Aimée Ferreira
Ítalo Médici

Linguagens de Programação Funcional

Cap. 15

Introdução

Baseado em funções matemáticas, o paradigma de programação funcional é a base de projeto dos estilos de linguagem não imperativos mais importantes. Esse estilo de programação levou a uma expansão significativa das áreas da computação, usado em linguagens funcionais.

Funções matemáticas

Uma função matemática é um mapeamento de membros de um conjunto, chamado de conjunto domínio, para outro, chamado de conjunto imagem. As funções são geralmente aplicadas a um elemento específico do conjunto domínio, fornecido como parâmetro para a função.

Funções simples

Uma função matemática é um mapeamento de membros de um conjunto, chamado de conjunto domínio, para outro, chamado de conjunto imagem. As funções são geralmente aplicadas a um elemento específico do conjunto domínio, fornecido como parâmetro para a função.

$$cube(x) \equiv x * x * x,$$

Funções simples

Aplicações de funções são especificadas pot **nome da função** e um elemento de **conjunto domínio**.
O elemento da imagem é obtido ao avaliarmos a expressão da função com o domínio substituído para as ocorrências do **parâmetro**.
Cada ocorrência de um **parâmetro** é vinculada a um valor do conjunto domínio e é uma constante durante a avaliação. Por exemplo, considere a seguinte avaliação de cube(x):

$$cube\,(2) = 2 * 2 * 2 = 8$$

Formas funcionais

Uma forma funcional, é aquela que **recebe uma ou mais funções como parâmetros**, ou que **leva a uma função como resultado**, ou ambos. Um tipo de forma funcional é a composição funcional, que tem dois parâmetros funcionais e leva a uma função cujo valor é o primeiro parâmetro de função real aplicado ao resultado do segundo, escrita como uma expressão, usando ° como um operador, como em:

$$h \equiv f \circ g \quad \rightarrow \quad \begin{aligned} f(x) &\equiv x + 2 \\ g(x) &\equiv 3 * x \end{aligned} \quad \rightarrow \quad \begin{aligned} h(x) &\equiv f(g(x)) \\ h(x) &= (3 * x) + 2 \end{aligned}$$

Fundamentos das linguagens de programação funcional

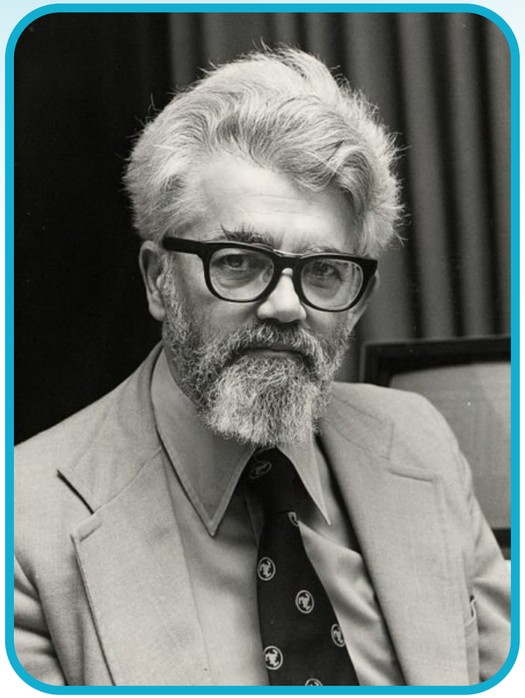
O objetivo do projeto de uma linguagem de programação funcional é imitar as funções matemáticas ao máximo possível.

Uma linguagem funcional fornece um conjunto de funções primitivas, um conjunto de formas funcionais para construir funções complexas a partir dessas funções primitivas, uma operação de aplicação de função e alguma estrutura ou estruturas para representar dados. Essas estruturas são usadas para representar os parâmetros e os valores computados pelas funções. Se uma linguagem funcional é bem projetada, ela requer apenas um número relativamente pequeno de funções primitivas.

A Primeira Linguagem de Programação Funcional: LISP

- Desenvolvida por John McCarthy no MIT, em 1959, Lisp é a linguagem mais antiga e mais utilizada (ou uma de suas descendentes).

