

TRABALHO DE PROGRAMAÇÃO FUNCIONAL LINGUAGEM DE PROGRAMAÇÃO I

Raimundo Alan F. Moreira, Antônio Victor C. Passos, Otavio R. Neto, José Almeida Júnior.

Bacharelado em Ciências da Computação
Centro de Ciências Exatas e Tecnologia (CCET)
Universidade Estadual Vale do Acaraú (UeVA), Brasil, 62040-370
Fone: (88) 3677-4271

alan_freire@msn.com, vpassos10@hotmail.com, otavio-rn@hotmail.com, jr.poranga@yahoo.com.br.

Resumo. *Prendemos neste artigo mostra como surgiu o estilo de programação funcional, para que é usada, o conceito desse modo de programação, os contrastes com a programação imperativa, sua aplicação no desenvolvimento de softwares e as vantagens e desvantagem de se trabalhar com o paradigma funcional.*

Palavras-chave: Linguagem Funcional, Haskell, Scheme, paradigma funcional.

Abstract. *We intend in this article shows how did the functional programming style, that is used, the concept of this mode, the contrast with imperative programming, its application in software development and the advantages and disadvantages of working with the functional paradigm.*

Keywords: Functional Language, Haskell, Scheme, functional paradigm.

1. Introdução

O cálculo lambda é um modelo de computação projetado por Alonzo Church nos anos 1930 que oferece um modo muito formal de descrever um cálculo de uma função, embora não tenha sido criada para computador foi o alicerce para a programação funcional. A primeira linguagem com esse paradigma foi LISP projetada por Jhon MacCarthy no MIT no fim dos anos 1950. A partir daí surgem outras LPs funcionais como ML, Miranda, Scheme, Haskell, etc. Nas LPs estritamente funcionais não há alocação explícita de memória nem de declaração de variáveis, pois essa operação ocorre automaticamente quando a função é invocada.

Nesse trabalho mostraremos os dados históricos, paradigma funcional, contraste com a programação imperativa, universo de aplicação, uso prático, trabalhando com a programação funcional e sua aplicação prática, vantagens e desvantagens de se trabalhar com o paradigma funcional.

2. Dados históricos

Nos anos 40 os primeiros computadores foram construídos. Nesta época devido aos altos custos, era justificável ter uma linguagem que trabalhasse o mais próximo possível da arquitetura do computador, ou seja, as primeiras linguagens de programação tinham como abstração o próprio hardware.

Como sabemos, um computador consiste de uma unidade de processamento e memória, então um programa era composto por instruções que modificavam a memória, executados pela unidade de processamento. Linguagens como C e Pascal foram marcadas por esse

estilo, chamado de programação imperativa, onde havia uma serie de atribuições executadas sequencialmente.

No entanto antes de existirem computadores, as pessoas resolviam problemas de outras formas. A principal delas, através da matemática pura. Na matemática, pelo menos nos últimos 400 anos, funções tem desempenhado um papel central. Funções representam a conexão entre parâmetros de entrada e o resultado de saída de um determinado processo. Em uma função determinística o resultado depende apenas dos seus parâmetros. Logo uma função é uma excelente forma de se especificar uma computação. Esta é a base do estilo de programação funcional.

O cálculo lambda pode ser considerado a primeira linguagem de programação funcional, embora nunca tenha sido projetada para ser realmente executada em um computador. É um modelo de computação projetado por Alonzo Church nos anos 1930 que oferece um modo muito formal de descrever um cálculo de uma função. A primeira linguagem de programação funcional criada para computadores foi LISP, desenvolvida por Jhon McCarthy no Instituto de Tecnologias de Massachusetts (MIT) no fim dos anos 1950. Mesmo não sendo uma linguagem de programação puramente funcional, LISP introduziu a maioria das características hoje encontradas nas modernas linguagens de programação funcional. Scheme foi uma tentativa posterior de simplificar e melhorar LISP. Nos anos 1970 a linguagem ML foi criada pela Universidade de Edimburgo, e David Turner desenvolveu a linguagem Miranda na Universidade de Kent. A linguagem Haskell foi lançada no fim dos anos 1980 em uma tentativa de juntar muitas idéias na pesquisa de programação funcional.

