LPs: evolução, paradigmas e conceitos básicos

Prof. Rodrigo Rocha (rodrigo@dcc.ufba.br)

MATA56 - Paradigmas de Linguagens de Programação,
UFBA, 2016.1

Prof. Rodrigo Rocha (rodrigo@dcc.ufba.br)

Introdução

Links interessantes

- Comparação de sintaxe: `http://merd.sourceforge.net/pixel/language-study/syntax-across-languages/`
- Programa escrito em várias linguagens: `http://www.99-bottles-of-beer.net/`
- Popularidade de linguagens: `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

Considerações sobre LPs

Domínios

- Aplicações científicas
  - números ponto flutuante, operações vetorizadas
- Aplicações de negócio
  - relatórios, números decimais
- Inteligência artificial
  - manipulação de símbolos
- Programação de sistemas
  - eficiência
- Web
  - marcação, script...

Critérios de avaliação (Sebesta)

- Legibilidade
- Escritabilidade
- Confiabilidade
- Custo

Características de LPs (Tucker)

- Simplicidade e legibilidade
- Clareza sobre amarração (binding)
- Confiabilidade
- Suporte
- Abstração
- Ortogonalidade
- Implementação eficiente

Implementação de LPs

- Compilada
- Interpretada
- Híbrida
  - JIT (just-in-time compilation)

Sintaxe

- Descreve a "aparência" (forma) dos programas
- Um programa pode estar sintaticamente correto mas, mesmo assim, não fazer sentido (ex.: `"oi" * 5` em Java)
- A sintaxe de uma LP pode ser descrita por expressões regulares e uma gramática BNF. Exemplo (LISP, simplificado):

```
expression ::= atom | list
atom       ::= number | symbol
number     ::= [+-]?['0'-'9']+
symbol     ::= ['A'-'Z''a'-'z'].*
list       ::= '(' expression* ')'
```

Semântica

- Se refere ao significado do programa
- `"oi" * 5`: `"oi"` é string, `5` é inteiro, `*` é um operador aritmético binário
- Sistema de tipos
  - tipagem forte vs. fraca
  - tipagem estática vs. dinâmica

Nomes, escopos e amarração

- Qual valor será impresso? (Javascript)

```
var x = 1;
function teste() {
    var x = 2;
    console.log(x);
}
teste();
```

Gerenciamento de memória

- alocação estática, baseada em pilha, baseada em heap
- baseada em heap
  - controle explícito
  - contagem de referências
  - garbage collection

Evolução das LPs

História

- 1949-55
  - computadores a válvula
  - linguagem de máquina: `803C0800`
  - 1949, primeiro assembler (montador): `cmp  byte [eax + ecx], 0`
  - compiladores de expressões (ex.: `b*b - 4*a*c`)

Linguagem de montagem

```
; Comprimento de uma string terminada em 0
; in:   eax = endereço da string
; out:  ecx = comprimento da string
B9FFFFFFFF  mov  ecx, -1

            .loop:
41            inc  ecx
803C0800      cmp  byte [eax + ecx], 0
75F9          jne  .loop

            .done:
C3              ret
```

Fonte:
https://en.wikipedia.org/wiki/Assembly_language

História

- 1956-60
  - transistores
  - compiladores, interpretadores
  - FORTRAN, LISP, ALGOL, COBOL

História

- 1961-65
  - famílias de arquiteturas compatíveis
  - ALGOL, COBOL, APL

História

- 1966-70
  - circuitos integrados
  - compiladores com otimização
  - BASIC, PL/I, SIMULA

História

- 1971-75
  - microcomputadores
  - programação estruturada, engenharia de software
  - Pascal, C, Scheme, Prolog

História

(Ver imagem)

Paradigmas

Introdução

- Linguagem de programação
  - ... é uma linguagem formal construída para comunicar instruções a uma máquina.
    - não pode ser ambígua!
  - ... pode ser usada para criar programas para controlar máquinas ou para expressar algoritmos
- Paradigma de programação
  - ... estilo de programação, forma de construir a estrutura e os elementos de programas de computador.
  - Linguagens de programação dão suporte a um ou mais paradigmas

Gerações

- 1ª geração: linguagem de máquina
  - 803C0800
- 2ª geração: linguagem de montagem
  - `cmp  byte [eax + ecx], 0`
- 3ª geração
  - imperativas: FORTRAN, COBOL, BASIC, ALGOL, ADA, Pascal, C...
  - lógicas e funcionais: LISP, ML, Prolog
- 4ª/5ª geração
  - terminologia nebulosa
  - alguns consideram SQL e R de 4ª geração
  - alguns consideram Prolog de 5ª geração

Paradigmas

- Dois principais grupos de linguagem:
  - Imperativo
    - Descreve o passo-a-passo da execução de um programa
  - Declarativo
    - Descreve *o que* deve ser feito (e não *como*)

Paradigma imperativo

- Imperativo
  - Sequência de comandos, passo-a-passo
  - Lê e modifica a memória (estado)
  - Descreve como o programa opera
  - Base teórica: máquinas de Turing, arquitetura de Von Neumann
  - Analogia: quadro-negro
  - Linguagens: BASIC, Pascal, C

