Лямбда-исчисление

Сергей Серебряков, 345 гр.

15 марта 2010

План

- Определения
- Описание синтаксиса
- Преобразования
- Нормальная форма
- Стратегии β-редукций
- Программирование на λ-исчислении
- Рекурсивные функции

Что за покемон

- λ-исчисление – формальная система для определения и применения функций
- Нетипизированное – раздел, связанный с вычислениями
- Типизированное – +типы для выражений

Кто виноват

- 1930 – Алонзо Чёрч предложил
- 1935 – парадокс Клини-Россера
- 1936 – Чёрч выделил нетипизированное
- 1940 – Чёрч предложил слаботипизированное

Идеи

- Неважно, как называется функция

  $(x, y) \rightarrow x * x + y * y$

- Неважно, как называются аргументы

  $x \rightarrow x$ эквивалентно $y \rightarrow y$

- Можно свести к функциям одной переменной (карринг)

  $(x, y) \rightarrow x * x + y * y$ эквивалентно

  $x \rightarrow (y \rightarrow x * x + y * y)$

Карринг: пример

- Было:

  $((x, y) \rightarrow x * x + y * y)\ (2, 4)$
  $= 2 * 2 + 4 * 4 = 20$

- Стало:

  $((x \rightarrow (y \rightarrow x * x + y * y))\ (2))\ (4)$
  $= (y \rightarrow 2 * 2 + y * y)\ (4)$
  $= 2 * 2 + 4 * 4 = 20$

Формальное описание

- Алфавит: буквы для переменных, символы λ и . для абстракций, скобки для приоритетов и понятности
- Множество λ-выражений Λ определяется рекурсивно:
  - если x – переменная, то $x \in \Lambda$
  - если x – переменная и $M \in \Lambda$, то $(\lambda x.M) \in \Lambda$
  - если M и $N \in \Lambda$, то $(M\ N) \in \Lambda$

Абстракции и аппликации

- Абстракция позволяет строить новые функции
  - $\lambda x.M$ означает функцию, принимающую один аргумент и подставляющую его в M вместо всех свободных вхождений x
- Аппликация – применение функции к аргументу
  - $M\ N$ означает результат вычисления функции M на входе N или процесс этого вычисления

Формальное описание: примеры

- $\lambda x.x$ – тождественная функция $x \to x$
- $(\lambda x.x)\ y$ – её применение к переменной y
- $(\lambda x.y)$ – константная функция $x \to y$
  - всегда возвращает y, независимо от входа

Нотация (лишние скобки не ок)

- Внешние скобки не пишут
  - $M\ N$ вместо $(M\ N)$
- Аппликации левоассоциативны
  - $M\ N\ P$ – это $(M\ N)\ P$
- Область действия абстракции распространяется вправо
  - $\lambda x.M\ N$ – это $\lambda x.(M\ N)$, а не $(\lambda x.M)\ N$
- Несколько абстракций подряд объединяют
  - $\lambda x.\lambda y.\lambda z.N$ пишут как $\lambda xyz.N$

Свободные и связанные переменные

- В выражении $\lambda x.M$ оператор λ *связывает* x. Все вхождения x в этом выражении – *связанные*
- Все остальные вхождения – *свободные*
  - $FV(x) = \{x\}$, если x – переменная
  - $FV(\lambda x.M) = FV(M) \setminus \{x\}$
  - $FV(M\ N) = FV(M) \cup FV(N)$
- *Замкнутое* выражение – без свободных вхождений

α-конверсия

- Неважно, как называются аргументы, поэтому имена связанных переменных можно менять в пределах одной абстракции
- $\lambda x.x$, $\lambda y.y$ – *альфа-эквивалентные* λ-выражения
- *x* и *y* **не** альфа-эквивалентны, потому что вхождения свободные

α-конверсия: исключения

- Можно менять только в пределах одной абстракции
  - $\lambda x.\lambda x.x$ эквивалентно $\lambda y.\lambda x.x$ и $\lambda x.\lambda y.y$, но не $\lambda y.\lambda x.y$
- Нельзя превращать свободное вхождение в связанное
  - замена x на y в $\lambda x.\lambda y.x$ приведёт к $\lambda y.\lambda y.y$