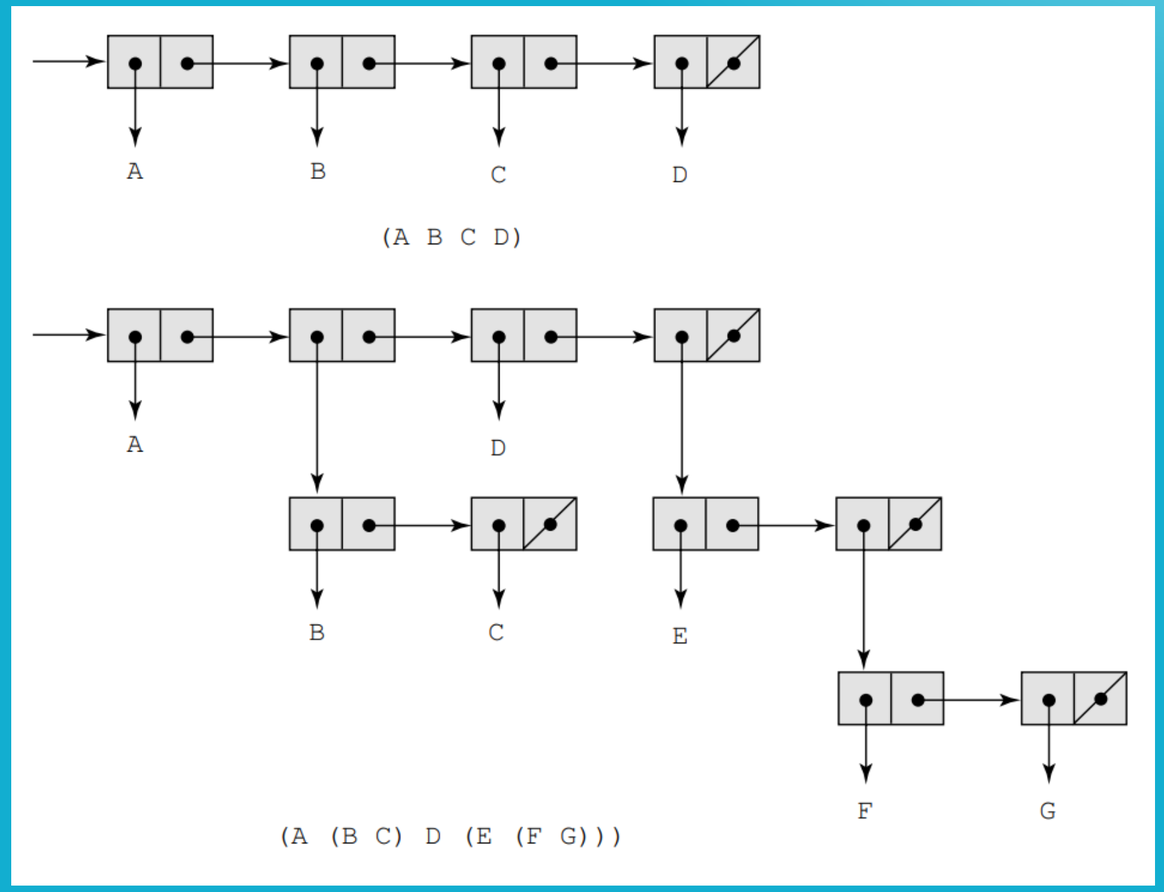
- Com exceção da primeira versão, todos os dialetos de Lisp incluem recursos imperativos.



As listas são especificadas em Lisp ao delimitarmos seus elementos com parênteses. Os elementos de listas simples são restritos aos átomos.
As estruturas também podem ser especificadas com parênteses.

