Почему ФП?

Максим Трескин
maxim.treskin@gmail.com
@mtreskin

2 октября, 2010

- Сначала был матан: Алонзо Чёрч придумал λ-исчисления

Функциональное программирование

- Сначала был матан: Алонзо Чёрч придумал λ-исчисления
- Затем практика: Джон Маккарти открыл Lisp, содержащий реализацию λ-исчисления

В чём соль?

- Не изменяется состояние

В чём соль?

- Не изменяется состояние
- Хвостовая рекурсия

В чём соль?

- Не изменяется состояние
- Хвостовая рекурсия
- Функции первого класса

В чём соль?

- Не изменяется состояние
- Хвостовая рекурсия
- Функции первого класса
- Алгебраические типы и сопоставление шаблону

Неизменяемое состояние

- Сложно накосячить

Неизменяемое состояние

- Сложно накосячить
- Легко тестировать

Неизменяемое состояние

- Сложно накосячить
- Легко тестировать
- Легко распределять

Неизменяемое состояние

Функция с побочными действиями

```
shared = 5
def dirty(x):
    global shared
    shared = shared + x
    return shared
```

Неизменяемое состояние

Функция с побочными действиями

```
shared = 5
def dirty(x):
    global shared
    shared = shared + x
    return shared
```

```
>>> dirty(1)
6
>>> dirty(1)
7
>>> dirty(1)
8
```

Неизменяемое состояние

Функция с побочными действиями

```
shared = 5
def dirty(x):
    global shared
    shared = shared + x
    return shared
```

```
>>> dirty(1)
6
>>> dirty(1)
7
>>> dirty(1)
8
```

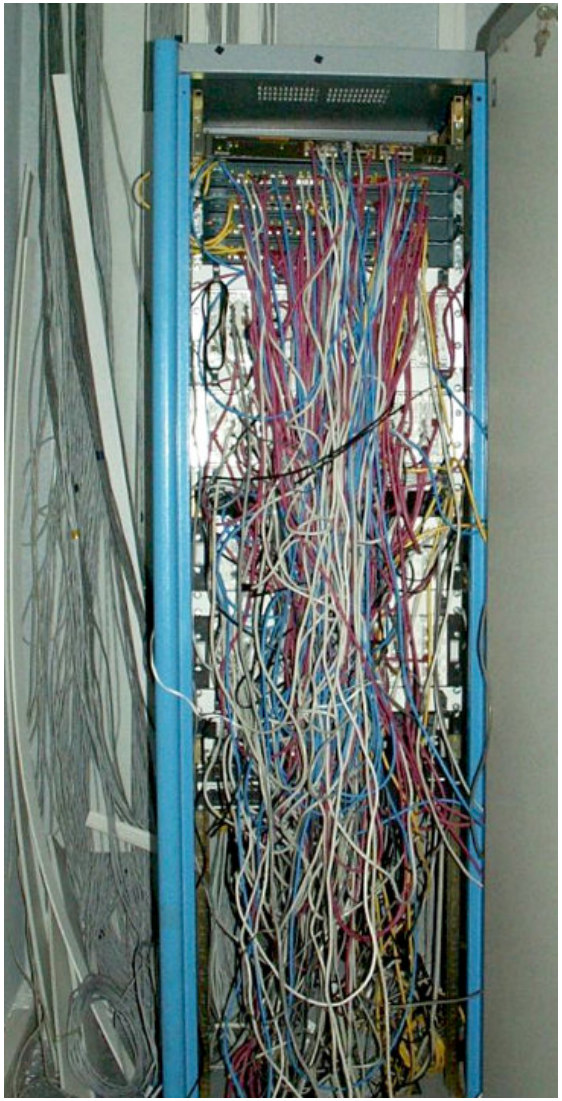
Функция без побочных действий

```
def pure(x):
    return (x + 5)
```

```
>>> pure(1)
6
>>> pure(1)
6
>>> pure(1)
6
```