Подстановка

- *Подстановка E* [*V* := *E'*] – процесс замены всех свободных вхождений переменной *V* в выражение *E* на выражение *E'*
  - $x\ [x := N] \equiv N$
  - $y\ [x := N] \equiv y$, если $x \neq y$
  - $(M_1\ M_2)\ [x := N] \equiv (M_1\ [x := N])\ (M_2\ [x := N])$
  - $(\lambda x.M)\ [x := N] \equiv \lambda x.M$
  - $(\lambda y.M)\ [x := N] \equiv \lambda y.(M\ [x := N])$, если $x \neq y$ и $y \notin FV(N)$
- Иногда нужно сделать α-конверсию
  - нельзя сделать $(\lambda x.y)\ [y := x]$ и получить $(\lambda x.x)$ – *захват*
  - нужно сделать α-конверсию и получить $(\lambda z.x)$

β-редукция

- Преобразовывает выражения, применяя функции к аргументам
  - β-редукция выражения ((λ*V*.*E*) *E*') – это подстановка *E* [*V* := *E*']

- Например: ((λ*n*.*n* * 2) 7) → 7 * 2

- Если подстановка невозможна из-за захвата, требуется предварительная α-конверсия

η-конверсия

- Две функции можно считать эквивалентными, если для любого входа возвращаемые ими значения совпадают
- *η-конверсия* преобразовывает (λ*x*.*M x)* в *M*, если *x* не входит свободно в *M*

- Например, (λ*u*.*v u*) η-конвертируется в *v*
- Но (λ*u*.*u u*) не η-конвертируется в *u*

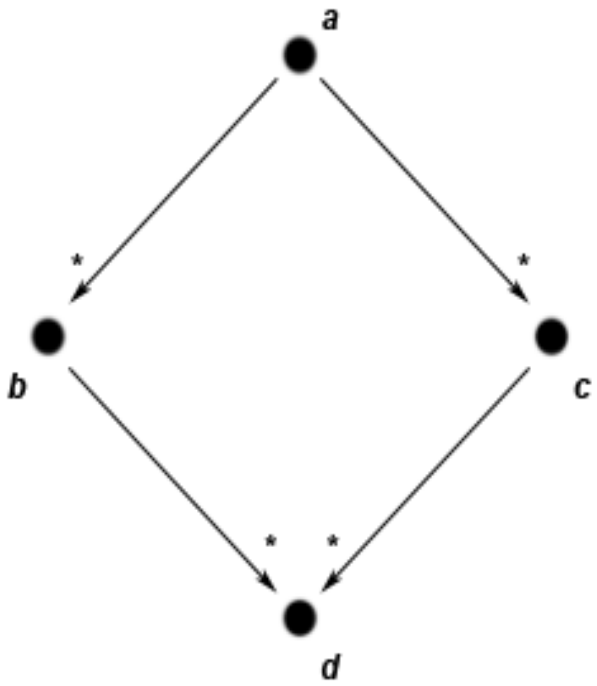
Нормальная форма

- *Редукция* – преобразование выражения (α-, β-, η-)

- Выражение имеет *нормальную форму*, если к нему нельзя применить никакие преобразования, кроме α-конверсии

Теорема Чёрча-Россера

- Для двух конечных последовательностей редукций, начатых с некоторого выражения, найдутся две другие последовательности, сводящие результаты предыдущих к одному и тому же выражению



Следствие из теоремы Чёрча-Россера

- Если нормальная форма выражения существует, то она единственна с точностью до α-конверсии

- Можно применять редукции в разном порядке – если последовательности приводят выражение в нормальную форму, то результат одинаков с точностью до α-конверсии

Стратегии β-редукций

- $(\lambda x.y)\ ((\underline{\lambda x.x\ x\ x})\ (\lambda x.x\ x\ x))$

  $\rightarrow (\lambda x.y)\ ((\underline{\lambda x.x\ x\ x})\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x))$

  $\rightarrow (\lambda x.y)\ ((\underline{\lambda x.x\ x\ x})\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x))$

  $\rightarrow \ldots$

- $(\underline{\lambda x.y})\ ((\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)) \rightarrow y$

Строгие (жадные) вычисления

- Аргументы вычисляются до применения функции
- Аппликативный порядок (самый левый и глубокий – функция упрощается до подстановки)
- Call-by-value (вычисляются аргументы, передаются в неупрощённую функцию)

Нестрогие (ленивые) вычисления

- Аргументы не вычисляются, пока не потребуются для вычисления функции
- Нормальный порядок (функция упрощается до подстановки)
- Call-by-name (аргументы подставляются в каждое вхождение)
- Call-by-need (аргумент вычисляется только в первый раз, результат потом используется)

