Лямбда исчисление

Лямбда исчисление

Лямбда исчисление (ЛИ) - наряду с машиной Тьюринга, одна из формальных систем, созданных для описания вычислений и исследования вычислимости алгоритмов.
Конструктивными элементами системы являются так называемые термы
Термы бывают

- **λx.y -** лямбда абстракция или, если более привычно, анонимная функция одного переменного. Где **x-** входной параметр (терм), **y** - терм, описывающий тело функции
- **(XY) -** это аппликация или, иначе, вызов терма **X** с параметром **Y** применение терма. В качестве терма X - может выступать любая абстракция. В качестве терма Y может выступать любой терм (в том числе и абстракция)
- **a**, **b**, **c** или любая другая цифро-буквенная строка. Это термы переменные

Лямбда исчисление

Примеры термов ЛИ

- **λx.x** или **(λx.x) -** определение абстракции (анонимной функции)
- **fghx** или **((fg)h)x -** аппликация; применение термов к другим термам. В применении группируем скобки влево
- **λx.λy.x** или **(λx.(λy.x)) -** определение лямбды, которая возвращает другую лямбду. В определении функций группируем скобки вправо
- **λxy.x** - сокращенная запись лямбды из предыдущего примера. Каррирование
- **λfxy.fyx** или **λf.λxy.fyx** - Так называемая функция, **FLIP,** которая применяет параметры лямбды в обратном порядке

Лямбда исчисление

Редукция. Вычисление термов

Связанная переменная (связанный терм) - это терм, содержащийся во входных параметрах лямбды

Свободная переменная(свободный терм) - это терм, не перечисленный во входных параметрах лямбды

В выражении **λx.fx** - **f,** является свободной переменной, а **x** - связанной

Лямбда исчисление

Редукция. Вычисление термов

α - переименование (αП) и α - эквивалентность ($=_\alpha$; αЭ)
αЭ - определяет отношение эквивалентности между двумя термами.
Терм P **$=_\alpha$** P\` ∀P ⇔ **λ**x.P **$=_\alpha$** **λ**y**.**P[x:=y] при условии, что y не входит в число свободных термов.
αП - это процесс переименования свободных термов перед аппликацией, необходимый для того, чтобы сохранить смысл терма после аппликации.
Пример

β -редукция -
η-преобразование, η-расширение и η-редукция -

Лямбда исчисление

Редукция. Вычисление термов

α - переименование (αП) и α - эквивалентность ($=_\alpha$; αЭ)
αЭ - определяет отношение эквивалентности между двумя термами.
Терм P $=_\alpha$ P\` ∀P ⇔ **λ**x.P **=**$_\alpha$ **λ**y**.**P[x:=y] при условии, что y не входит в число свободных термов.
αП - это процесс переименования свободных термов перед аппликацией, необходимый для того, чтобы сохранить смысл терма после аппликации.
Пример:
Терм **(λxy.x) y == λy.y** после аппликации, что неверно. Так произошло из-за того, что свободный терм y, после аппликации, стал связанным
(λxy.x) =$_\alpha$ **(λxz.x)** - проведем **αП**
Теперь **(λxz.x) y == λyz.y,** что верно.

Лямбда исчисление

Редукция. Вычисление термов

β -редукция - это процесс подстановки аргументов в функции **(λx.M)N** Термы такой формы называются редексами и обозначают то, что к функции **λx.M** будет применен аргумент **N.** Запись **$(\lambda x.M)N =_\beta M[x:=N]$,** обозначает, что после **β-редукции** все термы **x** в терме **M** будут заменены на **N**

Запись термов, не содержащей редексов, называется **нормальной формой**

Редукция производится до тех пор, пока не будет достигнута нормальная форма

- **$x[x=N]$** $\Rightarrow$ N - identity function
- **$y[x=N]$** $\Rightarrow$ y - constant function
- **$(PQ)[x=N]$** $\Rightarrow$ **$(P[x=N]\ Q[x=N])$**
- **$(\lambda y.P)[x=N]$** $\Rightarrow$ **$(\lambda y.P[x=N])$, $y \notin FV(N)$**
- **$(\lambda x.P)[x=N]$** $\Rightarrow$ **$(\lambda x.P)$**

Лямбда исчисление

Редукция. Вычисление термов

Пример редукций
(λb.λx.bxx)N $=_\beta$ λx.Nxx ; Редексов в результирующем терме нет, значит, редукция завершена
(λx.xx)(λx.xx) $=_\beta$ (λx.xx)(λx.xx) После первого шага редукции получился тот же замый терм. Это пример терма, не редуцирующегося к нормальной форме

