

Функциональное программирование на языке F#

Введение

Юрий Литвинов

02.12.2016г

Императивное программирование

Программа как последовательность **операторов**, изменяющих **состояние** вычислителя.

Для конечных программ есть **начальное состояние**, **конечное состояние** и последовательность переходов:

$$\sigma = \sigma_1 \rightarrow \sigma_2 \rightarrow \dots \rightarrow \sigma_n = \sigma'$$

Основные понятия:

- ▶ Переменная
- ▶ Присваивание
- ▶ Поток управления
 - ▶ Последовательное исполнение
 - ▶ Ветвления
 - ▶ Циклы

Функциональное программирование

Программа как вычисление значения **выражения** в математическом смысле на некоторых входных данных.

$$\sigma' = f(\sigma)$$

- ▶ Нет состояния \Rightarrow нет переменных
- ▶ Нет переменных \Rightarrow нет циклов
- ▶ Нет явной спецификации потока управления

Порядок вычислений не важен, потому что нет состояния, результат вычисления зависит только от входных данных.

Сравним

C++

```
int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; ++i) {  
        result *= i;  
    }  
    return result;  
}
```

F#

```
let rec factorial x =  
    if x = 1 then 1 else x * factorial (x - 1)
```

Как с ЭТИМ ЖИТЬ

- ▶ Состояние и переменные «эмулируются» параметрами функций
- ▶ Циклы «эмулируются» рекурсией
- ▶ Последовательность вычислений — рекурсия + параметры

F#

```
let rec sumFirst3 ls acc i =  
    if i = 3 then  
        acc  
    else  
        sumFirst3  
            (List.tail ls)  
            (acc + ls.Head)  
            (i + 1)
```

Зачем

- ▶ Строгая математическая основа
- ▶ Семантика программ более естественна
 - ▶ Применима математическая интуиция
- ▶ Программы проще для анализа
 - ▶ Автоматический вывод типов
 - ▶ Оптимизации
- ▶ Более декларативно
 - ▶ Ленивость
 - ▶ Распараллеливание
- ▶ Модульность и переиспользуемость
- ▶ Программы более выразительны

Пример: функции высших порядков

F#

```
let sumFirst3 ls =  
    Seq.fold  
        (fun x acc -> acc + x)  
        0  
        (Seq.take 3 ls)
```

F#

```
let sumFirst3 ls = ls |> Seq.take 3 |> Seq.fold (+) 0
```

F#

```
let sumFirst3 = Seq.take 3 >> Seq.fold (+) 0
```

F#

- ▶ Типизированный функциональный язык для платформы .NET
- ▶ НЕ чисто функциональный (можно императивный стиль и ООП)
- ▶ Первый раз представлен публике в 2005 г.
- ▶ Создавался под влиянием OCaml (практически диалект OCaml под .NET)
- ▶ Использует .NET CLI
- ▶ Компилируемый и интерпретируемый
- ▶ Используется в промышленности, в отличие от многих чисто функциональных языков

Что скачать и поставить

- ▶ Под Windows — Visual Studio, из коробки
- ▶ Под Linux — Mono + MonoDevelop + F# Language Binding, из репозитория
- ▶ Прямо в браузере: <http://www.tryfsharp.org/Learn>

Пример программы

F#

```
printfn "%s" "Hello , world"
```

let-определение

Как жить без переменных

F#

```
let x = 1  
let x = 2  
printfn "%d" x
```

МОЖНО ЧИТАТЬ КАК

F#

```
let x = 1 in let x = 2 in printfn "%d" x
```

let-определение, функции

F#

```
let powerOfFour x =  
    let xSquared = x * x  
    xSquared * xSquared
```

- ▶ Позиционный синтаксис
 - ▶ Отступы строго пробелами
 - ▶ Не надо ";"
- ▶ Не надо писать типы
- ▶ Не надо писать *return*

Вложенные let-определения

F#

```
let powerOfFourPlusTwoTimesSix n =  
    let n3 =  
        let n1 = n * n  
        let n2 = n1 * n1  
        n2 + 2  
    let n4 = n3 * 6  
    n4
```

- ▶ *n3* — не функция!
- ▶ Компилятор отличает значения и функции по наличию аргументов
- ▶ Значение вычисляется, когда до *let* «доходит управление», функция — когда её вызовут. Хотя, конечно, функция — тоже значение.

Типы

F#

```
let rec f x =  
    if x = 1 then  
        1  
    else  
        x * f (x - 1)
```

F# Interactive

```
val f : x:int -> int
```

Каждое значение имеет тип, известный во время компиляции

Элементарные типы

- ▶ *int*
- ▶ *double*
- ▶ *bool*
- ▶ *string*
- ▶ ... (.NET)
- ▶ *unit* — тип из одного значения, (). Аналог void.

