7. Definieren Sie eine Subtraktionsfunktion *sub*, die zwei Church-Zahlen voneinander abzieht. Was ist die Normalform von $\mathit{sub}\ c_2\ c_1$? [Nicht Teil der Klausur]

Beispiellösung:

$$\mathit{sub} = \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ \mathtt{n}\ \mathit{pred}\ \mathtt{m}$$
$$\mathit{sub}\ c_2\ c_1 \overset{2}{\Rightarrow} c_1\ \mathit{pred}\ c_2 \overset{2}{\Rightarrow} \mathit{pred}\ c_2 \overset{*}{\Rightarrow} c_1$$

5 B-Seite: Omega and beyond...

Aus der Vorlesung kennen Sie bereits $\omega = (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})$, einen divergierenden Term. In dieser Aufgabe befassen wir uns näher mit solchen Termen.

1. Führen Sie volle β-Reduktion für folgende Terme aus:

   (a) $(\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{m}\ \mathtt{y}\ \mathtt{n})$

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

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

   Beispiellösung:

   (a)
   $$\begin{aligned} & (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{m}\ \mathtt{y}\ \mathtt{n}) \\ \Rightarrow\ & (\lambda \mathtt{y}.\ \mathtt{m}\ \mathtt{y}\ \mathtt{n})\ (\lambda \mathtt{y}.\ \mathtt{m}\ \mathtt{y}\ \mathtt{n}) \\ \Rightarrow\ & \mathtt{m}\ (\lambda \mathtt{y}.\ \mathtt{m}\ \mathtt{y}\ \mathtt{n})\ \mathtt{n} \end{aligned}$$

   (b)
   $$\begin{aligned} & (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{f}\ \mathtt{y}) \\ \Rightarrow\ & (\lambda \mathtt{y}.\ \mathtt{f}\ \mathtt{y})\ (\lambda \mathtt{y}.\ \mathtt{f}\ \mathtt{y}) \\ \Rightarrow\ & \mathtt{f}\ (\lambda \mathtt{y}.\ \mathtt{f}\ \mathtt{y}) \end{aligned}$$

   (c)
   $$\begin{aligned} & (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ (\mathtt{f}\ \mathtt{y})\ \mathtt{y}) \\ \Rightarrow\ & (\lambda \mathtt{y}.\ (\mathtt{f}\ \mathtt{y})\ \mathtt{y})\ (\lambda \mathtt{y}.\ (\mathtt{f}\ \mathtt{y})\ \mathtt{y}) \\ \Rightarrow\ & \mathtt{f}\ (\lambda \mathtt{y}.\ (\mathtt{f}\ \mathtt{y})\ \mathtt{y})\ (\lambda \mathtt{y}.\ (\mathtt{f}\ \mathtt{y})\ \mathtt{y}) \end{aligned}$$

2. (a) Führen Sie 4 β-Reduktionsschritte für folgenden Term aus:

   $$\omega' = (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{m}\ (\mathtt{y}\ \mathtt{y})\ \mathtt{n})$$

   Hinweis: Sparen Sie sich Schreibarbeit, indem Sie sich Abkürzungen für wiederholte Subterme definieren.

   Beispiellösung: Sei $\theta = \lambda \mathtt{y}.\ \mathtt{m}\ (\mathtt{y}\ \mathtt{y})\ \mathtt{n}$:
   $$\begin{aligned} \omega' = \ & (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{m}\ (\mathtt{y}\ \mathtt{y})\ \mathtt{n}) \\ \Rightarrow\ & \theta\ \theta\ =\ (\lambda \mathtt{y}.\ \mathtt{m}\ (\mathtt{y}\ \mathtt{y})\ \mathtt{n})\ (\lambda \mathtt{y}.\ \mathtt{m}\ (\mathtt{y}\ \mathtt{y})\ \mathtt{n}) \\ \Rightarrow\ & \mathtt{m}\ (\theta\ \theta)\ \mathtt{n} \\ \Rightarrow\ & \mathtt{m}\ (\mathtt{m}\ (\theta\ \theta)\ \mathtt{n})\ \mathtt{n} \\ \Rightarrow\ & \mathtt{m}\ (\mathtt{m}\ (\mathtt{m}\ (\theta\ \theta)\ \mathtt{n})\ \mathtt{n})\ \mathtt{n} \end{aligned}$$

   (b) Es sei

   $$\mathit{count} = \lambda \mathtt{y}.\lambda \mathtt{n}.\ \mathit{isZero}\ \mathtt{n}\ c_1\ ((\mathtt{y}\ \mathtt{y})\ (\mathit{sub}\ \mathtt{n}\ c_1)).$$

   Reduzieren Sie folgenden Term vollständig unter Verwendung der Call-by-Name-Reihenfolge:

   $$(\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ \mathit{count}\ c_2$$

   Hinweis: Sparen Sie sich Schreibarbeit, wo Sie können:

   - Nehmen Sie an, dass *isZero* und *sub* so funktionieren, wie man es erwarten würde, ohne die Definitionen auszufalten.

