Генерики в F#

Юрий Литвинов

22.03.2019г

Шаблонные типы

```
type 'a list = ...
type list<'a> = ...

List.map : ('a -> 'b) -> 'a list -> 'b list
let map<'a,'b> : ('a -> 'b) -> 'a list -> 'b list =
  List.map

let rec map (f : 'a -> 'b) (l : 'a list) =
  match l with
  | h :: t -> (f h) :: (map f t)
  | [] -> []
```

Автоматическое обобщение

```
let getFirst (a, b, c) = a
let mapPair f g (x, y) = (f x, g y)
```

F# Interactive

val getFirst: 'a * 'b * 'c -> 'a
val mapPair : ('a -> 'b) -> ('c -> 'd)
   -> ('a * 'c) -> ('b * 'd)

Generic-сравнение

val compare : 'a -> 'a -> int
val (=) : 'a -> 'a -> **bool**
val (<) : 'a -> 'a -> **bool**
val (<=) : 'a -> 'a -> **bool**
val (>) : 'a -> 'a -> **bool**
val (>=) : 'a -> 'a -> **bool**
val (min) : 'a -> 'a -> 'a
val (max) : 'a -> 'a -> 'a

Сравнение сложных типов

F# Interactive

```
> ("abc", "def") < ("abc", "xyz");;
val it : bool = true
> compare (10, 30) (10, 20);;
val it : int = 1

> compare [10; 30] [10; 20];;
val it : int = 1
> compare [| 10; 30 |] [| 10; 20 |];;
val it : int = 1
> compare [| 10; 20 |] [| 10; 30 |];;
val it : int = -1
```

Generic-печать

F# Interactive

> any_to_string (Some(100, [1.0; 2.0; 3.1415]));;
val it : **string** = "Some (100, [1.0; 2.0; 3.1415])"

> sprintf "result = %A" ([1], [**true**]);;
val it : **string** = "result = ([1], [true])"
val it : int = -1

Boxing/unboxing

F# Interactive

```
> box 1;;
val it : obj = 1

> box "abc";;
val it : obj = "abc"

> let sobj = box "abc";;
val sobj : obj = "abc"

> (unbox<string> sobj);;
val it : string = "abc"

> (unbox sobj : string);;
val it : string = "abc"
```

Сериализация

open System.IO
open System.Runtime.Serialization.Formatters.Binary

```
let writeValue outputStream (x: 'a) =
   let formatter = new BinaryFormatter()
   formatter.Serialize(outputStream, box x)

let readValue inputStream =
   let formatter = new BinaryFormatter()
   let res = formatter.Deserialize(inputStream)
   unbox res
```

Сериализация, пример использования

```
let addresses = Map.ofList [
  "Jeff", "123 Main Street, Redmond, WA 98052";
  "Fred", "987 Pine Road, Phila., PA 19116";
  "Mary", "PO Box 112233, Palo Alto, CA 94301" ]

let fsOut = new FileStream("Data.dat", FileMode.Create)
writeValue fsOut addresses
fsOut.Close()

let fsIn = new FileStream("Data.dat", FileMode.Open)
let res : Map<string,string> = readValue fsIn
fsIn.Close()
```

Алгоритм Евклида, не генерик

```fsharp
let rec hcf a b =
   if a = 0 then b
   elif a < b then hcf a (b - a)
   else hcf (a - b) b
```

F# Interactive

```
val hcf : int -> int -> int

> hcf 18 12;;
val it : int = 6

> hcf 33 24;;
val it : int = 3
```

Алгоритм Евклида, генерик

```
let hcfGeneric (zero, sub, lessThan) =
  let rec hcf a b =
    if a = zero then b
    elif lessThan a b then hcf a (sub b a)
    else hcf (sub a b) b
  hcf

let hcfInt = hcfGeneric (0, (-), (<))
let hcfInt64 = hcfGeneric (0L, (-), (<))
let hcfBigInt = hcfGeneric (0I, (-), (<))
```

