Funktionale Programmierung
- Grundlagen -

λ

λ-Ausdrücke

1) Variablen: x, y, z, ...V,V_1,...
2) Applikationen: (E_1 E_2)
   Bsp. (sum($\underline{m}$, $\underline{n}$)) bezeichnet $\underline{m+n}$
3) Abstraktionen: (λV.E)
   Bsp. λx.sum(x,$\underline{1}$) Inkrementierfunktion

in BNF:
< λ-Ausdruck> ::= <Variable>
| (< λ-Ausdruck> <(λ-Ausdruck)>
| λ<Variable>.< λ-Ausdruck>)>

Vereinbarungen zur Notation

- Unterstrichene Bezeichnungen stehen für λ-Teilausdrücke
  Bsp. $\underline{1}$ steht für den λ-Ausdruck, der den Wert 1 repräsentiert.
- Funktionen werden linksassoziativ ausgewertet:
  E_1 E_2 ... E_n steht für (...((E_1 E_2) E_3)...E_n)
  λV.E_1... E_n steht für (λV.(E_1 ... E_n))
  λV_1...V_n.E steht für (λV_1.(...(λV_n.(E)...))
  Bsp. λx.y.add y x bezeichnet (λx. λy.((add y)x)))
- E[E'/V] steht für das Ergebnis einer textuellen Ersetzung (Substitution) jeder freien Variablen V in E durch E'.

Frei und gebundene Variablen

gebunden

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

frei

Eine Substitution E[E‘/V] wird genau dann als gültig bezeichnet, wenn keine freie Variable in E‘ zu einer gebundenen Variablen in E[E‘/V] wird.

Konvertierungsregeln

α-Regel
Jede Abstraktion der Form $\lambda V.E$ kann zu $\lambda V\ E[V'/V]$ konvertiert werden, vorausgesetzt, die Substitution von V in E durch V‘ ist gültig.

β-Regel
Jede Applikation der Form $(\lambda V.E_1)E_2$ kann zu $E_1[E_2/V]$ konvertiert werden, vorausgesetzt, die Substitution von V in E_1 durch E_2 ist gültig.

η-Regel
Jede Abstraktion der Form $\lambda V.(E\ V)$, in der V in E nur gebunden auftritt, kann zu E reduziert werden

Gleichheit und Identität

Zwei λ-Ausdrücke nennen wir **gleich**, wenn sie mit einer Sequenz von Anwendungen der Konvertierungsregeln ineinander überführt werden können (vorwärts wie rückwärts).

$E_1 = E_2$

Zwei λ-Ausdrücke nennen wir **identisch**, wenn sie aus genau der gleichen Folge von Zeichen bestehen.

$E_1 \equiv E_2$

Bsp.

$(\lambda x.x)((\lambda y.y)1) = 1$

$(\lambda x.\ \lambda y.\text{add } x\ y)3\ 4 = \text{add } 3\ 4$

Repräsentation von Objekten

LET true = λx. λy.x
LET false = λx. λy.y
LET not = λt.t false true

Exemplarischer Nachweis „üblicher“ Eigenschaften:

not true	= (λt.t false true) true	(* Def. von not *)
	= true false true	(* β-Regel *)
	= (λx. λy.x) false true	(* Def. von true *)
	= (λy.false) true	(* β-Regel *)
	= false	(* β-Regel *)

Bedingte Ausdrücke:
LET $(E \rightarrow E_1|E_2) = (E\ E_1\ E_2)$

es gilt:

$(\text{true} \rightarrow E_1|E_2) = \text{true}\ E_1\ E_2 = (\lambda x\ y.x)\ E_1\ E_2 = E_1$

$(\text{false} \rightarrow E_1|E_2) = \text{false}\ E_1\ E_2 = (\lambda x\ y.y)\ E_1\ E_2 = E_2$

Tupel:
LET fst = λp.p true
LET snd = λp.p false
LET $(E_1,E_2) = \lambda f.f\ E_1\ E_2$

es gilt:
$\text{fst}\ (E_1, E_2) = (\lambda p.p\ \text{true})\ (E_1, E_2)$
$= (E_1, E_2)\ \text{true}$
$= (\lambda f.f\ E_1\ E_2)\ \text{true}$
$= \text{true}\ E_1\ E_2$
$= (\lambda x\ y.x)\ E_1\ E_2$
$= E_1$

LET $(E_1, E_2, \ldots, E_n) = (E_1,(E_2,(\ldots (E_{n-1},E_n)\ldots)))$

Zahlen und arithmetische Operationen

n-te Applikation:
LET $E^0\ E' = E'$
LET $E^n\ E' = \underbrace{E(E(\ldots(E}_{\text{n Es}}\ E')\ldots))$

Zahlen:
LET 0 = λf x.x
LET 1 = λf x.f x
LET 2 = λf x.f(f x)
...
LET n = λf x f^n x

arithmetische Operationen:
LET suc λn f x.n f(f x)
LET add λm n f x.m f (n f x)
LET iszero = λn.n (λx. false) true