ЛЕКЦИЯ 10
λ-Исчисление. Теоретические основы функционального программирования

Лямбда-исчисление

λ-исчисление – это набор формальных систем, основанных на нотации, которую придумал Алонзо Черч (A. Church) в 1930 г.

Исчисление анонимных функций

Лямбда-исчисление

x−y можно рассматривать, как функцию f(x) и функцию g(y)

f(x) = x−y g(y) = x−y

f: x -> x−y g: y -> x−y

Нотация Черча:

f ≡ λx. x−y g ≡ λy. x−y

Настоящая нотация Чёрча

На самом деле Чёрч писал с «крышкой»: ^

ŷ. x−y

Есть версия, что при наборе его статьи в типографии не нашлось нужной литеры, поэтому напечатали так:

Λy. x−y

Дальше при перепечатке заменили большую лямбду на маленькую:

λy. x−y

Аппликация

Аппликация (apply, композиция):

$f \equiv \lambda x.\ x-y$

$f(0) = 0-y$ в нотации Чёрча: $(\lambda x.\ x-y)\ 0 = 0 - y$

$f(1) = 1-y$ в нотации Чёрча: $(\lambda x.\ x-y)\ 1 = 1 - y$

Каждая λ-функция – это функция от одного аргумента!

Аппликация

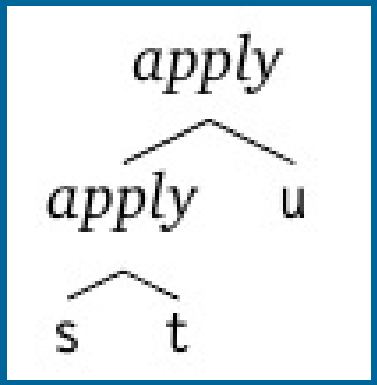
Функции от двух и более аргументов в λ-нотации:

f(x,y) = x−y ==> λx. λy. x−y

Аппликация левоассоциативная!

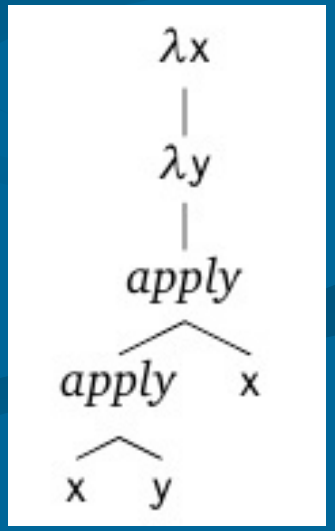
(λx. λy. x−y) 7 2 => (λy. 7−y) 2 ==> 7−2

s t u ≡ ((s t) u)



Аппликация «заберёт всё, до чего дотянется»

λx. λy. x y x ≡ λx. (λy. ((x y) x))



λ-нотация

Свободные переменные и связанные переменные

$$\lambda x.\ x - y$$

x – связанная переменная

y – свободная переменная

Формальное определение

λ-терм (λ-выражение) – это:

переменная (например, x);

константа (например, c1);

комбинация или аппликация s t функции s к аргументу t,
где s и t – λ-термы;

абстракция λx. s λ-терма s по переменной x

БНФ для λ-термов:

<λ-терм> ::= <переменная> | <константа> |
<λ-терм> <λ-терм> | λ <переменная>. <λ-терм>

Свободные и связанные переменные

Вхождение переменной x в λ-терм t является свободным, если оно лежит вне области действия соответствующей абстракции.

Формально FV(t) – множество свободных переменных терма t, BV(t) – связанных:

$FV(x) = \{x\}$	$BV(x) = \varnothing$
$FV(c1) = \varnothing$	$BV(c1) = \varnothing$
$FV(s\ t) = FV(s) \cup FV(t)$	$BV(s\ t) = BV(s) \cup BV(t)$
$FV(\lambda x.\ s) = FV(s) - \{x\}$	$BV(\lambda x.\ s) = BV(s) \cup \{x\}$

Пример: $s = (\lambda x.\ \lambda y.\ x)\ (\lambda x.\ z\ x)$

$FV(s) = \{z\}$	$BV(s) = \{x, y\}$

Подстановка

Подстановка $t[s/x]$ – это терм, получаемый из терма t заменой переменной x на терм s.

Пример: $(\lambda y.\ x+y)[5/x] = \lambda y.\ 5+y$

Ещё пример: $(\lambda y.\ x+y)[y/x] = \lambda y.\ y+y$??? $= \lambda w.\ y+w$!!!

