

A linguagem de programação F#

Erick Grilo Max Fratane Vítor Lourenço

Universidade Federal Fluminense

23 de Março de 2018

Linguagem escolhida

F#:

- Originário da Microsoft Research Cambridge
- *Powered by* Don Syme, 2005
- The F Word
- Microsoft Research → F# Software Foundation, 2012

Linguagem escolhida

F# é uma linguagem multiparadigma de propósito geral:

- Funcional
- Imperativo
- Orientado à Objetos

Linguagem escolhida

● Funcional

```
> type Cor = | azul = 0 | verde = 1 | preto = 2 ;;
type Cor =
| azul = 0
| verde = 1
| preto = 2

> let silly (cor: Cor) =
-   match cor with
-   | Cor.azul -> printfn "A cor inserida é a cor azul"
-   | _ -> printfn "A cor inserida não é azul ="
-   ;;
val silly : cor:Cor -> unit

> let asd = silly Cor.verde ;;
A cor inserida não é azul =(
val asd : unit = ()

> let asd = silly Cor.azul ;;
A cor inserida é a cor azul
val asd : unit = ()

> let sum3 k = k + 3;;
val sum3 : k:int -> int

> let sum3InInterval n =
-   [1..n] |> List.map sum3 ;;
val sum3InInterval : n:int -> int list

> let verTest = sum3InInterval 5;;
val verTest : int list = [4; 5; 6; 7; 8]
```

```
> let applyf1 (f: int -> int) (a:int) = f a;;
val applyf1 : f:(int -> int) -> a:int -> int

> let applyf2 f a = f a;;
val applyf2 : f:(('a -> 'b) -> 'a -> 'b) -> 'a -> 'b

> let aNumber = 555
- ;;
val aNumber : int = 555

> let square x = x*x ;;
val square : x:int -> int

> let resp = applyf1 square aNumber;;
val resp : int = 308025

> let resp = applyf2 square aNumber;;
val resp : int = 308025
```

Linguagem escolhida

- Imperativo

```
> let simpleLoop n =  
-   for i = 0 to n do  
-       printfn "%d" i  
-   ;;  
val simpleLoop : n:int -> unit  
  
> let forTest = simpleLoop 12 ;;  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
val forTest : unit = ()
```

```
> let fat n =  
-   let mutable i = 1  
-   for n = 2 to n do  
-       i <- i * n  
-   i ;;  
val fat : n:int -> int  
  
> let testFat = fat 5 ;;  
val testFat : int = 120
```

Linguagem escolhida

- Orientado a Objetos

```
> type Professor(siape:int, primNome: string, ultimoNome:string, depto:string,funcao:string) =  
-   member this.siape = siape  
-   member this.ultimoNome = ultimoNome  
-   member this.primNome = primNome  
-   member this.depto = depto  
-   member this.funcao = funcao  
- ;;  
type Professor =  
class  
    new : siape:int * primNome:string * ultimoNome:string * depto:string *  
        funcao:string -> Professor  
    member depto : string  
    member funcao : string  
    member primNome : string  
    member siape : int  
    member ultimoNome : string  
end  
  
> let QualDepto (p:Professor) =  
-   match p.depto with  
-   | "DCC" -> printfn "Departamento de Ciência da Computação"  
-   | _ -> printfn "Que departamento é esse?"  
- ;;  
val QualDepto : p:Professor -> unit  
  
> let profChristiano = new Professor(123456789, "Christiano", "Braga", "DCC", "Professor");;  
val profChristiano : Professor  
  
> let result = QualDepto profChristiano ;;  
Departamento de Ciência da Computação  
val result : unit = ()  
  
> let result2 = QualDepto (new Professor(666, "Demoniase", "Hell", "GMA", "Torturar alunos"));;  
Que departamento é esse?  
val result2 : unit = ()
```

Tipagem

- Forte
- Estática
- Inferência de tipos

F# possui apenas uma implementação oficial. Implementação essa que utiliza o Just-In-Time.

