Funktionale Programmierung
Sommersemester 2021

Prof. Dr. Jakob Rehof
TU Dortmund
LS XIV Software Engineering

Diese Vorlesung:

- DoberkatFP: Folien 57 – 98
  - Endlich! Wir lernen, mit Listen zu programmieren!
- Lambda-Kalkül:
  - Reduktionstheorie (call-by-name, lazy evaluation)
  - Kodierungen im reinen Lambda-Kalkül live demo
    - Live demo
    - Siehe dazu: https://en.wikipedia.org/wiki/Lambda_calculus#Encoding_datatypes

λ - Kalkül: Reduktionstheorie

Syntax (λ-Terme):

$$M ::= x \mid (M\ M) \mid (\lambda x.M)$$

Semantik (Reduktion):

$$((\lambda x.M)\ N) \longrightarrow_\beta M[x := N] \quad (Redex)$$

Sei $\longrightarrow_\beta^*$ die reflexiv-transitive Hülle von $\longrightarrow_\beta$.
Also, $P \longrightarrow_\beta^* Q$ genau dann, wenn

$$\exists n \geq 0.\ P = P_0 \longrightarrow_\beta P_1 \longrightarrow_\beta \ldots \longrightarrow_\beta P_n = Q$$

λ - Kalkül: Reduktionstheorie

Ein term M ist eine *Normalform*, genau dann wenn M kein Redex beinhält (also, es gibt keinen Term Q mit $M \longrightarrow_\beta Q$).

- $(\lambda x.x)$ ist eine Normalform.
- $(y(\lambda x.x))$ ist eine Normalform.
- $((\lambda x.x)y)$ ist keine Normalform.

Ein Term M ist *(schwach) normalisierend*, wenn M eine Normalform hat, das heisst es existiert eine Normalform N mit

$$M \longrightarrow_\beta^* N$$

Nicht alle Terme sind normalisierend. Sei $\Omega = ((\lambda x.(xx))(\lambda x.(xx)))$. Dann gilt offensichtlich

$$\Omega \longrightarrow_\beta \Omega$$

als einzig mögliche Reduktion.

λ - Kalkül: Reduktionstheorie

Terme können beliebig viele Redices beinhalten. Sei

- $\mathbf{I} = (\lambda x.x)$
- $\mathbf{I}^* = ((\mathbf{II})(\mathbf{II}))$
- $\mathbf{K} = (\lambda x.(\lambda y.x))$
- $\mathbf{R} = ((\mathbf{KI})\Omega)$

Dann haben wir

- $\mathbf{I}^* \longrightarrow_\beta (\mathbf{I}(\mathbf{II}))$ und $\mathbf{I}^* \longrightarrow_\beta ((\mathbf{II})\mathbf{I})$
- $\mathbf{R} \longrightarrow_\beta \mathbf{R}$
- $\mathbf{R} \longrightarrow_\beta ((\lambda y.\mathbf{I})\Omega) \longrightarrow_\beta \mathbf{I}$

λ - Kalkül: Reduktionstheorie

Es stellt sich sofort die Frage:

- Sind alle Reduktionspfade gleich gut?

Einige sind *effizienter* als andere, auch mit demselben Ergebnis:

$$\mathbf{R} \longrightarrow_\beta \mathbf{R} \longrightarrow_\beta \mathbf{R} \longrightarrow_\beta^* \mathbf{I}$$

ist nicht so effizient wie die direkte Reduktion zur Normalform, die wir vorher sahen:

$$\mathbf{R} \longrightarrow_\beta^* \mathbf{I}$$

Wir könnten aber auch Fragen:

- Hat jeder normalisierende Term eine *eindeutige Normalform*?

λ - Kalkül: Reduktionstheorie

Die Antwort auf diese letztere Frage ist:

