Tutorium 05: λ-Kalkül

Paul Brinkmeier

22. November 2021

Tutorium Programmierparadigmen am KIT

Übungsblatt 3

2 — Collatz-Vermutung

```
module Collatz where


collatz = iterate next
  where next aN | even aN   = aN `div` 2
                | otherwise = 3 * aN + 1

num = length . takeWhile (/= 1) . collatz

maxNum a b = bestNum [(m, num m) | m <- [a..b]]
  where bestNum = foldl maxSecond (0, 0)
        maxSecond a b
          | snd a >= snd b = a
          | otherwise      = b
```

2 — Collatz-Vermutung

```haskell
module CollatzAlt where

import Collatz (num)
import Data.Function (on)
import Data.List (maximumBy)

maxNum a b =
  maximumBy
    (compare `on` snd)
    [(m, num m) | m <- [a..b]]
```

- „eleganter“
- In der Klausur aber eher nur Funktionen aus der Prelude verwenden

3 — Stream-Kombinatoren

```
module Merge where

import Primes (primes)

merge (x:xs) (y:ys)
  | x <= y    = x : merge xs (y:ys)
  | otherwise = y : merge (x:xs) ys
merge xs ys = xs ++ ys

primepowers n = mergeAll $ map primesExp [1..n]
  where mergeAll = foldl merge []
        primesExp i = map (^i) primes
```

- Für `i` in `1..n` unendliche Liste der Primzahlen hoch `i` erstellen
- Dann: Alle miteinander vereinigen

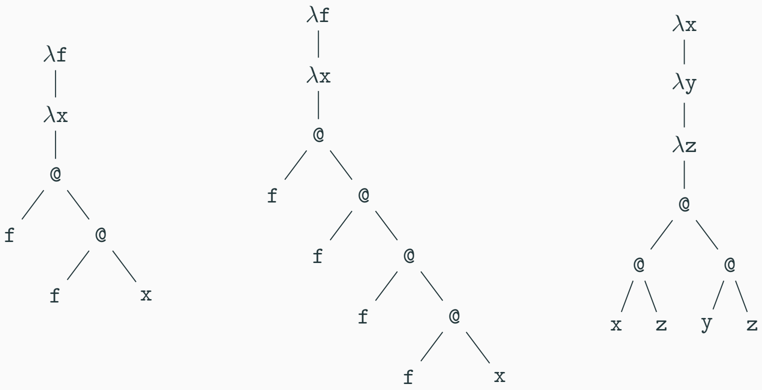
`primepowers`-Visualisierung

Wiederholung

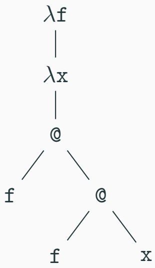
Cheatsheet: Lambda-Kalkül/Basics

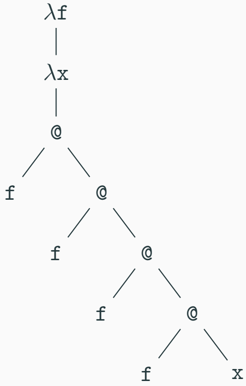
- Terme t: Variable (x), Funktion ($\lambda x.t$), Anwendung ($t\ t$)
- α-Äquivalenz: Gleiche Struktur
- η-Äquivalenz: Unterversorgung
- Freie Variablen, Substitution, RedEx
- β-Reduktion:
  $(\lambda p.b)\ t \Rightarrow b\,[p \to t]$

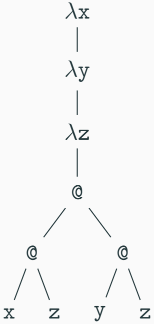
λ-Terme sind Bäume



λ-Terme sind Bäume

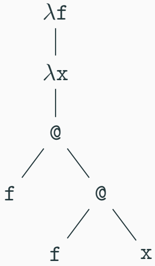




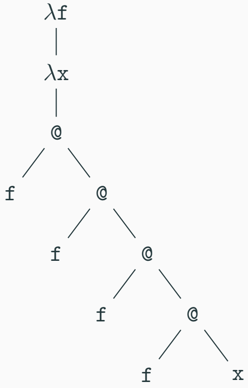


$\lambda f.\, \lambda x.\, f\ (f\ x)$

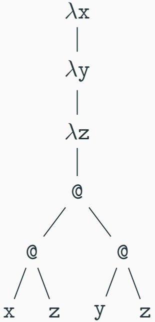
λ-Terme sind Bäume



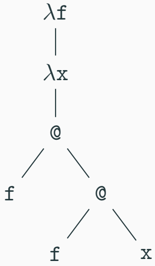
$\lambda f.\, \lambda x.\, f\ (f\ x)$



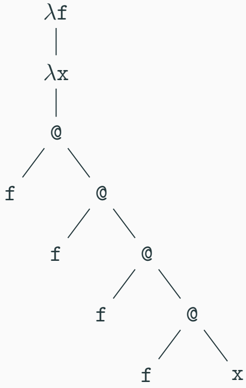
$\lambda f.\, \lambda x.\, f\ (f\ (f\ (f\ x)))$