F# Interactive

```
val hcfGeneric: 'a * ('a -> 'a -> 'a) * ('a -> 'a -> bool)
  -> ('a -> 'a -> 'a)
```

Словари операций

```
type Numeric<'a> =
  { Zero: 'a;
    Subtract: ('a -> 'a -> 'a);
    LessThan: ('a -> 'a -> bool); }

let hcfGeneric (ops : Numeric<'a>) =
  let rec hcf a b =
    if a = ops.Zero then b
    elif ops.LessThan a b then hcf a
      (ops.Subtract b a)
    else hcf (ops.Subtract a b) b
  hcf
```

Тип функции

F# Interactive

val hcfGeneric : Numeric<'a> -> ('a -> 'a -> 'a)

Примеры использования

```
let intOps = { Zero = 0;
   Subtract = (-);
   LessThan = (<) }

let bigintOps = { Zero = 0I;
   Subtract = (-);
   LessThan = (<) }

let hcfInt = hcfGeneric intOps
let hcfBigInt = hcfGeneric bigintOps
```

Результат

F# Interactive

val hcfInt : (int -> int -> int)
val hcfBigInt : (bigint -> bigint -> bigint)

> hcfInt 18 12;;
val it : int = 6

> hcfBigInt 18102871161622323830395761
    123902817829309283048023903211;;
val it : bigint = 332241

Повышающий каст

F# Interactive

```
> let xobj = (1 :> obj);;
val xobj : obj = 1

> let sobj = ("abc" :> obj);;
val sobj : obj = "abc"
```

Понижающий каст

F# Interactive

```
> let boxedObject = box "abc";;
val boxedObject : obj

> let downcastString = (boxedObject :?> string);;
val downcastString : string = "abc"

> let xobj = box 1;;
val xobj : obj = 1

> let x = (xobj :?> string);;
error: InvalidCastException raised at or near stdin:(2,0)
```

Каст и сопоставление шаблонов

```
let checkObject (x: obj) =
  match x with
  | :? string -> printfn "The object is a string"
  | :? int -> printfn "The object is an integer"
  | _ -> printfn "The input is something else"

let reportObject (x: obj) =
  match x with
  | :? string as s ->
    printfn "The input is the string '%s'" s
  | :? int as d ->
    printfn "The input is the integer '%d'" d
  | _ -> printfn "the input is something else"
```

Гибкие ограничения

F# Interactive

```
> open System.Windows.Forms
> let setTextOfControl (c : #Control) (s:string) =
    c.Text <- s;;
val setTextOfControl: #Control -> string -> unit

> open System.Windows.Forms;;
> let setTextOfControl (c : 'a when 'a :> Control)
    (s:string) = c.Text <- s;;
val setTextOfControl: #Control -> string -> unit
```

Гибкие ограничения: пример

module Seq =

...

val append : #seq<'a> -> #seq<'a> -> seq<'a>

val concat : #seq<#seq<'a>> -> seq<'a>

...

Seq.append [1; 2; 3] [4; 5; 6]

Seq.append [| 1; 2; 3 |] [4; 5; 6]

Seq.append (seq { **for** x **in** 1 .. 3 -> x }) [4; 5; 6]

Seq.append [| 1; 2; 3 |] [| 4; 5; 6 |]

Повышающий каст: проблема

```
open System
open System.IO
let textReader =
    if DateTime.Today.DayOfWeek = DayOfWeek.Monday
    then Console.In
    else File.OpenText("input.txt")
```

F# Interactive

else File.OpenText("input.txt")
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

error: FS0001: This expression has type StreamReader
but is here used with type TextReader
stopped due to error

Повышающий каст: решение

```fsharp
let textReader =
    if DateTime.Today.DayOfWeek = DayOfWeek.Monday
    then Console.In
    else (File.OpenText("input.txt") :> TextReader)
```