$$(A\ B\ C\ D)$$

$$(A\ (B\ C)\ D\ (E\ (F\ G)))$$



SCHEME

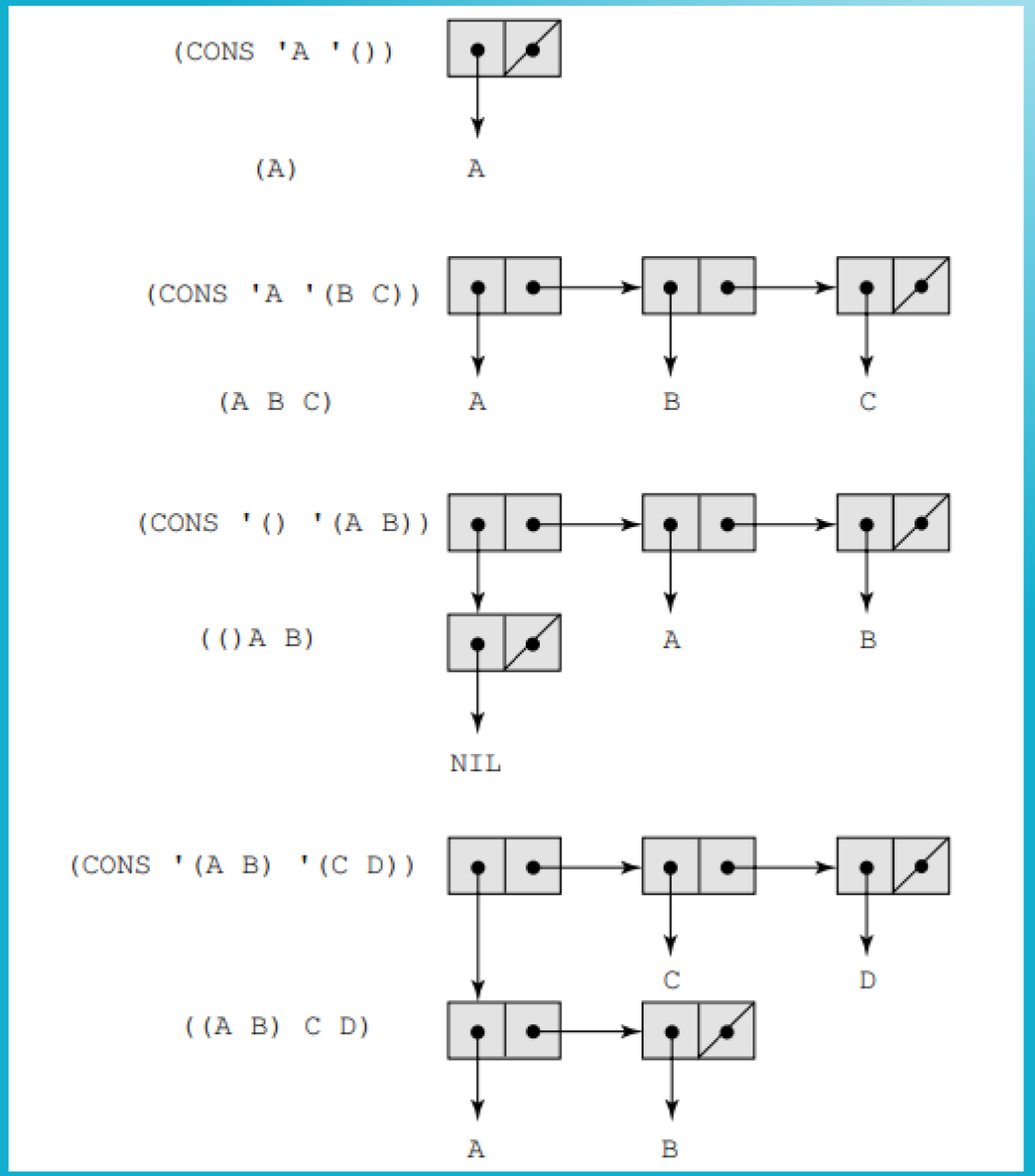
- **Dialeto de Lisp, foi desenvolvida no MIT em meados de 1970 (Sussman e Steele, 1975)**

- **Caracterizada por ser pequena, ter uso exclusivo de escopo estático e tratamento de funções como entidades de primeira classe. Como entidades de primeira classe, as funções Scheme podem ser os valores de expressões, elementos de listas, passadas como parâmetros e retornadas de funções. As primeiras versões de Lisp não forneciam todas essas capacidades.**

- **É adequada para aplicações educacionais, como cursos sobre programação funcional, e para introduções gerais à programação.**

Essa forma é típica de funções simples de processamento de listas. Em tais funções, os dados da lista são processados um elemento de cada vez.

CONS constrói uma nova lista a partir de duas partes de uma lista



COMMON LISP

- **(Steele, 1990)**

- **É o resultado de um esforço para combinar os recursos de diversos dialetos anteriores de Lisp, incluindo Scheme, em uma única linguagem.**

- **Já que consiste em uma espécie de união de linguagens, ela é muito extensa e complexa (similar a C++ e C#).**
  Sua base é Lisp original, portanto, sua sintaxe, funções primitivas e natureza fundamental vêm dessa linguagem.

Common Lisp inclui várias construções imperativas e alguns tipos mutáveis. Abaixo temos um exemplo de função factorial:

```
(DEFUN factorial (x)
    (IF (<= n 1)
        1
        (* n factorial (– n 1)))
))
```

ML

- (Milner et al., 1997)

- É uma linguagem de programação funcional com escopo estático, como Scheme. Entretanto, ela difere de Lisp e de seus dialetos, incluindo Scheme, de diversas maneiras significativas.

