Programação Lógica

Profª Andréa Schwertner Charão
DELC/CT/UFSM

Sumário

- Introdução
  - Conceitos básicos
  - Características
  - Origens
  - Vantagens e desvantagens
  - Aplicações
- Linguagem Prolog

Programação lógica

- Paradigma baseado no Lógica Matemática e no Cálculo de Predicados
- Programas são **declarativos**:
  - especificam **resultados** desejados em vez de procedimentos para produzi-los
- Possui semelhanças com o paradigma funcional

Programação lógica

- Separação de lógica e controle:
  - **Lógica**: definição do que deve ser solucionado
  - **Controle**: como a solução pode ser obtida
- Programador escreve definições que permitam a **dedução** da solução

Programação lógica

- Principais elementos
  - **Proposições** (axiomas lógicos) que formam uma base de **fatos** conhecidos
  - **Regras** que definem como deduzir novas proposições a partir da base
  - **Consultas** à base (execução do programa)

Programação lógica: ilustração

- Proposição (fato)
  - *Zé Carioca é um papagaio.*
- Regra de inferência
  - *Todo papagaio é uma ave.*
- Consulta
  - *Zé Carioca é uma ave?*
- Solução/resposta/resultado
  - *Sim*

Programação lógica: ilustração

- Proposições (fatos)
  - *João é pai de José.*
  - *João é pai de Maria.*
- Consulta
  - *João é pai de quem?*
- Solução/resposta/resultado
  - *José*
  - *Maria*

Programação lógica: ilustração

- Proposição (fato)
  - *O fatorial de 0 é 1.*
- Regra
  - *O fatorial de um número N (N > 0) é igual a N*fatorial(N-1).*
- Consultas
  - *O fatorial de 2 é 200?*
    - Resposta: *Não*
  - *Quanto é o fatorial de 3?*
    - Resposta: *6*

Bases da programação lógica

- Lógica matemática
  - Álgebra de Boole
  - Descrição de proposições e verificação quanto à validade das mesmas

- Cálculo de predicados
  - Lógica simbólica
  - Proposições envolvem símbolos e operadores lógicos
  - Cláusulas de Horn: proposições com forma restrita, 2 partes

Significado	Representação
p e q	$p \wedge q$
p ou q	$p \vee q$
p implica q	$p \rightarrow q$
p equivale a q	$p \leftrightarrow q$

Origens da programação lógica

- Lógica matemática, lógica simbólica e cálculo de predicados
- Linguagem Prolog
  - Início da década de 70: colaboração entre universidades
  - Colmerauer e Roussel (Marseille): processamento de linguagem natural
  - Kowalski (Edinburgh): prova automatizada de teoremas
  - Desenvolvimento independente a partir de meados de 70

Aplicações

- Inteligência artificial
  - Sistemas especialistas: emulação da habilidade humana em algum domínio do conhecimento (ex. medicina)
  - Sub-sistemas “inteligentes”, por exemplo:
    - jogos: capacidade de derivar novos cenários, comportamentos, etc.
    - bancos de dados: capacidade de deduzir informações a partir dos dados no banco
- Processamento de linguagem natural
  - Interfaces humano-computador
- Educação
  - Ensino de lógica, contribuindo para pensamento e expressão mais claros

Vantagens e desvantagens

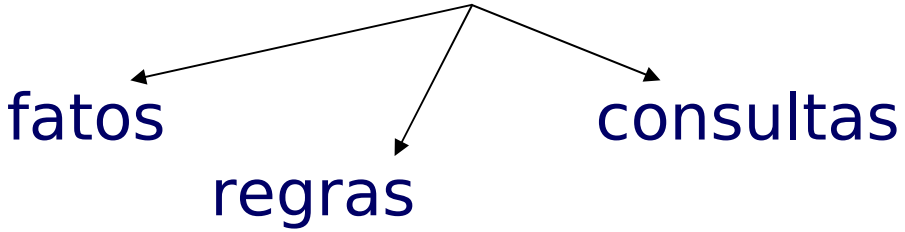
- Vantagens
  - Separação entre lógica de programa e controle
  - Programas fáceis de entender e manter
- Desvantagens
  - Eficiência
  - Baixa expressividade para certas aplicações

Programação Lógica em Linguagem Prolog

Profª Andréa Schwertner Charão
DELC/CT/UFSM

Linguagem Prolog

- Programas compostos por **cláusulas**



- Diferentes dialetos
- Implementações da linguagem
  - Interpretadores e compiladores
  - Exemplos: SWI Prolog, Turbo Prolog, LPA Prolog, GNU Prolog, etc.

