


Типичные ошибки и трудности в задачах курса «Функциональное программирование»



Михаил Симуни
simuni@mail.ru

Про что доклад

- Название в объявлении: «Что дается труднее всего на курсе «Функциональное программирование»?»
 - Конечно, монады...

- Но еще есть простые темы, которые для многих участников оказываются сложными
 - Особенно, когда начинаешь давать задачи
 - Будут обсуждаться такие темы

Про курс

Курс «Функциональное программирование», мат-мех СПбГУ,

- 4 или 3 курс
- Примерно 80 студентов
- На основе Haskell

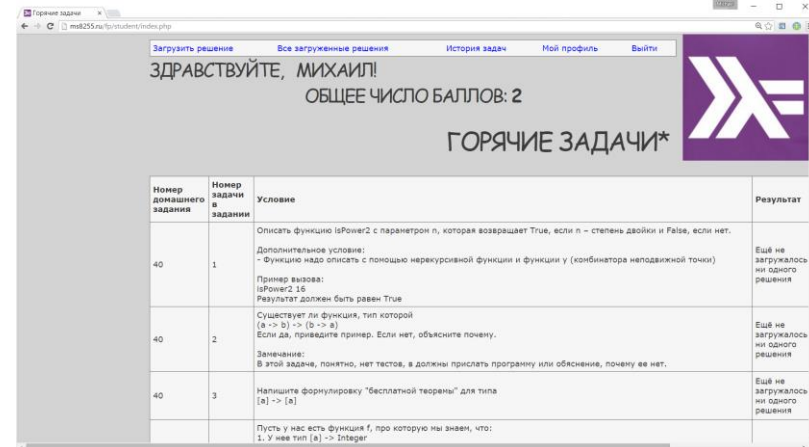
Сайт курса: <http://msimuni.wikidot.com/fp4>

Примерно 100 задач

- Есть online, частично с тестами

<http://ms8255.ru/fp/index.php>

Если кому-то интересно, можете порешать, я постараюсь проверить!
(Но сегодня будет много подсказок..)



Что проходят на курсе?

Часть 1, темы попроще

- ▣ Накапливающие параметры
- ▣ Списки, пары
- ▣ Функции высшего порядка
- ▣ `map`, `filter`, `any`, `all`
- ▣ `foldr` и `foldl`
- ▣ Алгебраические типы данных, деревья
- ▣ Карринг
- ▣ Понятие замыкания
- ▣ `List comprehension`
- ▣ Ленивые вычисления, бесконечные списки
- ▣ Полиморфные функции
- ▣ Классы

Что проходят на курсе?

Часть 2, темы посложнее

- Монады
- Понятие катаморфизма
- Алгоритм Хиндли-Милнера
- Foldable и Functor
- Continuation, failure continuation, CPS
- Представление множеств с помощью предикатов
- Символьные вычисления (упрощение выражений, интерпретатор)
- Чистое лямбда-исчисление, числа Черча (Church numerals), комбинатор неподвижной точки
- Изоморфизм Карри-Ховарда
- Parametricity и theorems for free
- В этом году немного Elm


Особенности

- ▣ Задачи на 'обычных' языках (C#)
 - Мне кажется, *очень* полезно
- ▣ Всюду, где можно, перед объяснением - задачи
 - Поэтому много простых задач

Просто как пример:

- ▣ Бета-редукция, это преобразование ...
 $(\lambda x \text{ } expr1) \text{ } expr2$
в $expr1$, в котором вместо x подставлено $expr2$...
 - Но есть два случая...

Д.з.



Задача для начала

На первом занятии:

В ФП нет побочных эффектов

... f 5 + ... f 5 ...

- Вызов функции всегда можно заменить на ее значение

$x = f\ 5$
... x + ... x ...

- Называется referential transparency
(прозрачность по ссылкам)

- Willard Quine



referentially transparent
контекст – предложение, в
котором слово можно
заменить на эквивалентное

Бегемот == гиппопотам?

- «Любой бегемот любит купаться» == «Любой гиппопотам любит купаться»
- «Все бегемоты добрые» == «Все гиппопотамы добрые»

А в каком предложении *нельзя* заменить «бегемот» на «гиппопотам»?

