Lambdakalkül

Die kleinste universelle Programmiersprache der Welt

Christian Höner zu Siederdissen
christian.hoener.zu.siederdissen@uni-jena.de

Theoretische Bioinformatik, Bioinformatik Uni Jena

Januar 2024

Berechnungsmodelle

Turing Maschine

Partiell-rekursive Funktionen

Lambdakalkül

Definition des λ-Kalküls

Ein Ausdruck ist rekursiv definiert:

$$
\begin{aligned}
\text{Expression} &:= \text{Name} \mid \text{Function} \mid \text{Application} \mid \text{BuiltIn} \\
\text{Function} &:= \lambda\text{Name} \,.\, \text{Expression} \\
\text{Application} &:= \text{Expression Expression}
\end{aligned}
$$

- wenn nötig können Klammern gesetzt werden, wobei $E \equiv (E)$
- die anderen Schlüsselwörter sind λ (Lambda) und . (Dot)
- *abcde* wird als $(((ab)c)d)e$ gelesen
- Zahlen, sowie elementare Funktionen $+, -, \times, \ldots$ sind vorgegeben, auch *true, false*
- den vorigen Punkt werden wir später betrachten

Function application

- Vorweg: Funktionen werden (immer) prefix geschrieben: $1 + 2$ wird zu $+\,1\,2$
- Die Funktion $f(x) = x + 1$ wird zu $\lambda x.(+\,1\,x)$
- Anwedungsbeispiel: $(\lambda x.(+\,1\,x))\,2 = (+\,1\,2) = 3$

Programmausdrücke

Gegeben $(+\,1\,2)$:

1. Exp
2. Exp Exp
3. (BuiltIn: $+$) Exp
4. $+$ Exp Exp
5. $+$ (BuiltIn: 1) Exp
6. $+$ 1 (BuiltIn: 2)
7. $+$ 1 2
8. $(+$ 1 2$)$

Funktionsanwendung

Gegeben sei die (Identitätsfunktion) mit Anwendung: $(\lambda x.x)y$

- Ersetze jedes x durch y
- Gebe den entstehenden Ausdruck zurück (wie bei "BuiltIn"s): y

Das werden wir jetzt formalisieren

Gebundene vs freie Variablen

Gegeben sei $(\lambda x. + \, x \, y)1$

gebundene Variablen x ist gebunden und zwar an den Wert 1

freie Variablen y ist nicht gebunden, sondern frei

Innerhalb eines Ausdrucks (Expression) ist eine Variable entweder gebunden oder frei

Delta (δ)-Regeln

- δ-Regeln evaluieren eingebaute Funktionen (BuiltIn)
- Gegeben $(+\,1\,2)$ folgt die Evaluierung:
- $(+\,1\,2) \xrightarrow{\delta} 3$

β-Reduktion

Formal schreibt man $(\lambda x.E)z \xrightarrow{\beta} E\left[\frac{z}{x}\right]$

- Im Ausdruck $(\lambda x.E)$ wird im Ausdruck E ersetzt
- Und zwar alle *freien* Variablen x durch z
- β-Reduktion gibt dann eine Kopie des (entsprechend ausgewerteten) Ausdrucks zurück

Beispiele:

- $(\lambda x.(+\,x\,1))\,2 \xrightarrow{\beta} (+\,2\,1) \xrightarrow{\delta} 3$

β-Reduktion

Formal schreibt man $(\lambda x.E)z \xrightarrow{\beta} E\left[\frac{z}{x}\right]$

- Im Ausdruck $(\lambda x.E)$ wird im Ausdruck E ersetzt
- Und zwar alle *freien* Variablen x durch z
- β-Reduktion gibt dann eine Kopie des (entsprechend ausgewerteten) Ausdrucks zurück

Beispiele:

- $(\lambda x.(+\,x\,1))\,2 \xrightarrow{\beta} (+\,2\,1) \xrightarrow{\delta} 3$
- $(\lambda x.(+\,x\,x))\,1 \xrightarrow{\beta} (+\,1\,1) \xrightarrow{\delta} 2$

β-Reduktion

Formal schreibt man $(\lambda x.E)z \xrightarrow{\beta} E\left[\frac{z}{x}\right]$

