

# Функциональное программирование. Семинар 3.

---

## Разбор дз

1. Терм в:

- WHNF, но не в HNF -- `\x -> (\y -> y) x`
- HNF, но не в NF -- `\x -> x ((\y -> y) x)`

2. Всякие функции над числами:

- Минус пишется так же, как и плюс. Так как отрицательных чисел не завезли, то если `m <= n`, то `m - n = 0`, ну тут ничего не попишешь
- Проверка на равенство. Проверка на ноль разности очевидно не сработает, потому что смотри выше. Поэтому надо проверять на ноль разность еще и `n - m`. Зато для написания `le` достаточно проверить разность `m - n` на равенство нулю, а все остальные сравнения выразить через него и с помощью `not`

3. Самое простое решение -- толкнуть в число в качестве `s` -- `not`, а в качестве `z` -- `tru`

4. Куча всего про списки:

- `length` и `sum` пишутся очень похоже. В одном случае нужно просто делать `succ`, игнорируя элемент, а во втором -- `plus`. И там и там в качестве значения для пустого списка -- ноль
- `map`. В роли `c` -- `\e l -> cons (f e) l`, в роли `n` -- очевидно `nil`
- `reverse`. Многие как-то сложно писали, но мысль была такова, что было бы неплохо иметь терм, который бы умел склеивать два списка. Тогда:
  - `append = \l r -> l cons r`
  - `reverse = \l -> l (\e l -> append l (cons e nil)) nil`
- `tail`. Вот тут предлагалось воспользоваться тем, же, что и для вычитания единицы. Возьмем в роли начального значения пару `(nil, nil)`. На каждом шаге берем пару, берем ее второй элемент. Возвращаем новую пару, в которой первым элементом лежит запомненный нами второй, а вторым элементом -- тот же чувак, но с подклеенным в голову элементом списка. В качестве ответа просто берем первый элемент этой пары. То есть:
  - `tailz = pair nil nil`
  - `tails = \e p -> pair (snd p) (cons e (snd p))`
  - `tail = fst (l tails tailz)`

5. И там и там просто абстрагируемся и берем неподвижную точку.

6. Вот тут мне присылали кучу решений, которые отличаются между собой, но как мне кажется, они все имеют право на существование. Мысль в том, что сначала выражаем из первого уравнения  $f$ , потом подставляем во второе, берем неподвижную точку и снова подставляем в первое
7. Можно было написать честный примрек, и в качестве начальной пары взять `pair (pair 1 0) 0`, где первая пара -- это текущее и предыдущее число. А можно было сложить на примрек и взять пару `pair 1 0` и складывать и свопать каждый раз. Это сработает для сдвинутой последовательности. Есть решения, где последовательность не сдвинута, им респект, я не додумался =(

## Разминка. Контексты и утверждения о типизации

В каких контекстах верны следующие утверждения о типизации?

- $G \vdash x : a$
- $G \vdash xy : a$
- $G \vdash xy : a \rightarrow b$
- $G \vdash \lambda x \rightarrow y : a \rightarrow a \rightarrow a$
- $G \vdash \lambda x \rightarrow y : (a \rightarrow b) \rightarrow a$
- $G \vdash \lambda x \rightarrow x : (a \rightarrow b \rightarrow c) \rightarrow a \rightarrow b \rightarrow c$
- $G \vdash \lambda x \rightarrow x : (a \rightarrow b) \rightarrow a$

## Стандартные типы

У нас был бестиповый язык с кучей конструкций (почти как питон), с натуральными числами, булевыми значениями, списками и так далее. И мы определяли конструкции, полагая, что пользоваться ими будут правильно (иначе говоря, что бы в `succ` например не передавали `tru` или еще что похуже).

Давайте попробуем приписать этим конструкциям типы, чтобы такие странные термы нельзя было писать не потому что мы так хотим, а потому что правила типизации не разрешают.

Начнем с булевых значений:

```
tru = \t f -> t : a -> b -> a
fls = \t f -> f : a -> b -> b
```

Ой, у нас тут типы разные. Получается, что `Bool` это и `a -> b -> a` и `a -> b -> b`, как быть? Надо унифицировать эти типы, в данном случае достаточно взять в роли типа `b` тип `a`. Получим, что `Bool = a -> a -> a`

Попробуем теперь типизировать конструкцию `if`:

```
if = \c t e -> c t e : Bool -> a -> a -> a
```

Или, что то же самое  $(a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a \rightarrow a$ . Кому еще можно приписать такой тип? Очевидно, комбинатору  $I$ .

Теперь возьмем пары:

```
pair = \x y f -> f x y : a -> b -> (a -> b -> c) -> c
```

То есть  $\text{Pair } A B = (A \rightarrow B \rightarrow c) \rightarrow c$

Натуральные числа:

```
0 = \s z -> z : a -> b -> b
1 = \s z -> s z : (b -> c) -> b -> c
2 = \s z -> s (s z) : (b -> b) -> b -> b
...
n = \s z -> s^{n}(z) : (b -> b) -> b -> b
```

Итого  $\text{Nat} = (b \rightarrow b) \rightarrow b \rightarrow b$

Наконец, списки:

```
nil = \c n -> n : a -> b -> b
[2] = cons 2 nil = \c n -> c 2 n : (Nat -> b -> b) -> b -> b
[1, 2] = cons 1 (cons 2 nil) = \c n -> c 1 (c 2 n) : (Nat -> b -> b) -> b -> b
```