Таплы

F#

```
let site1 = ("scholar.google.com", 10)
let site2 = ("citeseerx.ist.psu.edu", 5)
let site3 = ("scopus.com", 4)
let sites = (site1, site2, site3)

let url, relevance = site1
let site1, site2, site3 = sites
```


Лямбды

F#

```
let primes = [2; 3; 5; 7]
let primeCubes = List.map (fun n -> n * n * n) primes
```

F# Interactive

```
> primeCubes;;
val it : int list = [8; 27; 125; 343]
```

F#

```
let f = fun x -> x * x
let n = f 4
```

Списки

Синтаксис	Описание	Пример
<code>[]</code>	Пустой список	<code>[]</code>
<code>[<i>expr</i>; ...; <i>expr</i>]</code>	Список с элементами	<code>[1; 2; 3]</code>
<code><i>expr</i> :: <i>list</i></code>	cons, добавление в голову	<code>1 :: [2; 3]</code>
<code>[<i>expr</i> .. <i>expr</i>]</code>	Промежуток целых чисел	<code>[1..10]</code>
<code>[<i>for</i> <i>x</i> <i>in</i> <i>list</i> → <i>expr</i>]</code>	Генерированный список	<code>[<i>for</i> <i>x</i> <i>in</i> 1..99 → <i>x</i> * <i>x</i>]</code>
<code><i>list</i> @ <i>list</i></code>	Конкатенация	<code>[1; 2] @ [3; 4]</code>

Примеры работы со списками

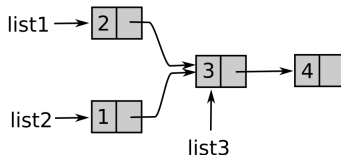
F#

```
let oddPrimes = [3; 5; 7; 11]
let morePrimes = [13; 17]
let primes = 2 :: (oddPrimes @ morePrimes)
```

F#

```
let printFirst primes =
    match primes with
    | h :: t -> printfn "First prime in the list is %d" h
    | [] -> printfn "No primes found in the list"
```

Устройство списков



F#

```

let list3 = [3; 4]
let list1 = 2 :: list3
let list2 = 1 :: list3
  
```

- ▶ Списки немутабельны
- ▶ Cons-ячейки, указывающие друг на друга
- ▶ cons за константное время, @ — за линейное

Операции над списками

Модуль Microsoft.FSharp.Collections.List

Функция	Описание	Пример	Результат
List.length	Длина списка	<i>List.length</i> [1; 2; 3]	3
List.nth	n-ый элемент списка	<i>List.nth</i> [1; 2; 3] 1	2
List.init	Генерирует список	<i>List.init</i> 3 (<i>fun i</i> → <i>i * i</i>)	[0; 1; 4]
List.head	Голова списка	<i>List.head</i> [1; 2; 3]	1
List.tail	Хвост списка	<i>List.tail</i> [1; 2; 3]	[2; 3]
List.map	Применяет функцию ко всем элементам	<i>List.map</i> (<i>fun i</i> → <i>i * i</i>) [1; 2; 3]	[1; 4; 9]
List.filter	Отбирает нужные элементы	<i>List.filter</i> (<i>fun x</i> → <i>x % 2 <> 0</i>) [1; 2; 3]	[1; 3]
List.fold	"Свёртка"	<i>List.fold</i> (<i>fun x acc</i> → <i>acc * x</i>) 1 [1; 2; 3]	6

Тип Option

Либо *Some* что-то, либо *None*, представляет возможное отсутствие значения.

F#

```
let people = [ ("Adam", None); ("Eve", None);  
               ("Cain", Some("Adam", "Eve"));  
               ("Abel", Some("Adam", "Eve")) ]
```

F#

```
let showParents (name, parents) =  
    match parents with  
    | Some(dad, mum) ->  
        printfn "%s, father %s, mother %s" name dad mum  
    | None -> printfn "%s has no parents!" name
```

Рекурсия

F#

```
let rec length l =  
    match l with  
    | [] -> 0  
    | h :: t -> 1 + length t  
  
let rec even n = (n = 0u) || odd(n - 1u)  
and odd n = (n <> 0u) && even(n - 1u)
```

Операторы | > и >>

Forward pipeline и Композиция

F#

```
let sumFirst3 ls = ls |> Seq.take 3 |> Seq.fold (+) 0
```

или

F#

```
let sumFirst3 = Seq.take 3 >> Seq.fold (+) 0
```


Каррирование, частичное применение

F#

```
let shift (dx, dy) (px, py) = (px + dx, py + dy)
let shiftRight = shift (1, 0)
let shiftUp = shift (0, 1)
let shiftLeft = shift (-1, 0)
let shiftDown = shift (0, -1)
```