Programas em Prolog

- Programas consistem em **cláusulas** terminadas por pontos

```
papagaio("Ze Carioca").
ave(X) :- papagaio(X).
?- ave("Ze Carioca").
```

- Cláusulas podem ser de 3 tipos
  - **fatos**: declaram algo que é sempre verdadeiro
  - **regras**: declaram algo que é verdadeiro em uma dada condição
  - **consultas**: descobrem se uma determinada meta é verdadeira

Programas em Prolog

- "*Zé Carioca é um papagaio*"

```
papagaio("Ze Carioca").
```

fato

- "*Todo papagaio é uma ave*" (ou "*se X é um papagaio, então X é uma ave*")

```
ave(X) :- papagaio(X).
```

regra

- Zé Carioca é uma ave?

```
?- ave("Ze Carioca").
yes
```

consulta

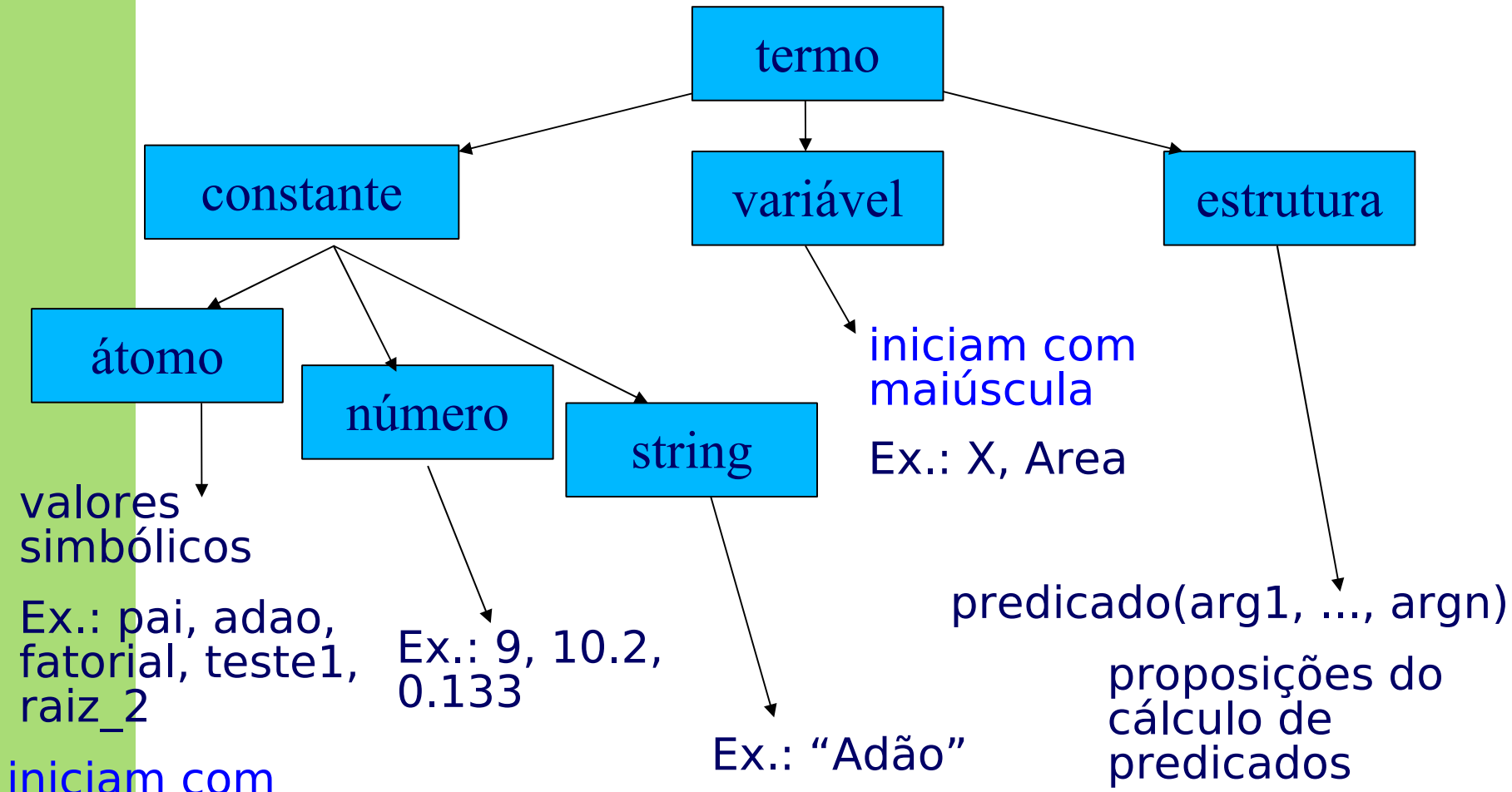
resultado

Programas em Prolog

- Ciclo de programação Prolog
  - Fornecer base de **fatos** e **regras** ao interpretador (carga do programa)
  - Solicitar a verificação de uma proposição (**consulta**)

- Programa é um “mundo fechado”
  - Um fato é considerado falso quando
    - Não está presente na base de fatos do programa
    - Não é dedutível a partir da base de fatos usando a base de regras

Instruções em Prolog



Variáveis em Prolog

- "Parentes distantes" das variáveis de linguagens imperativas
- Variáveis não são declaradas e não são vinculadas a tipos
- Vinculação de valor (instanciação) ocorre no processo de resolução
- Instanciações servem para verificar se alguma meta é verdadeira
- "*Todo papagaio é uma ave*" (ou "*se X é um papagaio, então X é uma ave*")

```
ave(X) :- papagaio(X).
```

Declaração de fatos

- A cláusula seguinte é um exemplo de **fato**:

```
pai(bobbypai, bobbyfilho).
```

- Neste exemplo:
  - `pai(...)` é uma **estrutura**
  - `bobbyfilho` é o primeiro argumento
  - `bobbypai` é o segundo argumento
  - argumentos são separados por vírgulas
  - `bobbyfilho` e `bobbypai` são **constantes** (átomos)

