Свалка из некоторых коротких тем

Косарев Дмитрий a.k.a. Kakadu

матмех СПбГУ

22 ноября 2018 г.

Мемоизация на Python

Фибоначчи курильщика

```
def fib(n):
    if n < 2: return 1
    return fib(n-1) + fib(n-2)
```

Тормоза можно пофиксить.

```
fib_memo = {}
def fib(n):
    if n < 2: return 1
    if not fib_memo.has_key(n):
        fib_memo[n] = fib(n-1) + fib(n-2)
    return fib_memo[n]
```

Мемоизация

Фибоначчи курильщика

```
fib :: Int -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Сэмулируем присваивание с помощью lazy evaluation:

```
memoize :: (Int -> a) -> (Int -> a)
memoize f = (map f [0 ..] !!)
```

Надо только вызывать... 

```
fibMemo = memoize fib
```

Мемоизация

Вполне себе норм Фибоначчи

```
import Data.Function (fix)

fib :: (Int -> Integer) -> Int -> Integer
fib f 0 = 0
fib f 1 = 1
fib f n = f (n - 1) + f (n - 2)

fibMemo :: Int -> Integer
fibMemo = fix (memoize . fib)
```

Мемоизация

Есть библиотека для всего этого через Trie

Пример был взят отсюда. Там в конце много ссылок как полиморфные функции мемоизировать и прочее

Про Untyped λ-calculus

- α-эквивалентность: $\lambda y \to (\lambda x \to x) \sim \lambda u \to (\lambda y \to y)$
- β-редукция: $(\lambda x \to M)N$
- η-expansion/conversion: $f\ x = g\ x \sim f = g$
- *Capture avoiding substitution*:
  можно ли подставлять $x \mapsto y$ в терм $\lambda y \to x + y$?
- индексы де Брауна (de Bruijn)
  $\lambda x.\lambda y.x \sim \lambda\lambda 2 \sim K$
  $\lambda x.\lambda y.\lambda z.xz(yz) \sim \lambda\lambda\lambda 31(21) \sim S$
  $\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx) \sim \lambda(\lambda 1(\lambda 1))(\lambda 21)$

Про системы типов (STLC)

Simple Typed Lambda Calculus (a.k.a. STLC)