Проблемы в выводе типов, методы и свойства

F# Interactive

```
> let transformData inp =
    inp |> Seq.map (fun (x, y) -> (x, y.Length));;

inp |> Seq.map (fun (x, y) -> (x, y.Length))
----------------------------------------^^^^^^^^
stdin(11,36): error: Lookup on object of indeterminate
type. A type annotation may be needed prior to this
program point to constrain the type of the object.
This may allow the lookup to be resolved.
```

Решение

```fsharp
let transformData inp =
  inp |> Seq.map (fun (x, y:string) -> (x, y.Length))
```

Уменьшение общности

```
let printSecondElements (inp : #seq<'a * int>) =
  inp
  |> Seq.iter (fun (x, y) -> printfn "y = %d" x)
```

F# Interactive

```
|> Seq.iter (fun (x, y) -> printfn "y = %d" x)
---------------------------------------^^^^^^^^^
```

stdin(21,38): warning: FS0064: This construct causes
code to be less generic than indicated by the type
annotations. The type variable 'a has been
constrained to the type 'int'.

Уменьшение общности, отладка

type PingPong = Ping | Pong

let printSecondElements (inp : #seq<PingPong * int>) =
    inp |> **Seq**.iter (**fun** (x, y) -> printfn "y = %d" x)

F# Interactive

|> Seq.iter (fun (x,y) -> printfn "y = %d" x)

--^

stdin(27,47): error: FS0001: The type 'PingPong' is not
compatible with any of the types byte, int16, int32,
int64, sbyte, uint16, uint32, uint64, nativeint,
unativeint, arising from the use of a printf-style
format string

Value Restriction

F# Interactive

> let empties = Array.create 100 [];;
------^^^^^^^^

error: FS0030: Value restriction. Type inference
has inferred the signature
val empties : '_a list []
but its definition is not a simple data constant.
Either define 'empties' as a simple data expression,
make it a function, or add a type constraint
to instantiate the type parameters.

Корректные определения

```
let emptyList = []
let initialLists = ([], [2])
let listOfEmptyLists = [[]; []]
let makeArray () = Array.create 100 []
```

F# Interactive

```
val emptyList : 'a list
val initialLists : ('a list * int list)
val listOfEmptyLists : 'a list list
val makeArray : unit -> 'a list []
```

Способы борьбы

```
let empties : int list [] = Array.create 100 []


let mapFirst = List.map fst
('a * 'b) list -> 'a list

let mapFirst inp = List.map fst inp
let printFstElements = List.map fst
  >> List.iter (printf "res = %d")

let printFstElements inp = inp
  |> List.map fst
  |> List.iter (printf "res = %d")
```

Способы борьбы (2)

```fsharp
let empties () = Array.create 100 []
let intEmpties : int list [] = empties ()
let stringEmpties : string list [] = empties ()

let emptyLists = Seq.init 100 (fun _ -> [])
let emptyLists<'a> : seq<'a list> = Seq.init
  100 (fun _ -> [])
```

Способы борьбы, результат

F# Interactive

```
> Seq.length emptyLists;;
val it : int = 100

> emptyLists<int>;;
val it : seq<int list> = seq [[]; []; []; []; ...]

> emptyLists<string>;;
val it : seq<string list> = seq [[]; []; []; []; ...]
```

Point-free

let fstGt0 xs = **List**.filter (**fun** (a, b) -> a > 0) xs

let fstGt0'1 : (int * int) **list** -> (int * int) **list** =
  List.filter (**fun** (a, b) -> a > 0)

let fstGt0'2 : (int * int) **list** -> (int * int) **list** =
  List.filter (**fun** x -> fst x > 0)

let fstGt0'3 : (int * int) **list** -> (int * int) **list** =
  List.filter (**fun** x -> ((<=) 0 << fst) x)

let fstGt0'4 : (int * int) **list** -> (int * int) **list** =
  List.filter ((<=) 0 << fst)

Арифметические операторы

```
let twice x = (x + x)
let threeTimes x = (x + x + x)
let sixTimesInt64 (x:int64) = threeTimes x + threeTimes x
```

F# Interactive

val twice : x:int -> int
val threeTimes : x:int64 -> int64
val sixTimesInt64 : x:int64 -> int64