Declaração de fatos

- Formas gerais de declaração de fatos:

  1) propriedade(obj_const).

  Ex.: `cachorro(bobbypai).`

  `gordo(bobbypai).`

predicado

Declaração de fatos

- Formas gerais de declaração de fatos:

  1) propriedade(obj_const).

  Ex.: `cachorro(bobbypai).`

  `gordo(bobbypai).`

predicado

  2) relacao(obj_const1, ..., obj_constn);

  Ex.: `pai(bobbypai, bobbyfilho).`

Declaração de fatos

- Formas gerais de declaração de fatos:

  3) fatouniversal(const1, ...,var1).

  Ex.: ancestral("Adão", X).

predicado

"Para todo X, Adão é ancestral de X"

- As formas 1 e 2 usam **constantes**.
- A forma 3 usa **variáveis** e constantes.

Declaração de fatos

- Observações:
  - Semântica é arbitrária (relação pode significar o que o programador quiser)
  - Deve-se seguir convenções consistentemente
- Convenções quanto à:
  - Aridade (número de argumentos)

```
pai(joao,maria,jose).
```
≠
```
pai(joao,maria).
```

  - Ordem

```
pai(jose,paulo).
```
≠
```
pai(paulo,jose).
```

  - Nome

```
pai(joao, maria).
```
≠
```
pAi(joao,maria).
```

Consultas

- Consultas devem seguir as mesmas convenções da base de fatos
- Formas de consultas (metas):
  1) validação de uma proposição
     - ?- predicado(arg1, ..., argn).
     - Resposta/resultado: `yes` ou `no`

```
Ex.:
cachorro(bobbypai).
cachorro(bobbyfilho).
?- cachorro(bobbypai).
Yes
?- cachorro(pluto).
No
```

Consultas

- Formas de consultas (metas):
  2) verificação de existência
  - ?- predicado(..., var1, ...).
  - Resposta/resultado: constante(s)

```
Ex.:
cachorro(bobbypai).
cachorro(bobbyfilho).
gordo(bobbypai).
?- gordo(X).
X = bobbypai
?- cachorro(X).
X = bobbypai ;
X = bobbyfilho
```

Ponto-e-vírgula: verifica se existe outra resposta

Consultas com operadores

- Operadores relacionais (aritméticos)