При подстановке следует учитывать свободные/связанные переменные.

$x[t/x] = t$ $\quad y[t/x] = y$, если $x \neq y$ $\quad c[t/x] = c$

$(s\ u)[t/x] = s[t/x]\ u[t/x]$ $\quad (\lambda x.\ s)[t/x] = \lambda x.\ s$

$(\lambda y.\ s)[t/x] = \lambda y.\ (s[t/x])$, если $x \neq y$, либо $x \notin FV(s)$, либо $y \notin FV(t)$

$(\lambda y.\ s)[t/x] = \lambda z.\ (s[z/y][t/x])$, иначе, причём $z \notin FV(s) \cup FV(t)$

Преобразования λ-выражений

- α-редукция «переименование связанной переменной»
  - если v и w – переменные, а t – λ-терм, то

  λv. t -α-> λw. t[v/w], если w $\notin$ FV(t)
- Пример: λx. λy. x−y -α-> λx. λv. x−v

Преобразования λ-выражений

- β-редукция «вычисление функции для заданного аргумента»
  - (λx. s) t -β-> s[t/x]
- Пример:

(λx. (λy. x−y)) 7 2 -β-> (λy. 7−y) 2 -β-> 7−2

- Бывает ещё η-редукция, но мы её не рассматриваем.

Примеры β-редукции

- (λx. (λy. y x) z) v -β-> (λy. y v) z -β-> z v
- (λx. x x) (λx. x x) -β-> (λx. x x) (λx. x x) -β-> (λx. x x) (λx. x x) -β-> … -β-> (λx. x x) (λx. x x)
- (λx. x x y) (λx. x x y) -β-> (λx. x x y) (λx. x x y) y -β-> (λx. x x y) (λx. x x y) y y -β-> …

Эквивалентность

- Термы t и v эквивалентны (записывается t = v), если есть цепочка термов s, r, ..., такая что t ---> s ---> r ---> ... ---> v, где каждая ---> является либо -α->, либо -β->, либо «обратной β-редукцией» <-β-
- Справедливо, что

t = t

если s = t, то t = s

если s = t и t = v, то s = v

если s = t, то s u = t u

если s = t, то u s = u t

если s = t, то λx. s = λx. t

- Эквивалентность – не тождественность:

  λx. x = λy. y но они не тождественны

λ-Редукция

- λ-редукция, это «эквивалентность в одну сторону» (без «обратной β-редукции» <-β-)

t --> t

~~если s --> t, то t --> s~~

если s --> t и t --> v, то s --> v

если s --> t, то s u --> t u

если s --> t, то u s --> u t

если s --> t, то λx. s --> λx. t

- термин λ-редукция соотносится с понятием «вычисление функ. программы»

Нормальная форма

λ-выражение находится в нормальной форме, если ни одна β-редукция не может быть применена.

- Для выражения (λx. (λy. y x) z) v нормальная форма z v
- Для выражения (λx. x x y) (λx. x x y) нормальной формы нет

Нормальная форма

- В каком порядке делать редукции?
- Например, обозначим L= (λx. x x y) (λx. x x y)
- Найдем н. ф. выражения (λu. v) L
  - (λu. v) L = v, если начинаем слева
  - но можно всегда делать справа и зациклиться:

  (λu. v) L -β-> (λu. v) (L y) -β-> (λu. v) (L y y)…
- Выражение (λx. x x) (λx. x x) нормальной формы не имеет (выбор стратегии не спасает).

Стратегии редукции

- Теорема: Если s --> t, где t имеет нормальную форму, по последовательность редукций, в которой всегда выбирается самое левое редуцируемое выражение приводит к терму в нормальной форме.
- Это нормальный порядок редукции

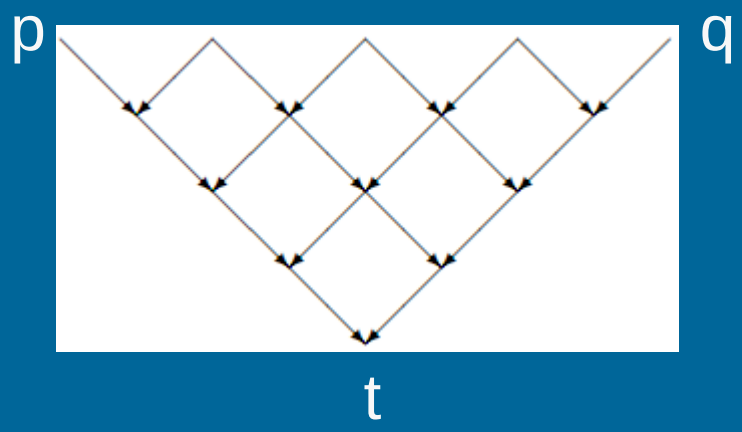
Теорема Черча-Россера

Если t --> u и t --> w, то существует терм v:

u --> v и w --> v.