Parsing Expression Grammars (PEG)

PEGs são estruturas com estilo similar à Gramáticas Livres de Contexto (GLC) com a possibilidade do uso de algumas estruturas de Expressões Regulares. PEGs podem ser vistas como um subconjunto de Gramáticas Livres de Contexto determinísticas:

- O operador de escolha priorizada ($/$) que PEGs empregam (diferente de $|$ para GLCs) retira o não determinismo existente em GLCs. Este operador define padrões alternativos para serem testados em ordem (da esquerda para a direita).
- Portanto, regras em GLC como $A \rightarrow a b \mid a$ e $A \rightarrow a \mid a b$ são equivalentes, mas $A \leftarrow a b / a$ e $A \leftarrow a / a b$ são diferentes:
 - No caso da regra definida em PEG, a segunda opção da segunda regra nunca será usada uma vez que a entrada esperada sempre começa com "a".

Suporte à construção de compiladores

Existe uma ferramenta chamada ScanRat que oferece suporte à construção de PEGs (Parsing Expression Grammars) em F# onde as gramáticas são vistas como sendo um conjunto de regras de produção especificadas por meio dos seguintes combinadores:

- Combinador de sequência: $+$
 - `let doisDigitos = digito + digito` (associativo à esquerda)
- Combinador de escolha priorizada: $|-$
 - `let regra = "a" |-"b"`
 - O combinador de escolha priorizada também é associativo à esquerda, mas com menor prioridade que $+$:
 - `let irParaAula7h = semDespertador + dormir |-"irParaUFF"`
- Combinador de produção: $-->$
 - Captura e converte o resultado do parsing obtido pela expressão à esquerda e passa este resultado para uma função (esperada à direita do combinador) que espera os argumentos obtidos:
 - `let twoDigitNumber = digito + digito --> fun (digito1, digito2) -> digit1 * 10 + digit2`

Suporte à construção de compiladores

- Combinadores específicos para string:
 - `~~` efetua o parsing em uma string simples. É um combinador unário que precede uma string, convertendo-a para uma regra de parsing. Logo, `~~"String"` torna "String" em uma regra de derivação.
 - `let earth = ~~"Earth"` define uma regra que efetua o *parsing* da string "Earth"
 - `oneOf` efetua o parser de um simples caracter, retornando-o
 - `let vogais = "aeiou"` define uma regra na qual é esperada algum dos símbolos especificados em vogais. A função `oneOf` é uma abreviação otimizada de: `let vogais = (~~"a" | - ~~"e" | - ~~"i" | - ~~"o" | - ~~"u") -- > fun str → str.[0]`

Suporte à construção de compiladores

- Parsing:
 - a função *parse* efetua o parsing em uma string dada como entrada:
 - `let digit = oneOf "0123456789" -- > fun c → int(c) - int('0')`
 - `let r = parse digit "3"`
 - se a operação de parsing ocorrer corretamente, o valor inteiro 3 é retornado para *r* (devido à operação definida em *digit*. O resultado do parsing é retornado por:
 - `match r with`
 - | `Success s → s.value`
 - | `Failure f → failwith "error"`

Prós e contras

- Prós
 - Interoperabilidade na .Net Framework
 - Acesso a bibliotecas em outras linguagens .Net (e.g. C#, ASP.NET, VB.NET e C++/CLI)
 - Suporte nativo no Visual Studio e amparo completo no Visual Studio Code
 - Multiplataforma com o .Net Core (suporte também para o Xamarin e Mono, mas vamos esquecer essa parte porque ninguém gosta deles)
 - É a linguagem mais bem paga globalmente segundo o StackOverflow Survey 2018
 - Quando comparada a Scala e Python (suas principais concorrentes de mercado), F# é uma linguagem funcional enquanto Scala e Python dão apenas suporte

Prós e contras

- Contras

- Quando comparada, novamente, a Scala e Python não apresenta muitas ferramentas para Data Science e Analytics
- É vendida como independente dos ambientes da Microsoft, mas para correto aproveitamento das suas vantagens precisa ser aplicada nesses ambientes

Obrigado!

A linguagem de programação F#

Erick Grilo Max Fratane Vítor Lourenço

Universidade Federal Fluminense

23 de Março de 2018