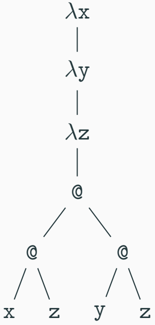
λ-Terme sind Bäume



$\lambda f.\, \lambda x.\, f\ (f\ x)$



$\lambda f.\, \lambda x.\, f\ (f\ (f\ (f\ x)))$



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

Church-Zahlen

Peano-Axiome

$$
\begin{aligned}
c_0 &= ? \\
c_1 &= s(c_0) \\
c_2 &= s(s(c_0)) \\
c_3 &= s(s(s(c_0))) \\
c_8 &= s(s(s(s(s(s(s(s(c_0))))))))
\end{aligned}
$$

1. Die 0 ist Teil der natürlichen Zahlen
2. Wenn n Teil der natürlichen Zahlen ist,
   ist auch $s(n) = n + 1$ Teil der natürlichen Zahlen

Church-Zahlen

- „Zahlen“ im λ-Kalkül werden durch Funktionen in Normalform dargestellt
- $c_n\ f\ x = f$ n-mal angewendet auf x
- Bspw. $(c_3\ g\ y) = g\ (g\ (g\ y)) = g^3\ y$
  Mit $c_3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
- Schreibt eine λ-Funktion *succ*, die eine Church-Zahl nimmt und zu deren Nachfolger auswertet

Church-Zahlen

- „Zahlen“ im λ-Kalkül werden durch Funktionen in Normalform dargestellt
- $c_n\ f\ x = f\ n$-mal angewendet auf x
- Bspw. $(c_3\ g\ y) = g\ (g\ (g\ y)) = g^3\ y$
  Mit $c_3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
- Schreibt eine λ-Funktion *succ*, die eine Church-Zahl nimmt und zu deren Nachfolger auswertet

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

Auswertungsstrategien

Auswertungsstrategien

$$\overbrace{\underline{\text{succ}}\ (\overbrace{\underline{\text{succ}}\ c_0}^{\text{Redex 2}})}^{\text{Redex 1}}$$

mit

$$\begin{aligned} c_0 &= \qquad \lambda s.\, \lambda z.\, z \\ \text{succ} &= \lambda n.\, \lambda s.\, \lambda z.\, s\ (n\ s\ z) \end{aligned}$$

- Welcher Redex soll zuerst ausgewertet werden?
- $\rightsquigarrow$ verschiedene Auswertungsstrategien

Normalreihenfolge

$$
\begin{array}{rc}
& \underline{\mathsf{succ}}\ (\underline{\mathsf{succ}}\ c_0) \\
\Rightarrow_\beta & \lambda s.\,\lambda z.\,s\ ((\underline{\mathsf{succ}}\ c_0)\ s\ z) \\
\Rightarrow_\beta & \lambda s.\,\lambda z.\,s\ (\underline{(\lambda s.\,\lambda z.\,s\ (\underline{c_0}\ s\ z))}\ s\ z) \\
\Rightarrow_\beta^2 & \lambda s.\,\lambda z.\,s\ (s\ (\underline{c_0}\ s\ z)) \\
\Rightarrow_\beta^2 & \lambda s.\,\lambda z.\,s\ (s\ z)\ \not\Rightarrow
\end{array}
$$

Normalreihenfolge: Linkester Redex zuerst.

Call-by-Name

$$\begin{aligned} & \underline{\mathsf{succ}}\ (\underline{\mathsf{succ}}\ c_0) \\ \Rightarrow_\beta \quad & \lambda s.\,\lambda z.\, s\ ((\underline{\mathsf{succ}}\ c_0)\ s\ z) \not\Rightarrow_{\mathsf{CbN}} \end{aligned}$$

Call-by-Name: Linkester Redex zuerst, aber:

- Funktionsinhalte werden nicht weiter reduziert
- $\leadsto$ Betrachte nur Redexe, die nicht von einem λ umgeben sind
- So funktioniert auch Laziness in Haskell (grob)

Call-by-Value

$$
\begin{aligned}
& \underline{\mathsf{succ}}\ (\underline{\mathsf{succ}}\ c_0) \\
\Rightarrow_\beta \quad & \underline{\mathsf{succ}}\ (\lambda s.\, \lambda z.\, s\ (\underline{c_0}\ s\ z)) \\
\Rightarrow_\beta \quad & \lambda s.\, \lambda z.\, \underline{(\lambda s.\, \lambda z.\, s\ (\underline{c_0}\ s\ z))}\ s\ z \ \not\Rightarrow_{\text{CbV}}
\end{aligned}
$$

Call-by-Value: Linkester Redex zuerst, aber:

- Funktionsinhalte werden nicht weiter reduziert
- $\leadsto$ Betrachte nur Redexe, die nicht von einem λ umgeben sind
- Berechne Argumente vor dem Einsetzen
- $\leadsto$ Betrachte nur Redexe, deren Argument unter CbV nicht weiter reduziert werden muss