3. Paradigma funcional

É um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estado ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa. Uma função, neste sentido, pode ter ou não ter parâmetros e um simples valor de retorno. Os parâmetros são os valores de entrada da função, e o valor de retorno é o resultado da função. A definição de uma função descreve como a função será avaliada em termos de outras funções. Por exemplo, a função $f(x) = x^2 + 2$ é definida em termos de funções de exponenciação e adição. Do mesmo modo, a linguagem deve oferecer funções básicas que não requerem definições adicionais.

Linguagem funcional é o conceito de funções de primeira classe. A idéia é que você pode passar funções como parâmetros para outras funções e devolvê-los como valores. Programação funcional envolve escrever código que não muda de estado. A principal razão para isso é assim que as sucessivas chamadas para uma função irá produzir o mesmo resultado. Você pode escrever código funcional em qualquer linguagem que suporte funções de primeira classe, mas existem alguns idiomas, como Haskell, que não permitem a alteração do estado.

Em uma função determinística o resultado depende apenas dos seus parâmetros. Logo uma função é uma excelente forma de se especificar uma computação. Esta é a base do estilo de programação funcional.

Linguagens funcionais operam apenas sobre funções, as quais recebem listas de valores e retornam um valor. O objetivo da programação funcional é definir uma função que retorne um valor como a resposta do problema. Um programa funcional é uma chamada de função que normalmente chama outras funções para gerar um valor de retorno. As principais operações nesse tipo de programação são a composição de funções e a chamada recursiva de funções.

Pode-se pensar na programação funcional como simplesmente avaliação expressões. O programador define uma função para resolver um problema, e passa esta para o computador avaliar. Uma função pode envolver várias outras funções em sua definição. O computador funciona então como uma calculadora que avalia as expressões escritas pelo programador através de simplificações até chegar a uma forma normal.

A característica que domina na programação funcional é que o significado de uma expressão é seu valor, e o papel do computador é simplesmente obtê-lo. Outra característica é que uma função em uma linguagem funcional pode ser construída, manipulada e resolvida, como qualquer outro tipo de expressão matemática, usando leis algébricas. Como a entidade principal da programação funcional é a função, pode-se definir listas de funções, funções podem devolver como resultado outras funções e podem ser passadas como argumento para funções.

4. Contraste com a programação imperativa

A programação funcional pode ser contrastada com a programação imperativa. Na programação funcional parecem faltar diversas construções freqüentemente (embora incorretamente) consideradas essenciais em linguagens imperativas, como C ou Pascal. Por exemplo, em uma programação estritamente funcional, não há alocação explícita de memória, nem declaração explícita de variáveis. No entanto, essas operações podem ocorrer automaticamente quando a função é invocada; a alocação de memória ocorre para criar espaço para os parâmetros e para o valor de retorno, e a declaração ocorre para copiar os parâmetros dentro deste espaço recém-alocado e para copiar o valor de retorno de volta para dentro da função que a chama. Ambas as operações podem ocorrer nos pontos de entrada e na saída da função, então efeitos colaterais no cálculo da função são eliminados. Ao não permitir efeitos colaterais em funções, a linguagem oferece transparência referencial. Isso assegura que o resultado da função será o mesmo para um dado conjunto de parâmetros não importando onde, ou quando, seja avaliada. Transparência referencial facilita muito ambas as tarefas de comprovar a correção do programa e automaticamente identificar computações independentes para execução paralela.