```
Type ::= 'A' | 'B' | 'C' | ...
       | Type '->' Type
```

- 👍 Очень простая
- 👎 Слишком примитивная
- 👎 Никакого полиморфизма
- 👎 тип Y-комбинатор не выразим в STLC => нет рекурсии
- 👍 Функции всегда завершаются (strong normalization)
- 👍 В каком порядке не вычисляй $(\lambda x \to M)N$ – один ответ (congruence)
- 👍 Задача проверки типов разрешима
- 👍 Задача вывода типов разрешима

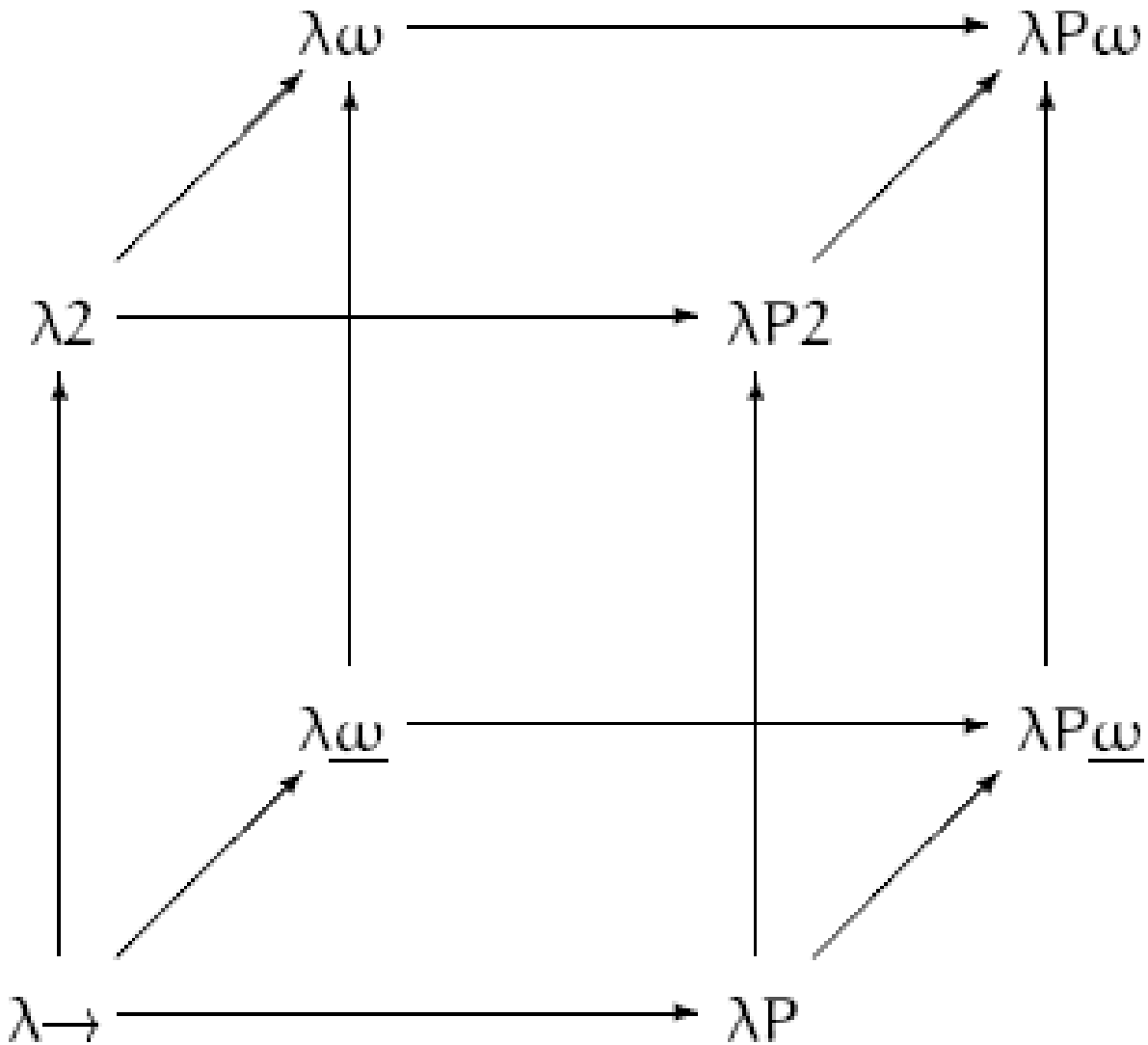
Про системы типов (HM)

Система Hindley-Milner-Damas и алгоритм вывода типов W

- 👍 Богатая!
- 👎 Но бывают ещё богаче
- 👍 Параметрический полиморфизм: типы параметризуются типами
- 👍 Y-комбинатор: `fix :: (a -> a) -> a`
- 👎 Функции не всегда завершаются завершаются (нет strong normalization)
- 👍 Задача проверки типов разрешима
- 👍 Задача вывода типов разрешима: эффективно (линейно)
- 👎 Но есть крайние (экспоненциальные) случаи (искать в книжке Пирса)

λ куб от Henk Barendregt (1991)

- $\lambda 2$ данные от типов (parametric polymorphism)
- $\lambda\omega$ типы от типов (type operators)
- $\lambda\Pi$ типы от данных (dependent types)



Про полиморфизм и let'ы

- ? Во что вычисляется
  `(\id -> (id "hi", id False)) (\x -> x)`
- ? Какой тип у результата?
- ? Какой тип у аргумента с именем `id`?
- ? Во что вычисляется?

```
let id x = x in
(id "hi", id False)
```

- ? Какой тип у результата?
- ? Какой тип у `id`?
- Сюда же пример с Ω

```
let id x = x in
id id
```

Оффтоп: упражнение

Сколько способов вы найдете, чтобы описать Ω комбинатор на Haskell?

Let-полиморфизм

На самом деле мы имеем в виду вот этот тип: $\forall a.a \to a$

```
{-# RankNTypes #-}

id :: forall a. a -> a
id x = x
```

К `id` приписалась *схема типов* (полиморфный тип с кванторами).

Важная часть системы типов HM.
Хотя некоторые (*"Let Should not be Generalised"* от SPJ et al.) пишут, что let-полиморфизм это так себе...