- Wenn Sie einen bekannten Subterm nochmal reduzieren müssen, können Sie gleich das Endresultat einsetzen.

Beispiellösung:

$$
\begin{aligned}
& (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ \mathit{count}\ c_2 \\
\Rightarrow\ & \mathit{count}\ \mathit{count}\ c_2 = (\lambda \mathtt{y}.\lambda \mathtt{n}.\ \mathit{isZero}\ \mathtt{n}\ c_1\ ((\mathtt{y}\ \mathtt{y})\ (\mathit{sub}\ \mathtt{n}\ c_1)))\ \mathit{count}\ c_2 \\
\Rightarrow\ & (\lambda \mathtt{n}.\ \mathit{isZero}\ \mathtt{n}\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ \mathtt{n}\ c_1)))\ c_2 \\
\Rightarrow\ & \mathit{isZero}\ c_2\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_2\ c_1)) \\
\Rightarrow\ & c_{\mathit{false}}\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_2\ c_1)) = (\lambda \mathtt{t}.\lambda \mathtt{f}.\ \mathtt{f})\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_2\ c_1)) \\
\Rightarrow^2\ & (\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_2\ c_1) \\
\Rightarrow\ & (\lambda \mathtt{n}.\ \mathit{isZero}\ \mathtt{n}\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ \mathtt{n}\ c_1)))\ (\mathit{sub}\ c_2\ c_1) \\
\Rightarrow^*\ & (\lambda \mathtt{n}.\ \mathit{isZero}\ \mathtt{n}\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ \mathtt{n}\ c_1)))\ c_1 \\
\Rightarrow\ & \mathit{isZero}\ c_1\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_1\ c_1)) \\
\Rightarrow^*\ & (\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_1\ c_1) \\
\Rightarrow^*\ & (\mathit{count}\ \mathit{count})\ c_0 \\
\Rightarrow^*\ & \mathit{isZero}\ c_0\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_0\ c_1)) \\
\Rightarrow^*\ & c_{\mathit{true}}\ c_1\ ((\mathit{count}\ \mathit{count})\ (\mathit{sub}\ c_0\ c_1)) \\
\Rightarrow^*\ & c_1
\end{aligned}
$$

(c) Wie viele Reduktionsschritte kann man auf ω' anwenden? Machen Sie sich die Rolle des Subterms `y y` in diesen Reduktionen klar.

Beispiellösung: ω' divergiert, und nimmt dabei die Form `m (m (...) n) n` an. Man kann also beliebig viele β-Reduktionen anwenden. Die Selbstapplikation `y y` *kopiert* dabei einen Term und *substituiert* ihn in sich selbst hinein. Dieser Schritt entspricht dem "Auffalten" einer rekursiven Definition um einen Schritt. Da der kopierte Term wieder eine Selbstapplikation enthält, wird dieser Trick beliebig weiter perpetuiert.

3. (a) Geben Sie einen Term W an, welcher unter β-Rekursion nach und nach die Form `f (f (f (f (...))))` annimmt.

Beispiellösung: Man muss ω' nur leicht anpassen: $W = (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{y}.\ \mathtt{f}\ (\mathtt{y}\ \mathtt{y}))$

(b) Beschreiben Sie, welche Form der Term $X = \lambda \mathtt{f}.\ (\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$ unter β-Reduktion annimmt.

Beispiellösung: Übersehen Sie nicht das $\lambda \mathtt{f}.$:

$$(\lambda \mathtt{f}.\ \mathtt{f}\ (\mathtt{f}\ (\mathtt{f}\ (\mathtt{f}\ (...)))))$$

(c) Führen Sie einen β-Reduktionsschritt von X aus. Erkennen Sie den resultierenden Term wieder?

Beispiellösung:

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

Es handelt sich um den aus der Vorlesung bekannten Y-Kombinator. Wenn Sie sich nicht merken können, wie in Y die Klammern stehen müssen, denken Sie an X!

4. Geben Sie unter Benutzung von X einen Term an, welcher zu ω reduziert

Beispiellösung: $X\ (\lambda \mathtt{x}.\ \mathtt{x}) \Rightarrow \omega$

ω entspricht also dem trivialen rekursiven "Programm", das aus *nichts weiterem* als dem rekursiven Aufruf besteht.