Laços, outra construção de programação imperativa, está presente através da construção funcional mais geral de recursividade. Funções recursivas invocam-se a si mesmas, permitindo que uma operação seja realizada várias vezes. Na verdade, isso prova que laços são equivalentes a um tipo especial de recursividade chamada recursividade reversa. Recursividade em programação funcional pode assumir várias formas e é em geral uma técnica mais poderosa que o uso de laços. Por essa razão, quase todas as linguagens imperativas também a suportam (sendo Fortran e COBOL exceções notáveis).

5. Universo de aplicação

Linguagens de programação funcionais, especialmente as puramente funcionais, tem sido mais usadas academicamente como na área de pesquisas acadêmicas que no desenvolvimento de software. Entretanto, algumas linguagens notáveis usadas na indústria e no comércio incluem Erlang (aplicações concorrentes)R (estatística), Mathematica (matemática simbólica) J e K(análise financeira) e XSLT.

6. Usos práticos

Programas com milhares de linhas de código:

- Compiladores provadores de teoremas;
- Sistemas web;

- Serviços de chat de grande escala;
- Processamento de reclamações sobre abusos na internet;
- Simulação para a estimativa de riscos em operações financeiras;
- Programação de switches de redes na Ericsson;
- Parte do serviço de chat do Facebook;
- Serviço de filas de mensagens no twitter;
- Ensino de programação em varias universidades;
- Verificação de HW e Sw na Microsoft e na Intel;
- Estão ganhando destaque na implementação de sistemas operacionais;

7. Trabalhando com a Programação Funcional

Para avaliar uma expressão em um ambiente funcional, deve-se passá-la para o computador que irá reduzi-la a uma resposta.

Supondo que o símbolo (>) representa o prompt de um interpretador, tem-se:

```
> 29
29
```

O computador então devolve o número 29, pois ele é uma expressão em sua forma mais simples, não podendo sofrer um processo de avaliação.

Uma expressão mais complexa seria:

```
>5 * 3
15
```

Aqui o computador simplifica a expressão realizando a multiplicação.

Outra característica muito importante da programação funcional é a construção de definições. Uma lista de definições chama-se script. Exemplo de script:

```
square x = x * x
somaEmult x y = 3 * (x + y)
```

No script acima duas funções, square e somaEmult foram definidas. A função square toma um valor x e o multiplica por ele mesmo. A função somaEmult devolve a soma de seus argumentos multiplicada por 3. Nota-se que as funções foram definidas através de equações e que uma definição pode ser considerada como uma definição lógica de como a função se comporta. Em outras palavras, uma definição funcional pode ser vista como uma especificação do problema.

Tendo criado scripts pode-se usá-los em uma avaliação no computador.

Exemplos:

```
> square (2 + 3)
25
> somaEmult 1 2
9
> square (somaEmult 2 4)
324
```

O objetivo de uma definição é ligar um nome a um valor. No script anterior, por exemplo, o nome square é associado a uma função que eleva um valor ao quadrado.

8. Aplicação prática.

Um algoritmo funcional é tratado como um conjunto de funções matemáticas. O que isso quer dizer?

Para visualizar isso, vamos ao clássico exemplo do fatorial.

Matematicamente falando, $n!$ (n fatorial) é o produto de todos os números menores ou iguais a n . Ou seja, o produto entre todos os números de 1 até n . Por exemplo: $5! = 1 * 2 * 3 * 4 * 5$

Observe que $0! = 1$ já que o resultado de não se multiplicar nenhum número (identidade multiplicativa) é 1 (mais informações em http://en.wikipedia.org/wiki/Empty_product).

Normalmente existem 2 formas de se fazer isso: recursiva e iterativa:

Recursiva:

```
int fatorial(int n)
{
    if(n <= 1)
        return 1;
    return n * fatorial(n - 1);
}
```

Iterativa:

```
int fatorial(int n)
{
    int r = 1;

    while(n > 1)
        r *= n--;
    return r;
}
```

Isso em uma linguagem funcional (no caso, Haskell), ficaria como:

```
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)
```

Ou seja, literalmente o que diz a definição: o produto entre todos os números de 1 até n .