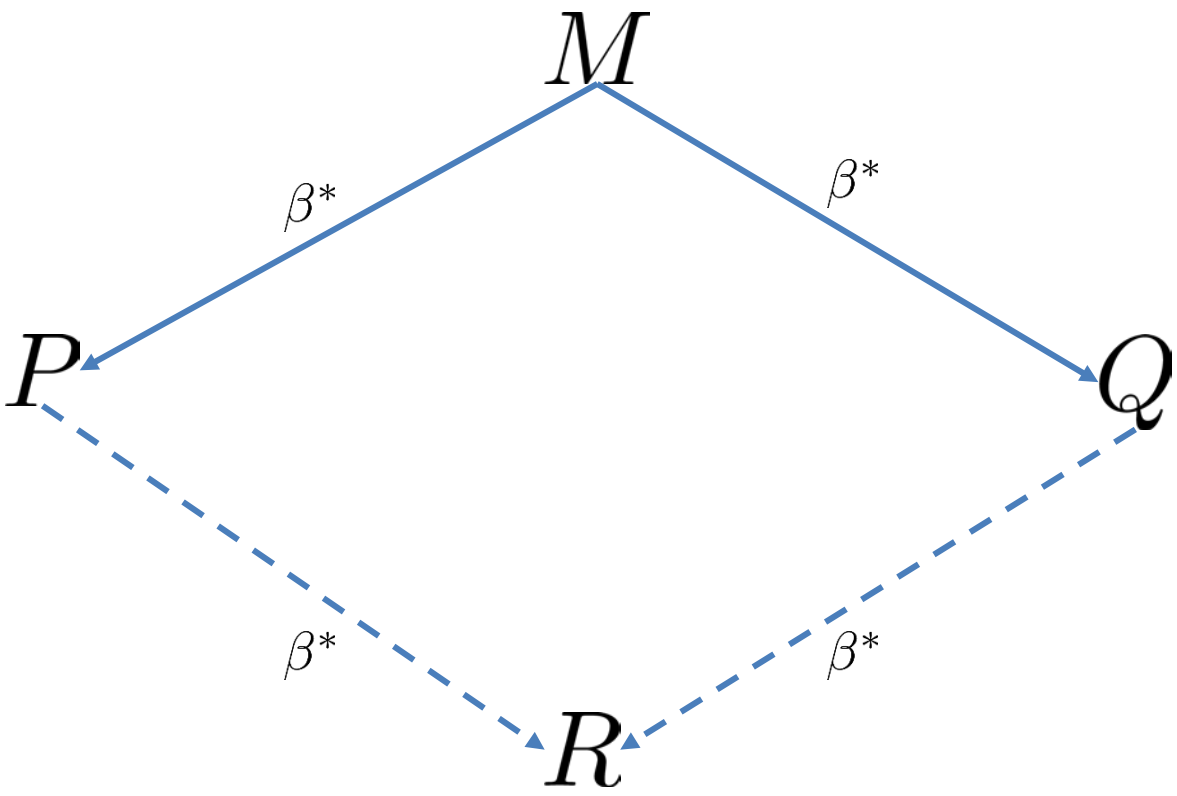
- JA!

Church-Rosser (Konfluenz) Satz:

- *Für alle* M, P, Q *mit* $M \longrightarrow_\beta^* P$ *und* $M \longrightarrow_\beta^* Q$ *existiert* R *mit* $P \longrightarrow_\beta^* R$ *und* $Q \longrightarrow_\beta^* R$

Daraus folgt die Eindeutigkeit von Normalformen unmittelbar.
Somit ist der Kalkül deterministisch bezüglich der Ergebnisse der Berechnung, obwohl die Reduktionssemantik von $\longrightarrow_\beta^*$ nicht deterministisch ist.

λ - Kalkül: Reduktionstheorie



"Church-Rosser-Diamant"

λ - Kalkül: Reduktionstheorie

Wir können nun fragen:

- Gibt es eine deterministische *Reduktionsstrategie*, die bei normalisierenden Termen immer zur Normalform führt?

Die Antwort auf diese Frage ist:

- JA!

Standardisierungssatz:

- *Leftmost-outermost (LO) Reduktion führt immer zur Normalform, wenn sie existsiert.*

λ - Kalkül: Reduktionstheorie

Vollständige Definition der β-Reduktion ($\longrightarrow_\beta$):

- $((\lambda x.P)Q) \longrightarrow_\beta P[x := Q]$
- $P \longrightarrow_\beta P' \Rightarrow (PQ) \longrightarrow_\beta (P'Q)$
- $Q \longrightarrow Q' \Rightarrow (PQ) \longrightarrow_\beta (PQ')$
- $P \longrightarrow_\beta P' \Rightarrow (\lambda x.P) \longrightarrow_\beta (\lambda x.P')$

λ - Kalkül: Reduktionstheorie

Call-by-name Reduktion $(\longrightarrow_{\beta N})$:

- $((\lambda x.P)Q) \longrightarrow_{\beta N} P[x := Q]$
- $P \longrightarrow_{\beta N} P' \Rightarrow (PQ) \longrightarrow_{\beta N} (P'Q)$

Die Relation $\longrightarrow_{\beta N}$ ist die LO-Strategie.
Sie wird auch *lazy evaluation* genannt.
Haskell ist grundsätzlich *lazy*.

λ - Kalkül: Reduktionstheorie

Call-by-value Reduktion ($\longrightarrow_{\beta V}$):

- $((\lambda x.P)V) \longrightarrow_{\beta V} P[x := V]$
- $P \longrightarrow_{\beta N} P' \Rightarrow (PQ) \longrightarrow_{\beta V} (P'Q)$
- $Q \longrightarrow_{\beta V} Q' \Rightarrow (VQ) \longrightarrow_{\beta V} (VQ')$

wobei V (*values*, Werte) durch

- $V ::= x \mid (\lambda x.P)$

definiert sind.
Die Reduktion ist außerdem left-to-right orientiert (dritte Regel).