Недетерминированные стратегии

- Полная β-редукция: любая редукция может быть применена в любое время

Нумералы Чёрча

- 0 := λfx.x
- 1 := λfx.f x
- 2 := λfx.f (f x)
- 3 := λfx.f (f (f x))
- …

- n := $\lambda fx.f^{n}\ x$

Прибавление единицы

- SUC := λ*nfx*.n f (*f x*)

$$
\begin{aligned}
\mathrm{SUC}\ n &= (\lambda n\ f\ x.\ n\ f\ (f\ x))(\lambda f\ x.\ f^n\ x) \\
&= \lambda f\ x.\ (\lambda f\ x.\ f^n\ x) f\ (f\ x) \\
&= \lambda f\ x.\ (\lambda x.\ f^n\ x)(f\ x) \\
&= \lambda f\ x.\ f^n\ (f\ x) \\
&= \lambda f\ x.\ f^{n+1}\ x \\
&= n+1
\end{aligned}
$$

Ещё операции

- PLUS := λ*mnfx*.*m f* (*n f x*)
- MULT := λ*mnf*.*m* (*n f*)
- PRED := λ*nfx*.*n* (λ*gh*.*h* (*g f*)) (λ*u*.*x*) (λ*u*.*u*)
- SUB := λ*mn*.*n* PRED *m*

Логические константы

- TRUE := $\lambda xy.x$
- FALSE := $\lambda xy.y$

- IFTHENELSE := $\lambda pab.p\ a\ b$
- IFTHENELSE TRUE $a\ b$ = TRUE $a\ b$ = a
- IFTHENELSE FALSE $a\ b$ = FALSE $a\ b$ = b

Логические операторы

- NOT *p* := IFTHENELSE *p* FALSE TRUE
- NOT = λ*p*.*p* FALSE TRUE

- AND *a b* := IFTHENELSE *a b* FALSE
- AND = λ*ab*.*a b* FALSE = λ*ab*.*a b a*

- OR *a b* := IFTHENELSE *a* TRUE *b*
- OR = λ*ab*.*a* TRUE *b* = λ*ab*.*a a b*

Логические предикаты

- ISZERO := λ*n*.*n* (λ*x*.FALSE) TRUE

$$\text{ISZERO } 0 = (\lambda f\ x.\ x)(\lambda x.\ \text{false})\ \text{true} = \text{true}$$

$$\begin{aligned}
\text{ISZERO } (n+1) &= (\lambda f\ x.\ f^{n+1}x)(\lambda x.\ \text{false})\text{true} \\
&= (\lambda x.\ \text{false})^{n+1}\ \text{true} \\
&= (\lambda x.\ \text{false})((\lambda x.\ \text{false})^{n}\ \text{true}) \\
&= \text{false}
\end{aligned}$$

Пары и кортежи

- PAIR := $\lambda xyf.f\ x\ y$
  - $(a, b) \leftrightarrow \lambda f.f\ a\ b$

- FIRST := $\lambda p.p$ TRUE
- SECOND := $\lambda p.p$ FALSE

- (a, b, c, …, z) = (a, (b, (c, (…, z))…) ↔
  ↔ $\lambda f.f\ \underline{a}\ \underline{\lambda f.f\ b\ \lambda f.f\ c\ \ldots\ \lambda f.f\ z}$

Рекурсивные функции

- Замкнутое выражение *Y* – *комбинатор неподвижной точки*, если для любого выражения *f* выполняется *f*(*Y f*) = *Y f*

Парадокс Клини-Россера

- $k = (\lambda x.\ \neg(x\ x))$
- $k\ k = (\lambda x.\ \neg(x\ x))\ k = \neg(k\ k)$
- o_O

Формулы комбинаторов HT

$$Y = \lambda f.\ (\lambda x.\ f(x\ x))(\lambda x.\ f(x\ x))$$

$$\begin{aligned} Y\,f &= (\lambda f.\ (\lambda x.\ f(x\ x))(\lambda x.\ f(x\ x)))\ f \\ &= (\lambda x.\ f(x\ x))(\lambda x.\ f(x\ x)) \\ &= f((\lambda x.\ f(x\ x))(\lambda x.\ f(x\ x))) \\ &= f(Y\ f) \end{aligned}$$

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

Использование Y-комбинаторов

- fact(n) = if (n == 0) then 1 else n*fact(n-1)
- fact := λn.IFTHENELSE
  (ISZERO n) 1 n*fact(PRED n)
- fact := (λfn.IFTHENELSE
  (ISZERO n) 1 n*f(PRED n)) fact
- F := λfn.IFTHENELSE
  (ISZERO n) 1 n*f(PRED n)
- fact := Y F