Paradigma imperativo

- Imperativo
  - Mesmo linguagens imperativas possuem partes declarativas
  - Ex.: expressões aritméticas
    - `2 + 3 * 4`

Paradigma imperativo

- Refinamentos do paradigma imperativo
  - Programação estruturada (evita uso do `GOTO`)
    - `if`, `while`, `for`
  - Programação procedural/procedimental
    - Subrotinas: procedimentos e funções

Programação orientada a objetos

- Programação orientada a objetos
  - Objeto: dados + comportamento
  - Passagem de mensagens
  - Encapsulamento, herança, polimorfismo
  - Pode ser baseada em classes (Java, C++) ou em protótipos (Javascript, Lua)
  - Se combina com outros paradigmas: imperativo, funcional...
  - Primeira linguagem: Smalltalk (design de interfaces)

Paradigmas declarativos

- Paradigma funcional (Scheme, ML, LISP)
- Paradigma baseado em lógica ou em restrições (Prolog, VisiCalc)

Paradigma funcional

- Um programa funcional corresponde à computação de uma função
- Funções não possuem efeitos colaterais (i.e., não alteram o estado)
  - Se eu chamar a mesma função duas vezes, o resultado é o mesmo
- Não há comandos, apenas expressões
- Características funcionais têm sido incorporadas a linguagens de programação imperativas, como Java 8, Python, Javascript, Ruby...
- Base teórica: cálculo-lambda
- Analogia: linha de produção

Paradigma funcional

- Exemplo: o n-ésimo número da sequência de Fibonacci, fib(n), é definido matematicamente como:
  - 0, se n = 0
  - 1, se n = 1
  - fib(n-1) + fib(n-2), caso contrário

Paradigma funcional

- Implementação em Elixir (funcional)

```
defmodule Fibonacci do
   def fib(0), do: 0
   def fib(1), do: 1
   def fib(n), do: fib(n-1) + fib(n-2)
end
```

Paradigma funcional

- Funções transformam entradas em saídas
- Parece uma linha de produção
- Exemplo (bash): `ls -l | cut -d' ' -f3 | sort | uniq -c`
- Filosofia Unix: programas fazem apenas uma coisa, mas fazem bem feito

Paradigma funcional

- Funções são elementos de primeira classe
  - podem ser passadas como parâmetro ou retornadas de outras funções
- Função de alta ordem: função que recebe ou retorna uma função
- Ex.: `map`
  - Ruby:
    - `['a', 'b', 'c'].map(&:upcase)`
  - Javascript (com underscore.js)
    - `_.map(['a', 'b'], function(x) { return x.toUpperCase(); } );`

Paradigma lógico

- Um programa consiste de fatos e regras
- Regras permitem derivar novos fatos
- Usuário faz uma consulta para determinar se um determinado fato pode ser derivado a partir dos fatos e regras existentes
- Base teórica: lógica de predicados
- Ex.: Prolog

Paradigma lógico

- Programa:

```
mulher(maria).
homem(joao).
humano(X) :- homem(X).
humano(X) :- mulher(X).
```

- Consulta: `humano(joao).`

Outros paradigmas

- Baseado em eventos (Visual Basic)
- Baseado em autômatos / máquinas de estado finitas
- Programação concorrente / paralela / distribuída (Erlang)
- Programação orientada a aspectos (AspectJ)
- Programação literária (Sweave, knitr)
- Ver mais: `https://en.wikipedia.org/wiki/Programming_paradigm`

Exemplos de código

Scheme (estilo LISP)

```
(define bottles
  (lambda (n)
    (cond ((= n 0) (display "No more bottles"))
          ((= n 1) (display "One bottle"))
          (else (display n) (display " bottles")))
    (display " of beer")))

(define beer
  (lambda (n)
    (if (> n 0)
        (begin
          (bottles n) (display " on the wall") (newli
          (bottles n) (newline)
          (display "Take one down, pass it around") (
          (bottles (- n 1)) (display " on the wall")
          (newline)
          (beer (- n 1))))))
```

Haskell

```
bottles 0 = "no more bottles"
bottles 1 = "1 bottle"
bottles n = show n ++ " bottles"

verse 0   = "No more bottles of beer on the wall, no
          ++ "Go to the store and buy some more, 99 bo

verse n   = bottles n ++ " of beer on the wall, " ++
          ++ "Take one down and pass it around, " ++ b
                                                ++ "

main      = mapM (putStrLn . verse) [99,98..0]
```

Prolog

```
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of
    write('Take one down, and pass it around,'), nl,
    write('Now they are all gone.'), nl,!.
bottles(X) :-
    write(X), write(' bottles of beer on the wall,'),
    write(X), write(' bottles of beer,'), nl,
    write('Take one down and pass it around,'), nl,
    NX is X - 1,
    write(NX), write(' bottles of beer on the wall.')
    bottles(NX).
```

Whitespace

Ver `http://www.99-bottles-of-beer.net/language-whitespace-154.html`

Referências

- TUCKER, NOONAN. Programming Languages Principles and Paradigms. Cap. 1.
- SCOTT. Programming Language Pragmatics. Cap. 1.
- SEBESTA. Concepts of Programming Languages. Caps. 1 e 2.
- Slides do prof. Manoel Mendonça (UFBA). Parte 1.