Uma das principais vantagens disso é a redução da quantidade de código. Considerando que a declaração do tipo não é necessariamente obrigatória (apesar de uma boa prática) a função pode ser implementada em apenas uma ou duas linhas, respeitando os padrões de indentação e tudo mais.

Soma de 1 a 100 em C e em Haskell

Operação baseada em atribuição de variáveis

```
int total = 0;
for (i = 1; i <= 100; i++)
```

```
total = total + 1;
```

A mesma variável(total) muda de valor 100 vezes durante a operação (efeito colateral)

Soma de 1 a 100 em Haskell

Baseada na função Sum e na ideia de lista implícita

```
s1_100 :: integer
s1_100 = sum [1..100]
```

- O método de programação é aplicação de funções
- Conseqüência de abordagem funcional

```
s1_100 = sum [1..100]
```

Uma vez estabelecido o valor de s1_100

- tem um tipo definido o valor não pode ser mudado.
- O valor pode ser usado como argumento de funções.
- Em termos de c++, é como se todos os valores fossem const

9. Vantagens de se trabalhar com o paradigma funcional

Programação funcional permite a codificação com menos potencial para erros, pois cada componente é completamente isolado. Além disso, a recursividade usado em funções de primeira-classe permite que as provas para correção sejam mais simples do que normalmente espelhar a estrutura do código.

Algumas das vantagens da programação funcional incluem o fato de que os programas funcionais são mais curtos, e mais fáceis de modificar (porque não há efeitos ocultos global a ter em conta).

Manipulação mais simples de programas, boa legibilidade.

Não permitir efeitos colaterais em funções, a linguagem oferece transparência referencial Modularidade, um conceito onde o sistema ou software é dividido em partes distintas.

Compõe ferramentas necessário para um programa mais legível com uma melhor manutenção e melhor desempenho por meio da programação.

10. Desvantagens de se trabalhar com o paradigma funcional

As linguagens funcionais também tendem a ter ambientes de execução muito grande. Haskell é uma exceção (GHC executáveis são quase tão pequenos quanto programas C, tanto em tempo de compilação e execução), mas SML, Common Lisp, Scheme e programas exigem sempre muita memória.

Os programas funcionais tendem a correr lentamente (por causa de todos copiando o que tem que fazer), e eles tendem a não interagem bem com outros programas, processos do sistema operacional.

11. Conclusão

Com esse artigo podemos nos familiarizar com a programação funcional, vendo que a mesma tem sido mais usada para pesquisas acadêmicas que para a produção de software.

Percebemos que a programação funcional envolve escrever códigos que não muda de estado, que não há uma alocação de memória nem de variáveis explícita. Foi observado que este tipo de programação tem um menor potencial para erros, devido ser completamente isolado cada componente, mas seu consumo de memória é muito alto na maioria da linguagem de programação funcional.

12. Referências

- [1] http://pt.wikipedia.org/wiki/Programação_funcional
- [2] http://pt.wikipedia.org/wiki/Cálculo_lambda
- [3] http://www.cin.ufpe.br/~mrsl/PLC/02-Introducao_Haskell.pdf
- [4] <http://www.inf.ufrgs.br/gppd/disc/cmp134/trabs/T1/991/ProgFunc/pf.html>
- [5] Varejão, Flávio Miguel. *Linguagens de programação: Java, C e C e outras : conceitos e técnicas*. Rio de Janeiro: Elsevier, 2004
- [6] Valença, José Manuel; Barros, José Bernardo, *Fundamentos da Computação, Livro II: Programação Funcional*. Universidade Aberta, 1999.
- [7] <http://informacaocomdiversao.blogspot.com/2009/02/paradigma-funcional.html>
- [8] <http://pt.scribd.com/doc/45211091/Apostila-Lisp>