Linguagens e Paradigmas de Programação

Paradigma Funcional: Linguagens Cat e Joy

[OUTUBRO 2018]

CONTEÚDO

1. INTEGRANTES

Nome	R.A.
Cecília de Oliveira Martins	81620964
Gabriel Ferreira da Silva	81621851
Gabriel Teixeira Pinto	8162259600
James da Silva Santos	81620085
Ricardo Ferla Silva	81620904
Wellington Shiniti Kawashima	81622278

2. INTRODUÇÃO

Neste documento apresentaremos definições sobre a Linguagem Funcional (Paradigma Funcional) e também as linguagens funcionais Joy e Cat. Abordando definições das linguagens, utilizando exemplos e apresentando posteriormente a conclusão do projeto apresentado.

3. DEFINIÇÕES

Expressão	Definição
Tipada	Linguagem onde as variáveis possuem tipos específicos e não se alteram.
Dinâmica	Linguagem onde as variáveis atuam de forma contrária a tipada, não possuem tipos fixos.
C#	Linguagem de Programação multiparadigma de tipagem forte.
Javascript	Linguagem para programação cliente-side.
Postfix	Notação pós-fixa ou notação polonesa inversa, é quando operador é colocado após os seus dois operandos.

4. LINGUAGEM FUNCIONAL

4.1 Definição:

Linguagem funcional é uma linguagem de programação construída sobre e ao redor de funções lógicas ou procedimentos dentro de sua estrutura de programação. Baseia-se e é semelhante a funções matemáticas em seu fluxo de programa.

A maioria das linguagens funcionais são baseadas na notação **Lambda de Church**, a qual dentre outros detalhes, permite a definição de outras funções, sem a necessidade de especificar um nome, ou seja, definimos apenas a operação que será realizada a partir de uma entrada.

Quanto aos seus fundamentos, há vários pontos que diferem a programação funcional dos paradigmas tradicionais. Primeiro, as linguagens imperativas e orientadas a objetos tem sua implementação baseada na arquitetura de Von Newman, que trabalham basicamente com referência a células de memórias que são as variáveis e a alteração dos valores contido nas mesmas através de funções, métodos etc.

Como já dito, o paradigma funcional utiliza a notação de Church, o qual não possui o conceito de referência a célula de memórias, trabalhando apenas com valores passados para funções e produzindo resultado, sendo assim, para determinar a entrada, a função sempre produzirá a mesma saída, evitando os chamados efeitos colaterais.

Outra coisa importante sobre esse paradigma, é que a execução das expressões é controlada por condições e recursão, e não pela sequência ou pela repetição iterativa, o que é comum em linguagens imperativas.

4.2 Função:

As linguagens de programação funcional trazem consigo alto poder de expressividade, com elas podemos aplicar funções sobre conjuntos de dados, principalmente através de funções de alta ordem, nos permitindo executar cálculos complexos em chamadas simples de funções. Sua maior vantagem é sem dúvidas a abstração e também evitar os efeitos colaterais.

5. JOY

5.1 Definição:

Joy é uma linguagem de programação puramente funcional criada por **Manfred von Thun of La Trobe** em 2001 na Universidade de Melbourne, na Austrália. Enquanto todas as outras linguagens de programação funcionais são baseadas na aplicação de funções aos argumentos, o Joy é baseado na composição de funções. Toda função Joy é unária, tendo uma pilha como argumento e produzindo uma pilha como valor. Consequentemente, grande parte da Joy parece uma notação postfix comum. Na sintaxe de Joy, a composição de funções é

simplesmente a concatenação do texto. Por esse motivo, o Joy pode ser chamado de Linguagem Concatenativa.

A concatenação de programas apropriados denota a composição das funções que os programas apresentam. Algumas funções esperam programas citados no topo da pilha e os executam de muitas maneiras diferentes, efetivamente através da demarcação. Assim, onde outras linguagens funcionais usam abstração e aplicação, o Joy usa citações e combinações.

A linguagem Joy elimina a abstração de Lambda e a aplicação de funções, e as substitui por cotação de programa e composição de funções.