- Im Ausdruck $(\lambda x.E)$ wird im Ausdruck E ersetzt
- Und zwar alle *freien* Variablen x durch z
- β-Reduktion gibt dann eine Kopie des (entsprechend ausgewerteten) Ausdrucks zurück

Beispiele:

- $(\lambda x.(+\,x\,1))\,2 \xrightarrow{\beta} (+\,2\,1) \xrightarrow{\delta} 3$
- $(\lambda x.(+\,x\,x))\,1 \xrightarrow{\beta} (+\,1\,1) \xrightarrow{\delta} 2$
- $(\lambda x.(\lambda y.(+\,x\,y)))\,1\,2 \xrightarrow{\beta} (\lambda y.(+\,1\,y))\,2 \xrightarrow{\beta} (+\,1\,2) \xrightarrow{\delta} 3$

β-Reduktion

Formal schreibt man $(\lambda x.E)z \xrightarrow{\beta} E\left[\frac{z}{x}\right]$

- Im Ausdruck $(\lambda x.E)$ wird im Ausdruck E ersetzt
- Und zwar alle *freien* Variablen x durch z
- β-Reduktion gibt dann eine Kopie des (entsprechend ausgewerteten) Ausdrucks zurück

Beispiele:

- $(\lambda x.(+\,x\,1))\,2 \xrightarrow{\beta} (+\,2\,1) \xrightarrow{\delta} 3$
- $(\lambda x.(+\,x\,x))\,1 \xrightarrow{\beta} (+\,1\,1) \xrightarrow{\delta} 2$
- $(\lambda x.(\lambda y.(+\,x\,y)))\,1\,2 \xrightarrow{\beta} (\lambda y.(+\,1\,y))\,2 \xrightarrow{\beta} (+\,1\,2) \xrightarrow{\delta} 3$
- $(\lambda f.f\,1)\,(\lambda x.(+\,x\,2)) \xrightarrow{\beta} (\lambda x.(+\,x\,2))1 \xrightarrow{\beta} (+\,1\,2) \xrightarrow{\delta} 3$

β-Reduktion 2

- $(\lambda x.(\lambda x. + (+\, x\, 1))\, x\, 2)\, 3$
- $\xrightarrow{\beta} (\lambda x. + (+\, x\, 1))\, 3\, 2$
- $\xrightarrow{\beta} + (+\, 3\, 1)\, 2$
- $\xrightarrow{\delta} +\, 4\, 2$
- $\xrightarrow{\delta} 6$

Alpha(α)-Konvertierung

Was ist das Problem? Betrachte folgende β-Reduktion:

- $(\lambda f.\lambda x.f(f\,x))x$
- $\xrightarrow{\beta} \lambda x.x(x\,x)$

Durch die gleichen Namen ist jetzt eine falsche Funktion entstanden! Wir müssen den Parameter x umbenennen:

$$\lambda x.f(f\,x) \xrightarrow{\alpha} \lambda y.f(f\,y)$$

formal:

$$(\lambda x.E) \xrightarrow{\alpha} \lambda y.E\left[\frac{y}{x}\right]$$

- $(\lambda f.\lambda x.f(f\,x))x$
- $\xrightarrow{\alpha} (\lambda f.\lambda y.f(f\,y))x$
- $\xrightarrow{\beta} \lambda y.x(x\,y)$

Eta(η)-Reduktion

Zwei Ausdrücke die sich gleich verhalten (wenn sie auf ein Argument angewandt werden) können mittels η-Reduktion ineinander umgewandelt werden.

Beispiel:

$$(\lambda x. + 1\, x) = (+\, 1)$$

Formal:

$$(\lambda x. E\, x) \xrightarrow{\eta} E$$

Normal-Formen, Reduktion

Evaluation eines Programms ist die Anwendung von Regeln bis Normalform erreicht wird.
Wie wollen wir "reduzieren"?

1. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,3\,(+\,3\,4)$

Normal-Formen, Reduktion

Evaluation eines Programms ist die Anwendung von Regeln bis Normalform erreicht wird.
Wie wollen wir "reduzieren"?

1. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,3\,(+\,3\,4)$
2. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,(+\,1\,2)\,7$

Normal-Formen, Reduktion

Evaluation eines Programms ist die Anwendung von Regeln bis Normalform erreicht wird.
Wie wollen wir "reduzieren"?