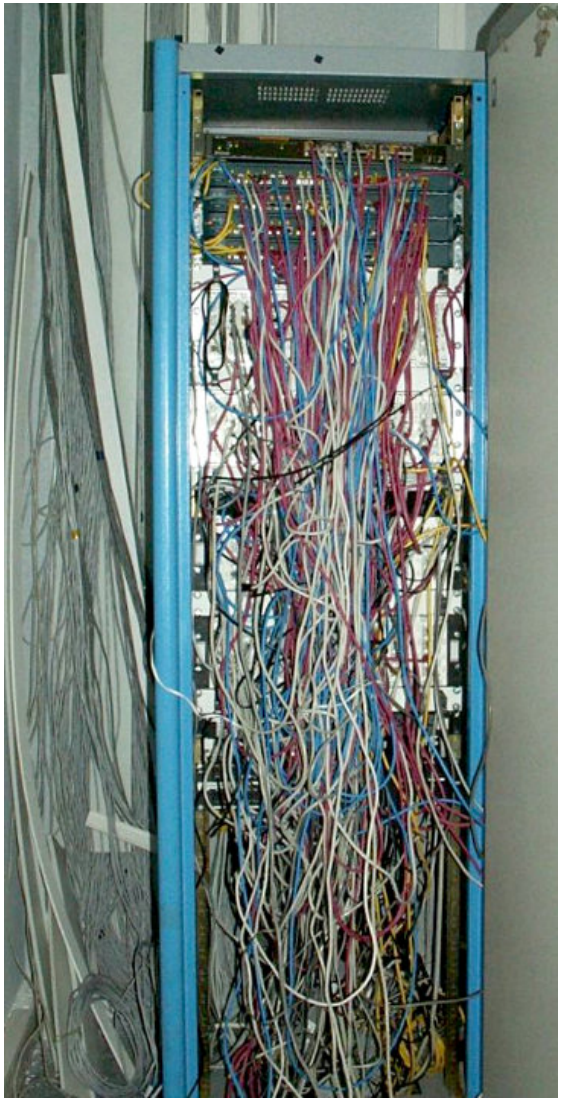
Неизменяемое состояние

Императивный код



Неизменяемое состояние

Императивный код



Функциональный код

Хвостовая рекурсия

```
State = {inc:integer = 0}
```

Хвостовая рекурсия

```
State = {inc:integer = 0}
iterate(State) ->
    NewState = State{inc + 5}
    iterate(NewState)
```

Обычное выполнение рекурсии

- Записали на стек адрес, куда надо вернуться

Обычное выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами

Обычное выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Записали на стек адрес, куда надо вернуться

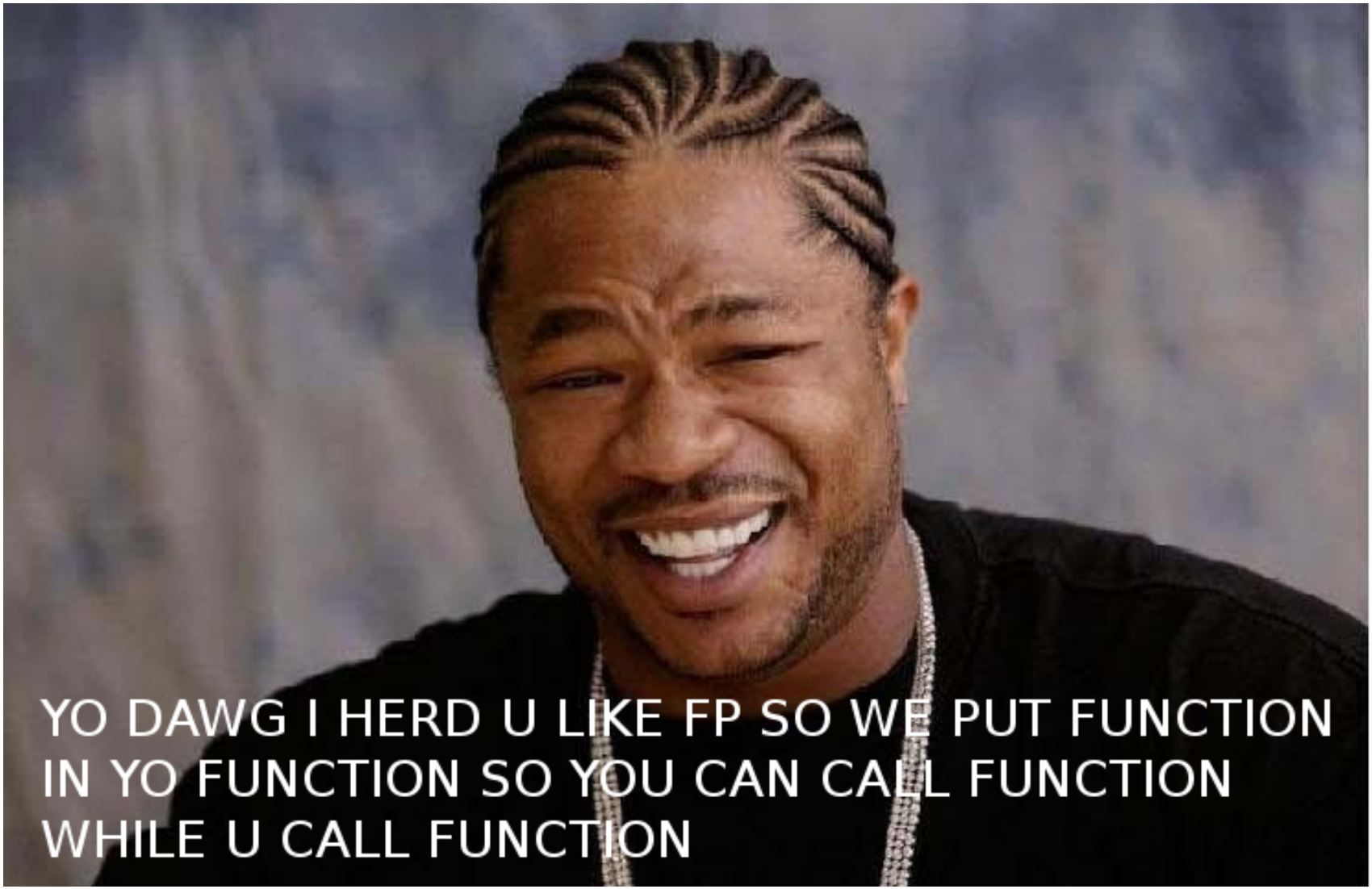
Обычное выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с новыми параметрами