Лямбда исчисление

Редукция. Вычисление термов

η-преобразование(η-расширение и η-редукция) и η-эквивалентность
η-расширение - это операция**,** преобразования терма содержащего свободный терм в лямбду, принимающую его в качестве параметра $\mathbf{f =_{\eta p} \lambda x.fx}$
η-эквивалентность - это эквивалентность лямбд относительно переданных в них параметров. Т.е. 2 лямбды **η-эквивалентны** если для всех входных параметров возвращают одни и те же результаты.
В scala **η-преобразованием** называется трансформация метода класс (трейта и т.д.) в функцию

```
object SomeObj {
 def method(prm: Int) = ???
}
val func = SomeObj.method _
```

Лямбда исчисление

Теорема Карри (теорема о редукции)
Если у терма есть нормальная форма, то последовательное сокращение самого левого внешнего редекса приводит к ней.
Эта стратегия редукции называется **нормальной стратегией** и сходна с передачей параметров в функции по имени
Пример
$\mathbf{(\lambda xy.x)\ z\ ((\lambda x.xx)(\lambda x.xx)) =_\beta z}$, если редуцировать левый внешний терм, зацикливания не будет, т.к. после редукции, неразрешимые термы будут удалены

$\mathbf{(\lambda xy.x)\ z\ ((\lambda x.xx)(\lambda x.xx)) =_\beta \infty}$, если с начала редуцировать параметры левого терма (аналог передачи по значению)

Лямбда исчисление

Натуральные числа Чёрча. Идея состоит в том, что каждое натуральное число кодируется 0-ым термом **z** или определенным количеством аппликацией **z** к функций получения следующего элемента. Количество аппликаций равно значению натурального числа

- **0 = λsz.z -** константная функция, возвращающая нулевой терм
- **Succ** = **λnsz.s(nsz) -** порождающая функция
- **1** = **(Succ) Zero = λsz.s((λSZ.Z)sz) = λsz.sz**
- **2** = **(Succ)(Succ)Zero** = **λsz.s(sz)**
- **3** = **λsz.s(s(sz))**
- **4** = **λsz.s(s(s(sz)))**

Лямбда исчисление

Натуральные числа Чёрча

Сложение Add M N

Мы знаем, что:

- **2 = Succ(Succ Zero)**
- **3 = Succ(Succ(Succ Zero))**
- **(Succ (Succ (Succ (Succ (Succ Zero))))) = (Succ (Succ 3)) = Add 2 3 = 5**

Отсюда можно сделать вывод, что сложение, это лямбда, которая передает вместо z в первый операнд, второй операнд т.е:

Add = λ m n s z.m s (n s z)

Задание:

- проверить, сложение на примере чисел 2 и 3
- убедится, что оно коммутативно

Лямбда исчисление

Натуральные числа Чёрча

Умножение Mul M N

Мы знаем, что:

- **2 = Succ(Succ Zero)**
- **3 = Succ(Succ(Succ Zero))**
- **Mul 3 2 = 2(2(2)) = (Succ(Succ Zero))(Succ(Succ Zero)(Succ(Succ Zero)))**

Видно, что умножение можно представить, например, как терм, складывающий M раз, свой второй операнд - N

Отсюда можно вывести, что: **Mul = λm n s z.m (Add n) Zero**

Задание:

- проверить, умножение на примере чисел 2 и 3
- убедится, что оно коммутативно

Лямбда исчисление

Логические выражения

Пусть:

- **True=λt f.t** Истина - это терм, всегда возвращающий свой левый операнд
- **False=λt f.f** Ложь - это терм, всегда возвращающий свой правый операнд
- **IF=λbxy.bxy** IF, это трехместный терм, первым параметром он принимает один из логических термов. Если **b == True** терм будет редуцирован до **x**, если **b == False**, до **y**

Таким образом термы **x** и **y** можно воспринимать как ветки условного оператора

Задание:

- проверить, что **IF True X Y = X**

Лямбда исчисление

Логические выражения

- **And = λa b.a b False**
- **Or = λa b.a True b**

Для And логика такая:

- Если a - True, то значение финального выражения зависит от значения b и => все выражение будет True, если и тоже равно True
- Если a - False, не имеет значения, что передано во втором параметра, терм будет редуцирован в False

Задание

- объяснить логику терма Or
- выполнить **lectures.types.lambda.Booleans**