F# Interactive

```
> shiftDown (1, 1);;
val it : int * int = (1, 0)
```

Использование библиотек .NET

F#

```
open System.Windows.Forms
```

```
let form = new Form(Visible = false, TopMost = true, Text = "Welcome to F#")
let textB = new RichTextBox(Dock = DockStyle.Fill, Text = "Some text")
form.Controls.Add(textB)
```

```
open System.IO
open System.Net
```

```
/// Get the contents of the URL via a web request
let http(url: string) =
    let req = System.Net.WebRequest.Create(url)
    let resp = req.GetResponse()
    let stream = resp.GetResponseStream()
    let reader = new StreamReader(stream)
    let html = reader.ReadToEnd()
    resp.Close()
    html
```

```
textB.Text <- http("http://www.google.com")
```

```
form.ShowDialog () |> ignore
```

Сопоставление шаблонов

F#

```
let urlFilter url agent =  
    match (url, agent) with  
    | "http://www.google.com", 99 -> true  
    | "http://www.yandex.ru" , _ -> false  
    | _, 86 -> true  
    | _ -> false
```

F#

```
let sign x =  
    match x with  
    | _ when x < 0 -> -1  
    | _ when x > 0 -> 1  
    | _ -> 0
```

F# — не Prolog

Не получится писать так:

F#

```
let isSame pair =  
    match pair with  
    | (a, a) -> true  
    | _ -> false
```

Нужно так:

F#

```
let isSame pair =  
    match pair with  
    | (a, b) when a = b -> true  
    | _ -> false
```

Какие шаблоны бывают

Синтаксис	Описание	Пример
(pat, \dots, pat)	Кортеж	$(1, 2, ("3", x))$
$[pat; \dots; pat]$	Список	$[x; y; 3]$
$pat :: pat$	cons	$h :: t$
$pat \mid pat$	"Или"	$[x] \mid ["X" \mid x]$
$pat \& pat$	"И"	$[p] \& [(x, y)]$
$pat \text{ as } id$	Именованный шаблон	$[x] \text{ as } inp$
id	Переменная	x
$_$	Wildcard (что угодно)	$_$
литерал	Константа	239, <i>DayOfWeek.Monday</i>
$:? type$	Проверка на тип	$:? string$

Последовательности

Ленивый тип данных

F#

```
seq {0 .. 2}  
seq {1 | .. 10000000000000 | }
```

F#

```
open System.IO  
let rec allFiles dir =  
    Seq.append  
    (dir |> Directory.GetFiles)  
    (dir |> Directory.GetDirectories  
        |> Seq.map allFiles  
        |> Seq.concat)
```

Типичные операции с последовательностями

Операция	Тип
Seq.append	$\#seq <'a> \rightarrow \#seq <'a> \rightarrow seq <'a>$
Seq.concat	$\#seq < \#seq <'a> > \rightarrow seq <'a>$
Seq.choose	$('a \rightarrow 'b\ option) \rightarrow \#seq <'a> \rightarrow seq <'b>$
Seq.empty	$seq <'a>$
Seq.map	$('a \rightarrow 'b) \rightarrow \#seq <'a> \rightarrow \#seq <'b>$
Seq.filter	$('a \rightarrow bool) \rightarrow \#seq <'a> \rightarrow seq <'a>$
Seq.fold	$('s \rightarrow 'a \rightarrow 's) \rightarrow 's \rightarrow seq <'a> \rightarrow 's$
Seq.initInfinite	$(int \rightarrow 'a) \rightarrow seq <'a>$

Задание последовательностей

F#

```
let squares = seq { for i in 0 .. 10 -> (i, i * i) }  
seq { for (i, isquared) in squares ->  
      (i, isquared, i * isquared) }
```

F#

```
let checkerboardCoordinates n =  
    seq { for row in 1 .. n do  
          for col in 1 .. n do  
              if (row + col) % 2 = 0 then  
                  yield (row, col) }
```


Ленивое чтение из файла

F#

```
let rec allFiles dir =  
    seq { for file in Directory.GetFiles(dir) -> file  
          for subdir in Directory.GetDirectories dir ->>  
            (allFiles subdir) }
```

F#

```
let reader =  
    seq {  
        use reader = new StreamReader(  
            File.OpenRead("test.txt")  
        )  
        while not reader.EndOfStream do  
            yield reader.ReadLine() }
```

Записи

F#

```
type Person =  
    { Name: string;  
      DateOfBirth: System.DateTime; }
```