Теорема об эквивалентности

- Если p = q, то существует выражение t, такое что p --> t и q -->t
- Схема доказательства:



Верхний «зигзаг» существует, так как p = q. Теорема Ч.-Р. позволяет достроить низ картинки.

Единственность нормальной формы

Следствие: Если t = v и t = w, причём v и w имеют нормальную форму, то v = w, причём цепочка редукций состоит только из α-редукций.

Значит, если нормальная форма существует, то она единственна с точность до α-редукций!

Комбинаторы

Комбинатор – λ-терм без свободных переменных.

Примеры:

- I ≡ λx. x (тождественность)
- S ≡ λf λg λx. (f x) (g x) (выделение)
- K ≡ λx λy. x (константа: K a --> λy. a)

Утверждается, что для любого λ-терма существует его эквивалент без λ-абстракций, являющийся композицией I, S, K и переменных.

I = S K K

Комбинаторы можно рассматривать как аналог машинного кода для λ-выражений.

Комбинатор неподвижной точки

- Y – комбинатор неподвижной точки, если, применив его к функции f , мы получим неподвижную точку x для f , то есть такое x, что f (x) = x. Для всех f имеем: f (Y f) = Y f
- Y-комбинатор Карри λf. (λx. f (x x))(λx. f (x x))
- Другой Y-комбинатор придумал А. Тьюринг
- Y-комбинаторы – основа для рекурсивных функций. Он даёт возможность описать λ-выражением анонимную рекурсивную функцию.

Арифметика в λ-исчислении

- $0 := \lambda f.\ \lambda x.\ x$
- $1 := \lambda f.\ \lambda x.\ f\ x$
- $2 := \lambda f.\ \lambda x.\ f\ (f\ x)$
- $3 := \lambda f.\ \lambda x.\ f\ (f\ (f\ x))$
- …

Арифметика в λ-исчислении

- SUCC := λ*n*. λ*f*. λ*x*. *f* (*n f x*)
- PLUS := λ*m*. λ*n*. λ*f*. λ*x*. *m f* (*n f x*)

т. е. PLUS := λ*n*. λ*m*. *m* SUCC *n*

- MULT := λ*m*. λ*f*. *m* (*n f*)

т. е. MULT := λ*m*. λ*n*. *m* (PLUS *n*) 0

Пример

PLUS 2 3 == (λ*m*. λ*n*. λ*f*. λ*x*. *m f* (*n f x*))

(λ*f*. λ*x*. *f* (*f x*)) (λ*f*. λ*x*. *f* (*f* (*f x*))) --->

(λ*n*. λ*f*. λ*x*. (λ*f*. λ*x*. *f* (*f x*)) *f* (*n f x*)) (λ*f*. λ*x*. *f* (*f* (*f x*))) --->

(λ*n*. λ*f*. λ*x*. (λ*a*. λ*b*. *a* (*a b*)) *f* (*n f x*)) (λ*f*. λ*x*. *f* (*f* (*f x*)))--->

(λ*n*. λ*f*. λ*x*. (λ*b*. *f* (*f b*)) (*n f x*)) (λ*f*. λ*x*. *f* (*f* (*f x*))) --->

(λ*n*. λ*f*. λ*x*. *f* (*f* (*n f x*))) (λ*f*. λ*x*. *f* (*f* (*f x*))) --->

(λ*f*. λ*x*. *f* (*f* ((λ*f*. λ*x*. *f* (*f* (*f x*))) *f x*))) -->

(λ*f*. λ*x*. *f* (*f* ((λ*a*. λ*b*. *a*(*a* (*a b*))) *f x*))) =>

(λ*f*. λ*x*. *f* (*f* (*f*(*f* (*f x*))))) -- это 5

Логика в λ-исчислении

- TRUE := λ*x*. λ*y*. *x*
- FALSE := λ*x* . λ*y*. *y*
- AND := λ*p*. λ*q*. *p q p*
- OR := λ*p*. λ*q*. *p p q*
- NOT := λ*p*. λ*a*. λ*b*. *p b a*
- IFTHENELSE := λ*p*. λ*a*. λ*b*. *p a b*