Не считаются:

- «рекомендую вам, донна, мою свиту. Этот валяющий дурака - кот Бегемот.»
- «Слово «бегемот» состоит из 7 букв»

Рекурсия

- ▣ *maxSin n* – максимум *sin 1, sin 2, ... sin n*

Примерно 3-5%

Рекурсия

- ▣ *maxSin n* – максимум *sin 1, sin 2, ... sin n*

maxSin 1 = sin 1

*maxSin n = if sin n > maxSin (n-1)
then sin n
else maxSin (n-1)*

- ▣ Запустите *maxSin 100* и подождите...

Примерно 3-5%

**Лишний вызов
функции в рекурсии
может очень
повлиять на
сложность**

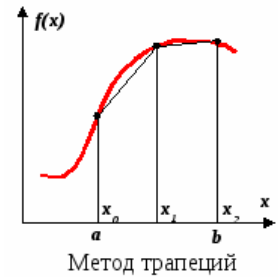


Простые функции высшего порядка

- ▣ `Integral(f, a, b)` – сосчитать интеграл методом трапеций

```
static double Integral(Func<Double, Double> f, double a, double b)
{
    double sum = 0;
    double h = (b-a)/100;
    for (double x = a; x < b; x += h) {
        sum += (f(x) + f(x + h)) * h / 2;
    }
    return sum;
}
```

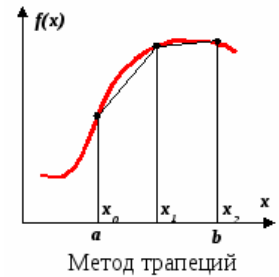
20-25%



Простые функции высшего порядка

- ▣ `Integral(f, a, b)` – сосчитать интеграл методом трапеций

```
static double Integral(Func<Double, Double> f, double a, double b)
{
    double sum = 0;
    double h = (b-a)/100;
    for (double x = a; x < b; x += h) {
        sum += (f(x) + f(x + h)) * h / 2;
    }
    return sum;
}
```




20-25%

**Вызовы функции-
параметра надо
экономить!**

- ▣ Мы же можем знать, сколько работает `f` – может быть и *очень* долго!

Использование функций высшего порядка



check и checkDiff

В самом начале, когда еще не прошли никакие стандартные функции:

- ▣ *check cond xs* - проверить, что в списке есть элемент, удовлетворяющий условию *cond*

check cond (x:xs) = ...

...

- ▣ *checkDiff xs* – проверить, что все числа разные

(Стандартных функций участники не знают, в том числе *elem*)

**Замечание
примерно у 60%**

check и checkDiff

checkDiff (x:xs) = if inList x xs then False
 else checkDiff xs

...

inList t (x:xs) = if t == x then True
 else inList t xs

**Можно
использовать check**

▣ check (\t -> t /= x) xs

Почему такой большой процент?

- ▣ Просто нет привычки использовать ФВП?
- ▣ Может быть, дело в capture?

```
checkDiff [] = True
checkDiff (x:xs) = if check (\t -> t == x) xs
                    then False
                    else checkDiff xs
```

- Тут можно написать только замыкание, с обычной функцией не получится. М.б. это чувствуют?

Как использовать tar?



Пример, где удобно использовать тар (но в условии это не сказано)

- ▣ *Нам посоветовали исправить все двойки.*

improve xs – заменить в списке все 2 на 5.



Используют тар - 50-60%

Функции высшего порядка похожи скорее на управляющие операторы

- `map` – это что-то вроде оператора `foreach`

```
map (\x ->  
    if x == 2  
    then 5  
    else x  
) xs
```

Что то вроде заголовка цикла:
«для каждого элемента `xs`...»

Что то вроде тела цикла

Очевидно, конечно, но, мне кажется, лучше явно обсудить.

Еще пример



- Пусть нам посоветовали ставить 4 и 5 всем без разбора.

generate n – составить список [4,5,4,5,4,5...] длины n.

generate n =

map (\i ->

if mod i 2 == 0

then 4

else 5

) [1..n]

Так пишут 25-30%
(но, тут, конечно, есть много
других хороших вариантов)

- map [1..n] – что-то вроде цикла арифметической прогрессии

Таблица умножения

- `multTable n` -> список `n` списков по `n` элементов

```
multTable 3 = [[1,2,3],  
               [2,4,6],  
               [3,6,9]]
```

```
multTable n =  
  map (\i ->  
    map (\j ->  
      i*j  
    ) [1..n]  
  ) [1..n]
```

*Для каждого i от 1 до n
Для каждого j от 1 до n
добавить в результат $i*j$*

**Так пишут
примерно 30%**

По-моему, важный пример!