То есть  $\text{List } A = (A \rightarrow b \rightarrow b) \rightarrow b \rightarrow b$

## Вывод типов

Выведите типы комбинаторов:

```
B = \f g x -> f (g x)
B* = \f g x -> g (f x)
S = \f g x -> f x (g x)
```

Определите типы комбинаторов:

```
\x y -> x (y x)
\x y -> x y x
```

## Тип комбинатора неподвижной точки

Припишите тип комбинатору неподвижной точки, которым мы **расширили** наше исчисление. То есть мы добавили константу `fix` и правило сокращения:

```
fix f => f (fix f)
```

Ну во-первых, раз он принимает на вход функцию, то у него должен быть тип в духе `fix : (a -> b) -> ?`. Во вторых, если он "сокращается" до `f (fix f)`, то `fix f` должно иметь тип

$a$ , то есть  $\text{fix} : (a \rightarrow b) \rightarrow a$ , но справа должен стоять тип  $b$ , поэтому мы их унифицируем. То есть в итоге у нас будет  $\text{fix} : (a \rightarrow a) \rightarrow a$

## Экспансия

В лямбда-исчислении кроме операции редукции (сокращения) есть обратная операция экспансии -- расширения термина. На лекции вам говорили о том, что множество типизированных термов **замкнуто** относительно редукции. Верно ли это относительно экспансии? На самом деле нет.

Из того, что  $M \rightarrow N \ \&\& \ G \ \vdash \ N : a$  совсем не следует, что  $M : a$ . Простейший пример, который можно рассмотреть в данном случае -- это терм  $K \ I \ W$ , где  $K = \lambda x y \rightarrow x$ ,  $I = \lambda x \rightarrow x$ ,  $W = (\lambda x \rightarrow x \ x) \ (\lambda x \rightarrow x \ x)$ . Он редуцируется к  $I$  и про него верно, что  $\vdash I : a \rightarrow a$ , но  $\vdash K \ I \ W : a \rightarrow a$  неверно, так как он содержит нетипизируемый подтерм.

Для системы в стиле Чёрча экспансия не сохраняет тип лишь из-за возможного наличия нетипизируемого подтерма. Для системы в стиле Карри верно более сильное отрицание: из того, что  $M \rightarrow N \ \&\& \ G \ \vdash \ M : a \ \&\& \ G \ \vdash \ N : b$  не следует, что  $G \ \vdash \ M : b$ . Например:

Пусть  $M = S \ K$ ,  $N = K^*$ . Про  $S$  верно, что  $\vdash S : (c \rightarrow a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$ , аналогично про  $K$  верно, что  $\vdash K : c \rightarrow a \rightarrow c$ . Применение тогда будет иметь тип  $(S \ K) : (c \rightarrow a) \rightarrow c \rightarrow c$ , но  $\vdash K^* : a \rightarrow c \rightarrow c$ . Посмотрим теперь на то, как все это добро редуцировалось:

```
> S K = \g x -> K x (g x)
> = \g x -> (\y -> x) (g x)
> = \g x -> x (+)
> ~ K*
```

В редукции, помеченной плюсом потерялась информация о том, что  $g$  имеет стрелочный тип,  $(g \ x) : a \Rightarrow g : c \rightarrow a$ ,  $x : c$

Если мы перейдем в систему а-ля Чёрч, то информация перестает теряться, потому что мы таскаем за собой типовые аннотации и вынуждены опираться на них

## Домашнее задание:

Далее используется система а-ля Карри (если не указано иного). Результирующие термы подразумеваются замкнутыми

1. Заселите типы:

- $a \rightarrow b \rightarrow c \rightarrow a$  (1 балл)
- $a \rightarrow b \rightarrow c \rightarrow b$  (1 балл)

- $a \rightarrow b \rightarrow a \rightarrow a$  (1 балл)
- $a \rightarrow a \rightarrow a \rightarrow a$  (1 балл)

2. Найдите обитателей типов:

- $(d \rightarrow d \rightarrow a) \rightarrow (a \rightarrow b \rightarrow c) \rightarrow (d \rightarrow b) \rightarrow d \rightarrow c$  (1,5 балла)
- $(d \rightarrow d \rightarrow a) \rightarrow (c \rightarrow a) \rightarrow (a \rightarrow b) \rightarrow d \rightarrow c \rightarrow b$  (1,5 балла)

Сколько обитателей второго типа вы можете привести?

3. Типизируйте **по Чёрчу**:

- **S K K** (1 балл)
- **S K I** (1 балл)

4. Сконструируйте терм типа:

- $(c \rightarrow e) \rightarrow ((c \rightarrow e) \rightarrow e) \rightarrow e$

Которому нельзя было бы приписать тип:

- $(a \rightarrow (a \rightarrow e) \rightarrow e)$

(2 балла)

5. Сконструируйте терм типа:

- $((a \rightarrow b) \rightarrow a) \rightarrow (a \rightarrow a \rightarrow b) \rightarrow a$  (2 балла)
- $((a \rightarrow b) \rightarrow a) \rightarrow (a \rightarrow a \rightarrow b) \rightarrow b$  (2 балла)
- $((((a \rightarrow b) \rightarrow a) \rightarrow a) \rightarrow b) \rightarrow a$  (3 балла)