6. CAT

6.1 Definição:

Cat é uma linguagem de programação funcional e orientada a pilhas, criada em 2006 por Christopher Diggins inspirada na linguagem Joy. A especificação da linguagem está sob domínio público, mas sua implementação oficial primária está sob a licença MIT (Massachusetts Institute of Technology), e foi escrita em C# (C Sharp). Há uma segunda implementação oficial, escrita em JavaScript e está sob domínio público.

Joy e Cat diferem da maioria das linguagens funcionais, como Scheme e Heskell, e linguagens formais como o Cálculo Lambda, uma vez que elas se baseiam na composição de funções e não na aplicação de funções.

A Linguagem CAT foi criada para cumprir as mesmas funções que a linguagem *Joy*, com o diferencial de ser mais restrita e tipada estaticamente, enquanto a *Joy* depende muito da verificação dinâmica. Por consequência, é menos flexível que a linguagem *Joy, porém* mais segura.

Nas palavras do criador, Chistopher Diggins: "Um apelo das linguagens de pilha é que é relativamente fácil criar implementações razoavelmente eficientes para elas. As vantagens da linguagem Cat, acredito, são sua simplicidade, expressividade, extensibilidade e portabilidade.
"

Todas as instruções na linguagem Cat são funções que recebem uma pilha como entrada e retornam uma nova pilha como saída. Na verdade, todas as expressões no Cat são funções, incluindo citações.

Novas funções são definidas usando a palavra-chave 'define'. As funções não podem ser redefinidas e são visíveis somente depois de serem definidas.

No centro da linguagem Cat estão instruções primitivas para:

Operações matemáticas: add, mul, sub, div, mod, rem, inc, dec.
Operações lógicas: and, or, not, xor.
Construção de funções: compose, papply, quote.
Executando funções: apply, dip, if, while.
Manipulando a pilha: swap, dup, pop.
Lidar com listas: list, cons, uncons, head, tail, count, nth.
Comparando valores: eq, lt, gt, lteq, gteq, neq.

A seguir, algumas das bibliotecas padrão da linguagem Cat:

```
dip      = { swap quote compose apply}
rcompose = { swap compose}
papply   = { quote rcompose}
dipd     = { swap [dip] dip}
popd     = { [pop] dip}
popop    = { pop pop}
dupd     = { [dup] dip}
swapd    = { [swap] dip}
rollup   = { swap swapd}
rolldown = { swapd swa}
```

```
define myFirstCatProgram
{ "what is your name?" writeln readln "Hello " swap strcat writeln }
define addone : (int -> int)
{ 1 + }
define swapd : ('a 'b 'c -> 'b 'a 'c)
{ [swap] dip }
define dupd : ('a 'b -> 'a 'a 'b)
{ [dup] dip }
define bury : ('a 'b 'c -> 'c 'a 'b)
{ swap swapd }
define fib // compute fibonnacci
{ dup 1 <= [dup dec fib swap fib +] [] if }
define fact // compute factorial
{ dup <= 1 [dup dec fact *] [pop 1] if }
```

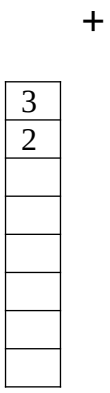
7. EXEMPLOS CAT & JOY

Um exemplo básico de como funciona o Joy:

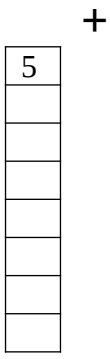
Para adicionarmos dois integers, falamos 2 e 3, e para escrever a soma deles, você digita o programa:

```
2       3       +
```

É assim que ele funciona internamente: o primeiro numeral faz com que o inteiro 2 seja colocado em uma pilha. O segundo numeral faz com que o inteiro 3 seja empurrado em cima disso.