Обычное выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с новыми параметрами
- Повторить до завершения памяти

Xzibit, прокачай нашу рекурсию



Xzibited выполнение рекурсии

- Записали на стек адрес, куда надо вернуться

Xzibited выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами

Xzibited выполнение рекурсии

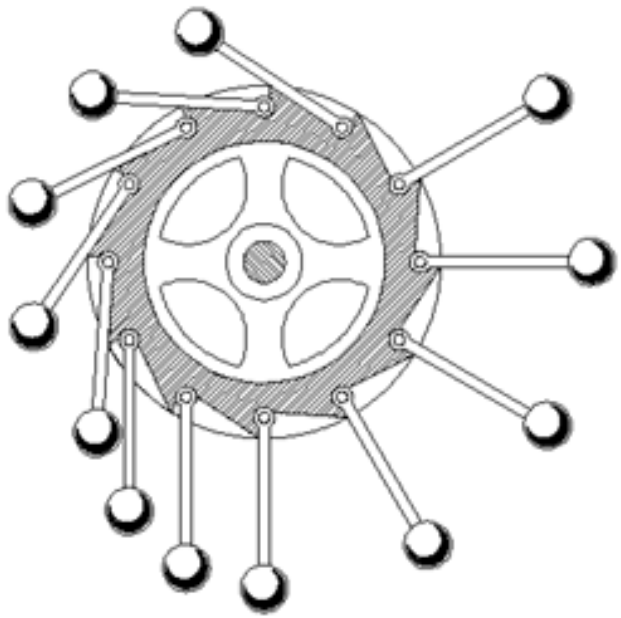
- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Вызвали функцию с новыми параметрами

Xzibited выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Вызвали функцию с новыми параметрами
- Вызвали функцию с новыми параметрами

Xzibited выполнение рекурсии

- Записали на стек адрес, куда надо вернуться
- Вызвали функцию с начальными параметрами
- Вызвали функцию с новыми параметрами
- Вызвали функцию с новыми параметрами
- Повторить до выключения питания



Функции первого класса

- Функции высшего порядка

Функции первого класса

- Функции высшего порядка
- Замыкания

Функции высшего порядка

Можно присваивать функции переменным и передавать их в другие функции

```
map f [] = []
map f (x:xs) = f x : map f xs
```

Функции высшего порядка

Можно присваивать функции переменным и передавать их в другие функции

```
map f [] = []
map f (x:xs) = f x : map f xs
add1 x = x + 1
```

Функции высшего порядка

Можно присваивать функции переменным и передавать их в другие функции

```
map f [] = []
map f (x:xs) = f x : map f xs
add1 x = x + 1
map add1 [1,2,3]
[2,3,4]
```