- Человек, который умеет писать вложенные лямбда-выражения, мне кажется, в общем знает ФП:)
- Вдали показались монады :)



Много задач, где это явно удобно:

- `corner 4 = [[1,2,3,4],
[2,2,3,4],
[3,3,3,4],
[4,4,4,4]]`
- `cross 5 = [[1,0,0,0,1],
[0,1,0,1,0],
[0,0,1,0,0],
[0,1,0,1,0],
[1,0,0,0,1]]`

Еще примеры, когда надо самому догадаться использовать функции высшего порядка?

- ❑ Примеры для filter?
 - По-моему, часто слишком простые
- ❑ Примеры для foldr?
 - По-моему, часто или очень простые или не очень убедительные
- ❑ Примеры, когда надо самому определить ФПВ и потом ее использовать (какой-нибудь обход сложной структуры?)
 - Слишком громоздкие?

Замыкания



Еще задача, на C#

- С помощью Select, убрать у всех чисел в списке все цифры, кроме последних n

```
static List<int> TruncateDigits(List<int> list, int n)
{
    var res = list.Select(i => i % (int)Math.Pow(10, n));
    return res.ToList();
}
```

**Так пишут
примерно 30-40%**

Как лучше?


```
static List<int> TruncateDigits(List<int> list, int n)
{
    var res = list.Select(i => i % (int)Math.Pow(10, n));
    return res.ToList();
}
```

**Повторяющиеся вычисления лучше
выносить из лямбда-выражения**

```
static List<int> TruncateDigits(List<int> list, int n)
{
    var m = (int)Math.Pow(10, n);
    var res = list.Select(i => i % m);
    return res.ToList();
}
```

- Если смотреть на Select, как на что-то вроде цикла, но это становится более очевидно.

Вред от list comprehensions



Примерно вот это мы проходим про list comprehension

- ▣ Удобная запись для map
`[sin x | x <- [1..n]]`
- ▣ Удобная запись для filter
`[x | x <- xs, x > 0]`
- ▣ Можно сразу и map и filter
`[sin x | x <- xs, x > 0]`
- ▣ Декартово произведение
`[x*y | x<-[1..n], y<-[1..n]]`
- ▣ Более сложный пример
`[(x,y) | x<-[1..n], y<-[1..n], x^2+y^2 < n^2]`
- ▣ *let переменная = выражение*
`[(x,y) | x<-[1..n], let xx = x^2, y<-[1..n], xx+y^2 < n^2]`

Задача, которую (почти) все решают неправильно

Перечислить все способы выдать данную сумму n монетами по 2, 3 и 5 коп.

Результат должен быть списком списков вида $[k, l, m]$, где k – количество двухкопеечных монет, l – количество трехкопеечных, m – количество пятикопеечных монет.



■ Вариант 1

$\text{coins } n = [[i, k, j] \mid i \in [0..n], j \in [0..n], k \in [0..n],$
 $i*2+j*3+k*5 == n]$

Улучшения

▣ Вариант 2

```
coins n = [[i, k, j] | i<-[0 .. n `div` 2], j<-[0 .. n `div` 3],  
                    k<-[0 .. n `div` 5], i*2 + j*3 + k*5 == n]
```

▣ Вариант 3

```
coins n = [(i, k, j) | i<-[0 .. n `div` 2],  
                  let n1 = n - 2*i,      -- сколько выдать монетами по 3 и 5  
                  j<-[0 .. n1 `div` 3],  
                  let n2 = n1 - 3*j,     -- сколько выдать монетами по 5  
                  k<-[0 .. n2 `div` 5],  
                  i*2 + j*3 + k*5 == n]
```

**Одним из этих способов
пишет примерно 90%.
Можно исправить
буквально несколько
символов**

Перебор вместо вычисления

```
coins n = [(i, k, j) | i<-[0 .. n `div` 2], let n1 = n - 2*i,  
                    j<-[0 .. n1 `div` 3], let n2 = n1 - 3*j,  
                    k<-[0..n2 `div` 5], i*2 + j*3 + k*5 == n]
```

Перебираем то, что можно сосчитать

```
coins n = [(i, k, j) | i<-[0 .. n `div` 2], let n1 = n - 2*i,  
                    j<-[0 .. n1 `div` 3], let n2 = n1 - 3*j,  
                    let k = n2 `div` 5, i*2 + j*3 + k*5 == n]
```