F#

```
{ Name = "Bill";  
  DateOfBirth = new System.DateTime(1962, 09, 02) }  
  
{ new Person  
  with Name = "Anna"  
  and DateOfBirth = new System.DateTime(1968, 07, 23) }
```

Размеченные объединения

Discriminated unions

F#

```
type Route = int
type Make = string
type Model = string

type Transport =
| Car of Make * Model
| Bicycle
| Bus of Route
```

Известные примеры

F#

```
type 'a option =  
    | None  
    | Some of 'a
```

F#

```
type 'a list =  
    | ([])  
    | (::) of 'a * 'a list
```

Использование размеченных объединений

F#

```
type IntOrBool = I of int | B of bool
let i = I 99
let b = B true
```

F#

```
type C = Circle of int | Rectangle of int * int
```

```
[1..10]
|> List.map Circle
```

```
[1..10]
|> List.zip [21..30]
|> List.map Rectangle
```

Использование в match

F#

```
type Tree<'a> =  
    | Tree of 'a * Tree<'a> * Tree<'a>  
    | Tip of 'a  
  
let rec size tree =  
    match tree with  
    | Tree(_, l, r) -> 1 + size l + size r  
    | Tip _ -> 1
```

Пример

Дерево разбора логического выражения

F#

```
type Proposition =  
    | True  
    | And of Proposition * Proposition  
    | Or  of Proposition * Proposition  
    | Not of Proposition  
  
let rec eval (p: Proposition) =  
    match p with  
    | True      -> true  
    | And(p1, p2) -> eval p1 && eval p2  
    | Or (p1, p2)  -> eval p1 || eval p2  
    | Not(p1)     -> not (eval p1)  
  
printfn "%A" <| eval (Or(True, And(True, Not True)))
```

Взаимосвязанные типы

F#

```
type node =  
    { Name : string;  
      Links : link list }  
and link =  
    | Dangling  
    | Link of node
```


Замена цикла рекурсией

Императивное разложение на множители

F#

```
let factorizeImperative n =  
    let mutable primefactor1 = 1  
    let mutable primefactor2 = n  
    let mutable i = 2  
    let mutable fin = false  
    while (i < n && not fin) do  
        if (n % i = 0) then  
            primefactor1 <- i  
            primefactor2 <- n / i  
            fin <- true  
        i <- i + 1  
    if (primefactor1 = 1) then None  
    else Some (primefactor1, primefactor2)
```

Замена цикла рекурсией

Рекурсивное разложение на множители

F#

```
let factorizeRecursive n =  
    let rec find i =  
        if i >= n then None  
        elif (n % i = 0) then Some(i, n / i)  
        else find (i + 1)  
    find 2
```

Хвостовая рекурсия, проблема

Императивный вариант

F#

```
open System.Collections.Generic
```

```
let createMutableList() =  
    let l = new List<int>()  
    for i = 0 to 100000 do  
        l.Add(i)  
    l
```

Хвостовая рекурсия, проблема

Рекурсивный вариант, казалось бы

F#

```
let createImmutableList() =  
    let rec createList i max =  
        if i = max then  
            []  
        else  
            i :: createList (i + 1) max  
    createList 0 100000
```

Факториал без хвостовой рекурсии

F#

```
let rec factorial x =  
    if x <= 1  
    then 1  
    else x * factorial (x - 1)
```

F#

```
let rec factorial x =  
    if x <= 1  
    then  
        1  
    else  
        let resultOfRecursion = factorial (x - 1)  
        let result = x * resultOfRecursion  
        result
```

Факториал с хвостовой рекурсией

F#

```
let factorial x =  
    let rec tailRecursiveFactorial x acc =  
        if x <= 1 then  
            acc  
        else  
            tailRecursiveFactorial (x - 1) (acc * x)  
    tailRecursiveFactorial x 1
```

После декомпиляции в C#

C#

```
public static int tailRecursiveFactorial(int x, int acc)
{
    while (true)
    {
        if (x <= 1)
        {
            return acc;
        }
        acc *= x;
        x--;
    }
}
```

Паттерн “Аккумулятор”

F#

```
let rec map f list =  
    match list with  
    | [] -> []  
    | hd :: tl -> (f hd) :: (map f tl)
```

```
let map f list =  
    let rec mapTR f list acc =  
        match list with  
        | [] -> acc  
        | hd :: tl -> mapTR f tl (f hd :: acc)  
    mapTR f (List.rev list) []
```


Аккумулятор — функция

F#

```
let printListRev list =  
    let rec printListRevTR list cont =  
        match list with  
        | [] -> cont ()  
        | hd :: tl ->  
            printListRevTR tl (fun () ->  
                printf "%d " hd; cont () )  
    printListRevTR list (fun () -> printfn "Done!")
```