- ML tem declarações de tipo para os parâmetros de função e para os tipos de retorno de funções, embora muitas vezes não sejam usadas, devido à sua inferência de tipos. O tipo de cada variável e expressão pode ser determinado estaticamente. ML, como outras linguagens de programação funcional, não tem variáveis, no sentido das linguagens imperativas.

Como ML não permite funções sobrecarregadas, não poderia coexistir baseada em int. O fato de o valor funcional ser tipado como real é suficiente para inferir que o parâmetro também é do tipo real. Cada uma das definições a seguir também é válida:

```
fun square(x : real) = x * x;
fun square(x) = (x : real) * x;
fun square(x) = x * (x : real);
```

HASKELL

- Thompson, 1999.

- É parecida com a ML por usar uma sintaxe semelhante, ter escopo estático, ser fortemente tipada e utilizar o mesmo método de inferência.

- É uma linguagem que possui foco no alcance de soluções para problemas matemáticos, clareza, e de fácil manutenção nos códigos, e possui uma variedade de aplicações e apesar de simples é muito poderosa.

Exemplo de algoritmo quicksort (ordenação rápida) em HASKELL:

```
sort [] = []
sort (h:t) = sort [b | b <- t, b <- h]
                ++ [h] ++
                sort [b | b <- t, b > h]
```

F#

- Linguagem de programação funcional .NET cujo núcleo é baseado em OCaml, que é uma descendente de ML e Haskell.

- Possui IDE completo, com uma ampla biblioteca de utilitários que suportam programação imperativa, orientada a objetos e funcional, além de interoperabilidade com uma coleção de linguagens não funcionais (todas as linguagens .NET).

F# é uma linguagem .NET de primeira classe. Isso significa que os seus programas podem interagir de todas as maneiras com outras linguagens .NET

Por exemplo, classes F# podem ser usadas e estendidas (por meio de especialização) por programas em outras linguagens e vice-versa. Além disso, os programas F# têm acesso a todas as APIs do Framework .NET

- ***F# contém uma variedade de tipos de dados. Ex: tuplas***
- ***F# tem vetores mutáveis e imutáveis.***

Iteradores também podem ser usados para criar sequências, como no exemplo a seguir:

```
let cubes = seq {for i in 1..5 -> (i, i * i * i)};;
```

Isso gera a seguinte lista de tuplas:

```
seq [(1, 1); (2, 8); (3, 27); (4, 64); (5, 125)]
```

Esse uso de iteradores para gerar coleções é uma forma de compreensão de lista.

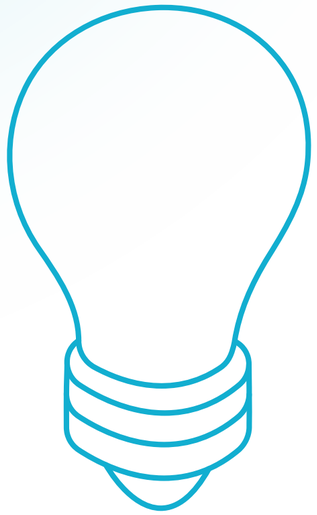
Suporte para programação funcional em linguagens basicamente imperativas

Uma indicação do crescente interesse e uso de programação funcional é o suporte parcial para ela, em linguagens imperativas.

Linguagens de programação imperativas normalmente fornecem apenas suporte limitado para programação funcional. Esse suporte resultou no pouco uso dessas linguagens para programação funcional. A restrição mais importante, relacionada à programação funcional, das primeiras linguagens imperativas era a ausência de suporte para funções de ordem superior.

Um exemplo disso, é quem em funções anônimas, que são como expressões lambda, agora fazem parte de JavaScript, Python, Ruby, Java e C#.

Uma comparação entre linguagens funcionais e imperativas



- Linguagens funcionais podem ter uma estrutura sintática muito simples
- A semântica das linguagens funcionais é mais simples que a das imperativas
- Programação funcional resulta um aumento de uma ordem de magnitude na produtividade
- Os programas funcionais têm apenas 10% do tamanho de seus correspondentes imperativos

Linguagens funcionais têm uma possível vantagem na legibilidade

```
int sum_cubes(int n){
    int sum = 0;
    for(int index = 1; index <= n;
index++)
    sum += index * index * index;
    return sum;
}
```

Em Haskell, a função poderia ser:

```
sumCubes n = sum (map (^3) [1..n])
```

Obrigado!