Еще задачи на ту же тему

- Задача дается сразу же после задачи про монетки
 - amicable n – список всех пар дружественных чисел, от 1 до n (a и b дружественные если a = сумма делителей b и b – сумма делителей a)'

**Лишний перебор
у примерно 60%**

- Задача для троечников
 - Список из всех целых точек в треугольнике с вершинами $(0,0)$, $(0, n)$, $(n,0)$

**Обходят весь
квадрат 95%**

- Задача со *
- Назовем число почти простым, если его можно представить, как произведение двух простых. Проверить, можно ли представить данное n , как сумму двух почти простых чисел?

**Самые разные неэффективности от
неэффективной проверки на почти
простоту до чего то совсем ужасного)**

Катаморфизм

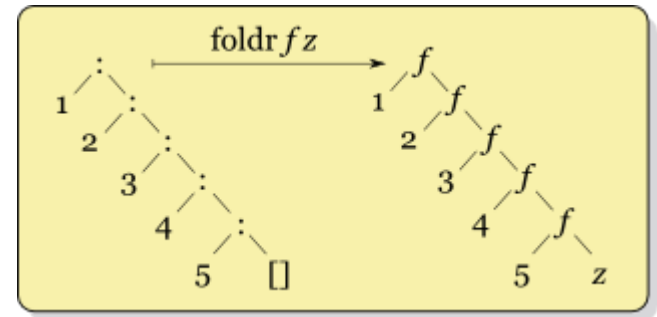


Из слайдов курса:

Как можно понимать foldr

Один из способов – заменяем в дереве внутреннего представления

- Заменяем [] на e
- Заменяем : на f
- Вычисляем то, что получится



- Обобщить на более сложные структуры?

Задача

Опишите функцию `foldTree`, которая для дерева делает что-то аналогичное `foldr` или `foldl` для списков.

Т.е. задача написать что-то, что позволяет искать сумму, произведение и т.д. для всех элементов дерева. На самом деле это можно сделать разными способами. Например, можно написать функцию `foldTree`, которая вызывается как-то так:

`foldTree (+) 0 t` — ищет сумму всех элементов в дереве

`foldTree (*) 1 t` — ищет произведение всех элементов в дереве

Но м.б. можно придумать что-то более общее? Например, *придумать такой вариант `foldTree`, который позволяет реализовывать более сложные функции (например, искать высоту дерева и т.д.)*

**Более общий вариант
придумывает примерно 25%**

Катаморфизм

```
foldTree f e (Node val l r) = let  
    resl = foldTree f e l  
    resr = foldTree f e r  
    in f val resl resr  
foldTree f e Empty = e
```

■ Сумма

```
sumTree = foldTree  
    (\v l r -> v+l+r) 0
```

■ Высота

```
height = foldTree  
    (\v l r -> 1+ max l r) (-1)
```

- Следующее д.з. – fold для типа, задающего электрические схемы (был у нас, когда проходили ADT)

Решает примерно 50%

- Т.е. почему-то понимается с трудом
 - Поэтому, наверное, полезно проходить)

Про сложные темы



Темы посложнее

- ▣ Ленивые вычисления, бесконечные списки
- ▣ Монады
- ▣ Continuation, failure continuation, CPS
- ▣ Представление множеств с помощью предикатов
- ▣ Числа Черча

Тут ошибок, на самом деле, меньше

- Или вообще не решают, или решают более-менее правильно

Разные примеры

- «Кантор»

Описать бесконечный список из всех пар положительных целых чисел

`cantor = [(i,j) | i<-[1..], j<-[1..]]`

- Привести пример функции, которая имеет тип $(a \rightarrow a) \rightarrow a \rightarrow a$

- Все, что связано с символьными вычислениями

- Упростить выражение по правилу « $0 * \text{выражение} = 0$ »

- Все, что связано с теорией (надо что-то доказать) ☹

- Существует ли алгоритм, который для двух функций, проверяет, эквивалентны они или нет?

Ответы на задачи)



Referential transparency

- ▣ «Петя думает, что все бегемоты добрые»

(Но Петя вообще не знает, кто такие гиппопотамы)



© Can Stock Photo - csp7126319

Иллюстрация к задаче 4



РОКУПКАМОНЕТ.RU



Иллюстрация к задаче 4



РОКУРКАМОНЕТ.RU