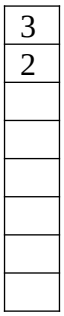
Em seguida, o operador de adição extrai os dois inteiros da pilha e empurra sua soma, 5. Portanto, a notação se parece com o postfix comum. O processador Joy lê programas como os acima até que eles sejam encerrados por um período. Só então eles são executados. No modo padrão, o item no topo da pilha (5 no exemplo) é então gravado no arquivo de saída, que normalmente é a tela



Para calcular o quadrado de um inteiro, ele deve ser multiplicado por si mesmo. Para calcular o quadrado da soma de dois inteiros, a soma deve ser multiplicada por si só. De preferência, isto deve ser feito sem calcular a soma duas vezes. O seguinte é um programa para calcular o quadrado da soma de 2 e 3:

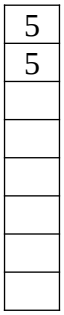
2 3 + dup *

+



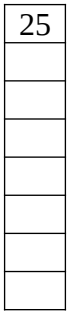
Após a soma de 2 e 3 ter sido computada, a pilha contém apenas o inteiro 5. O operador *dup* então envia outra cópia do 5 para a pilha.

dup *



Então o operador de multiplicação substitui os dois inteiros pelo seu produto, que é o quadrado de 5. O quadrado é então escrito como **25**. Além do operador *dup*, há vários outros para

reorganizar o topo da pilha. O operador pop por exemplo, remove o elemento superior, e o operador *swap* troca os dois elementos superiores.



Tudo em Joy é função. Cada função tem uma pilha como argumento e retorna outra pilha como resultado.

Em Joy, um número qualquer não é uma constante inteira como em Java, mas sim um programa que coloca esse número dentro de uma pilha. O operador * calcula o produto dos elementos do topo da pilha. Então, a definição (DEFINE) acima de uma função que eleva ao quadrado faz uma cópia do elemento do topo (através do *dup*) e multiplica os dois elementos do topo (a cópia e o elemento que já estava no topo), deixando o quadrado do elemento do topo no topo da pilha – e isso é retornado para o usuário.

Neste exemplo simples, primeiro inserimos o valor 33, depois inserimos o segundo valor 3 e por último o operando * para realizar a multiplicação. Na pilha o 33 primeiramente é inserido no topo, logo após o 3 vem acima empurrando o 33 para baixo na pilha, assim como o * vem por cima novamente. O resultando, 99, fica na posição 0 substituindo os valores digitados anteriormente. Logo após realizamos uma soma entre 1 e 2 e com o resultado multiplicamos por 5, obtendo 15 que fica logo acima na pilha, no exemplo a direita. Em Cat, o valor superior é tratado como o operando direito e o valor abaixo é tratado como o operando esquerdo. Digitamos o sinal maior > para verificar se 99 é maior que 15. O intérprete imprime o resultado, true.

Para limpar a pilha é utilizado o comando pop deixando assim a pilha vazia(empty).

```
>>     33     3      *
main stack: 99
>>     1      2      +      5      *
main stack: 99 15
>>     >
main stack: true
>>     pop
main stack: empty
```

8. CONCLUSÃO

Devido as linguagens Cat & Joy serem 100% funcionais e não ter expressões imperativas, são linguagens pouco utilizadas, pois os conceitos funcionais são ideais somente para alguns problemas em específico, o que torna o código muito mais complexo para problemas simples.

9. BIBLIOGRAFIA

- https://www.latrobe.edu.au/__data/assets/file/0007/240298/Joy-Programming.zip

- https://www.latrobe.edu.au/humanities/research/research-projects/past-projects/joy-programming-language

- http://wiki.c2.com/?JoyLanguage

- https://github.com/cdiggins/cat-language

- http://www.codecommit.com/blog/cat/the-joy-of-concatenative-languages-part-1

- https://thaniaclair.wordpress.com/2008/01/10/linguagem-cat/
Cat

- http://progopedia.com/language/cat/
Cat

- http://www.drdobbs.com/architecture-and-design/cat-a-functional-stack-based-little-lang/207200779
Cat

- https://www.codeproject.com/Articles/16247/Cat-A-Statically-Typed-Programming-Language-Interp

- https://hypercubed.github.io/joy/html/forth-joy.html