Замыкания

Замыкания

На vasya@pupkin:

```
M = 14,
Fun = fun(X) ->
        io:format("M = ~b, X = ~b~n", [M, X])
      end,
{eye, tolya@pipkin} ! Fun.
```

Замыкания

На vasya@pupkin:

```
M = 14,
Fun = fun(X) ->
        io:format("M = ~b, X = ~b~n", [M, X])
      end,
{eye, tolya@pipkin} ! Fun.
```

На tolya@pipkin:

```
Rcv = fun() ->
        receive M -> M(55) end
      end,
Pid = spawn(Rcv)
register(eye, Pid).
```

Замыкания

На vasya@pupkin:

```
M = 14,
Fun = fun(X) ->
        io:format("M = ~b, X = ~b~n", [M, X])
      end,
{eye, tolya@pipkin} ! Fun.
```

На tolya@pipkin:

```
Rcv = fun() ->
        receive M -> M(55) end
      end,
Pid = spawn(Rcv)
register(eye, Pid).
```

```
M = 14, X = 55
```

Алгебраические типы

Алгебраические типы

```
тип Живность =
```

Алгебраические типы

```
тип Живность =
    Человек Пол:bool, ЦветВолос:color
```

Алгебраические типы

```
тип Живность =
    Человек Пол:bool, ЦветВолос:color
  | Червь Длина:int
```

Алгебраические типы

```
тип Живность =
    Человек Пол:bool, ЦветВолос:color
  | Червь Длина:int
  | Гусеница Лапы:int
```

Алгебраические типы

```
тип Живность =
    Человек Пол:bool, ЦветВолос:color
  | Червь Длина:int
  | Гусеница Лапы:int
  | Цикада Громкость:int
```

Алгебраические типы

```
тип Живность =
    Человек Пол:bool, ЦветВолос:color
  | Червь Длина:int
  | Гусеница Лапы:int
  | Цикада Громкость:int
  | Мертвяк
```

Сопоставление шаблону

Сопоставление шаблону

```
что_делать :: Живность -> Action
```

Сопоставление шаблону

```
что_делать :: Живность -> Action
что_делать(Ж:Живность) ->
  сравним Ж с:
```

Сопоставление шаблону

```
что_делать :: Живность -> Action
что_делать(Ж:Живность) ->
  сравним Ж с:
    (Человек Пол=да, ЦветВолос=_) -> да;
```

Сопоставление шаблону

```
что_делать :: Живность -> Action
что_делать(Ж:Живность) ->
  сравним Ж с:
    (Человек Пол=да, ЦветВолос=_) -> да;
    (Цикада Громкость=$г) ->
        если $г > 100 -> убегать,
        иначе -> слушать;
```

Сопоставление шаблону

```
что_делать :: Живность -> Action
что_делать(Ж:Живность) ->
  сравним Ж с:
    (Человек Пол=да, ЦветВолос=_) -> да;
    (Цикада Громкость=$г) ->
        если $г > 100 -> убегать,
        иначе -> слушать;
    (Мертвяк) -> сообщить_куда_надо;
```

Сопоставление шаблону

```
что_делать :: Живность -> Action
что_делать(Ж:Живность) ->
  сравним Ж с:
    (Человек Пол=да, ЦветВолос=_) -> да;
    (Цикада Громкость=$г) ->
        если $г > 100 -> убегать,
        иначе -> слушать;
    (Мертвяк) -> сообщить_куда_надо;
    (_) -> созерцать.
```

А также...

- Ленивость

А также...

- Ленивость
- Карринг

А также...

- Ленивость
- Карринг
- Вывод типов

А также...

- Ленивость
- Карринг
- Вывод типов
- Просто хороший синтаксис

А также...

- Ленивость
- Карринг
- Вывод типов
- Просто хороший синтаксис
- И ещё много чего хорошего

Функциональные языки программирования

- Erlang

Функциональные языки программирования

- Erlang
- SML, OCaml, F#

Функциональные языки программирования

- Erlang
- SML, OCaml, F#
- Scala, Clojure

Функциональные языки программирования

- Erlang
- SML, OCaml, F#
- Scala, Clojure
- Haskell

Ресурсы

- http://fprog.ru
- http://erlanger.ru
- erlang-russian, scala-russian, clojure-russian на гуглогруппах
- Конференции на jabber.ru

Почему ФП?

Максим Трескин
maxim.treskin@gmail.com
@mtreskin

2 октября, 2010