Cheatsheet: Lambda-Kalkül/Fortgeschritten

- Auswertungsstrategien (von lässig nach streng):
  - Volle β-Reduktion
  - Normalreihenfolge
  - Call-by-Name
  - Call-by-Value
- Datenstrukturen:
  - Church-Booleans
  - Church-Zahlen
  - Church-Listen
- Rekursion durch Y-Kombinator

Klausuraufgaben zum λ-Kalkül

19SS A4 — SKI-Kalkül (13P.)

$$\begin{aligned} S &= \lambda x.\, \lambda y.\, \lambda z.\, x\; z\; (y\; z) \\ K &= \lambda x.\, \lambda y.\, x \\ I &= \lambda x.\, x \end{aligned}$$

- SKI-Kalkül kann alles, was λ-Kalkül auch kann, allein mit den Kombinatoren S, K und I
- Definiere $U = \lambda x.\, x\; S\; K$
- Aufgabe: Beweise, dass man S, K und I durch U darstellen kann:

19SS A4 — SKI-Kalkül (13P.)

$$\begin{aligned} S = &\quad \lambda x.\, \lambda y.\, \lambda z.\, x\ z\ (y\ z) \\ K = &\quad \lambda x.\, \lambda y.\, x \\ I = &\quad \lambda x.\, x \end{aligned}$$

- SKI-Kalkül kann alles, was λ-Kalkül auch kann, allein mit den Kombinatoren S, K und I
- Definiere $U = \lambda x.\, x\ S\ K$
- Aufgabe: Beweise, dass man S, K und I durch U darstellen kann:
  - $U\ U\ x \overset{?}{\Longrightarrow} x$
  - $U\ (U\ (U\ U)) = U\ (U\ I) \overset{?}{\Longrightarrow} K$
  - $U\ (U\ (U\ (U\ U))) = U\ K \overset{?}{\Longrightarrow} S$

18SS A4 — Currying im λ-Kalkül (13P.)

$$\begin{aligned} \text{pair} &= \lambda a.\,\lambda b.\,\lambda f.\,f\ a\ b \\ \text{fst} &= \lambda p.\,p\,(\lambda x.\,\lambda y.\,x) \\ \text{snd} &= \lambda p.\,p\,(\lambda x.\,\lambda y.\,y) \\ \text{fst}\ (\text{pair}\ a\ b) &= a \\ \text{snd}\ (\text{pair}\ a\ b) &= b \end{aligned}$$

- Schreibe curry und uncurry, sodass:
  - $(\text{curry}\ f)\ a\ b = f\ (\text{pair}\ a\ b)$
  - $(\text{uncurry}\ g)\ (\text{pair}\ a\ b) = g\ a\ b$

18WS A5 — Listen im λ-Kalkül (10P.)

$$\begin{aligned} \text{nil} &= \quad \lambda n.\, \lambda c.\, n \\ \text{cons} &= \quad \lambda x.\, \lambda xs.\, \lambda n.\, \lambda c.\, (c\ x\ xs) \end{aligned}$$

- Schreibe *head* und *tail*, sodass:
    - head (cons A B) $\stackrel{*}{\Longrightarrow}$ A
    - tail (cons A B) $\stackrel{*}{\Longrightarrow}$ B

18WS A5 — Listen im λ-Kalkül (10P.)

$$\begin{aligned} \text{nil} &= \qquad \lambda n.\, \lambda c.\, n \\ \text{cons} &= \lambda x.\, \lambda xs.\, \lambda n.\, \lambda c.\, (c\ x\ xs) \end{aligned}$$

- Schreibe *head* und *tail*, sodass:
  - head (cons A B) $\overset{*}{\Longrightarrow}$ A
  - tail (cons A B) $\overset{*}{\Longrightarrow}$ B
- Schreibe replicate, sodass:
  - replicate c_n $A = \underbrace{\text{cons } A \text{ (cons } A \text{ ...(cons } A \text{ nil))}}_{n \text{ mal}}$
- Erinnerung: $c_n\ f\ x = \underbrace{f\ (f\ ...(f\ x))}_{n \text{ mal}}$

18WS A5 — Listen im λ-Kalkül (10P.)

$$\begin{aligned} \text{nil} &= \qquad\quad \lambda n.\, \lambda c.\, n \\ \text{cons} &= \lambda x.\, \lambda xs.\, \lambda n.\, \lambda c.\, (c\ x\ xs) \end{aligned}$$

- Schreibe *head* und *tail*, sodass:
  - $\text{head}\ (\text{cons}\ A\ B) \stackrel{*}{\Longrightarrow} A$
  - $\text{tail}\ (\text{cons}\ A\ B) \stackrel{*}{\Longrightarrow} B$
- Schreibe replicate, sodass:
  - $\text{replicate}\ c_n\ A = \underbrace{\text{cons}\ A\ (\text{cons}\ A\ ...(\text{cons}\ A\ \text{nil}))}_{n\ \text{mal}}$
- Erinnerung: $c_n\ f\ x = \underbrace{f\ (f\ ...(f\ x))}_{n\ \text{mal}}$
- Werte aus: $\text{replicate}\ c_3\ A \stackrel{*}{\Longrightarrow}\ ?$