1. $+ (+\,1\,2)\,(+\,3\,4) \rightarrow +\,3\,(+\,3\,4)$
2. $+ (+\,1\,2)\,(+\,3\,4) \rightarrow +\,(+\,1\,2)\,7$

Das kann wichtig sein, betrachte:

$$(\lambda x.x\,x)\,(\lambda x.x\,x) \xrightarrow{\beta} (\lambda x.x\,x)\,(\lambda x.x\,x)$$

Normal-Formen, Reduktion

Evaluation eines Programms ist die Anwendung von Regeln bis Normalform erreicht wird.
Wie wollen wir "reduzieren"?

1. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,3\,(+\,3\,4)$
2. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,(+\,1\,2)\,7$

Das kann wichtig sein, betrachte:

$$(\lambda x.x\,x)\,(\lambda x.x\,x) \xrightarrow{\beta} (\lambda x.x\,x)\,(\lambda x.x\,x)$$

als Teil von:

1. $(\lambda x.3)((\lambda x.x\,x)\,(\lambda x.x\,x)) \xrightarrow{\beta} 3$

Normal-Formen, Reduktion

Evaluation eines Programms ist die Anwendung von Regeln bis Normalform erreicht wird.
Wie wollen wir “reduzieren”?

1. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,3\,(+\,3\,4)$
2. $+\,(+\,1\,2)\,(+\,3\,4) \rightarrow +\,(+\,1\,2)\,7$

Das kann wichtig sein, betrachte:

$$(\lambda x.x\,x)\,(\lambda x.x\,x) \xrightarrow{\beta} (\lambda x.x\,x)\,(\lambda x.x\,x)$$

als Teil von:

1. $(\lambda x.3)((\lambda x.x\,x)\,(\lambda x.x\,x)) \xrightarrow{\beta} 3$
2. $(\lambda x.3)((\lambda x.x\,x)\,(\lambda x.x\,x)) \xrightarrow{\beta} \ldots \xrightarrow{\beta} (\lambda x.3)((\lambda x.x\,x)\,(\lambda x.x\,x))$

Zusammenfassung

- β-Reduktion reduziert Ausdrücke
- α-Konvertierung dient der (Semantik-erhaltenden) Umbenennung
- η-Reduktion entfernt Lambda-Abstraktionen
- δ-Regeln erlauben es "eingebaute" Funktionen zu nutzen
- Die Wahl der Reduktion ist wichtig (Terminierung, Laziness)

Natürliche Zahlen

Via "Church-Encoding":

$$
\begin{aligned}
0 &\equiv \lambda s.\lambda z.z \\
1 &\equiv \lambda s.\lambda z.s(z) \\
2 &\equiv \lambda s.\lambda z.s(s(z)) \\
3 &\equiv \lambda s.\lambda z.s(s(s(z)))
\end{aligned}
$$

Zahlen sind Funktionen! Und Schleifen!

$$2\, f\, a \rightarrow \lambda s.\lambda z.s(s(z)) f\, a \rightarrow f(f(a))$$

"S"uccessor Funktion:

$$S \equiv \lambda n.\lambda a.\lambda b.a(n\, a\, b)$$

Addition und Multiplikation

Addition in $a + b$ ist die a-fache Anwendung der “S” Funktion auf b: $a\,S\,b$

Multiplikation in $a * b$ ist die Funktion b a-mal anwenden: $a\,b$

Booleans

True und False finden typischerweise Anwendung in Entscheidungen. Statt die Datentypen zu kodieren, werden die *Entscheidungen* kodiert:

True	$T \equiv \lambda x.\lambda y.x$
False	$F \equiv \lambda x.\lambda y.y$
"and"	$\wedge \equiv \lambda x.\lambda y.x\,y\,F$
"or"	$\vee \equiv \lambda x.\lambda y.x\;T\;y$
"not"	$\neg \equiv \lambda x.x\,F\;T$
"isZero"	$== 0 \equiv \lambda x.x\,F\,\neg\,F$

Rekursion mittels Y-Kombinator

Der Y-Kombinator ist definiert als:

$$Y \equiv (\lambda y.(\lambda x.y(x\,x))\,(\lambda x.y(x\,x)))$$

Angewandt auf f haben wir:

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

Eine unendliche Schleife ist nun: $Y(\lambda x.x)$