>	maior
<	menor
>=	maior ou igual
=<	menor ou igual
=:=	igual
=\=	não igual

- Exemplos

```
idade(pedro, 35).
idade(ana, 30).
idade(paulo, 27).
?- idade(N, X), X =< 30.
?- idade(N, X), X > 25,
                X < 30.

?- 1 + 2 =\= 2 + 1.
No
?- 1 + 2 =:= 2 + 1.
Yes
```

Declaração de regras

- A cláusula que segue é um exemplo de **regra**:

```
ave(X) :- papagaio(X).
```

- Neste exemplo
  - X é uma variável
  - ave e papagaio são predicados (functors)
- Regras são cláusulas com condições (cláusula de Horn com lado esquerdo e lado direito)
- Forma geral:

```
consequente :- antecedente
```

leia-se: `então :- se`

Declaração de regras

- Regras compostas: conetivo “e” (,)
- Ex.: *“Se X é pai de Y e Y é pai de Z, então X é avô de Z”*

  Em Prolog:

```
avo(X,Z) :- pai(X,Y) , pai(Y,Z).
```

- Regras compostas: conetivo “ou” (;)
- Ex.: *“Se X é um papagaio ou uma coruja, então X é uma ave”*

  Em Prolog:

```
ave(X) :- papagaio(X) ; coruja(X).
```

Processo de inferência

- A resolução de um problema consiste em satisfazer uma consulta

- Para ilustrar o processo de inferência, consideremos a seguinte base

```
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

Exemplo de inferência

- Meta: "Donald é um pato?"

```
?- pato(donald).
```

- Busca de predicado pato
- Fato encontrado
- Resposta:

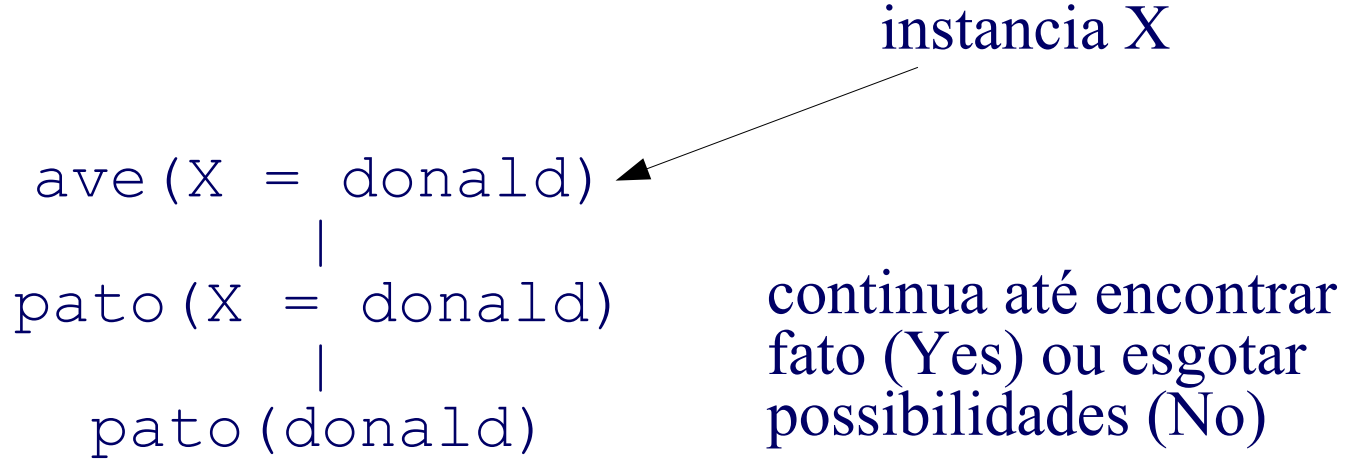
```
?- pato(donald).
Yes
```

Exemplo de inferência

- Meta: "Donald é uma ave?"

```
?- ave(donald).
```

- Busca de predicado ave
- Fato não encontrado, regra encontrada
- Unificação (matching) com instanciação de variáveis



Aritmética simples

- Operador “is”

```
soma(A,B,C) :- A is B + C.
?- soma(A, 1, 4).
5
```

- **Atenção!!!** este operador não deve ser usado como uma atribuição:
- Ex.: `soma(A,B,C) :- A is A + B + C.`
- Isso é ERRADO, pois as variáveis só são instanciadas para satisfazer metas. Se A já estiver instanciado, A não pode estar do lado esquerdo!

Aritmética simples

- **Atenção!!!** Não se pode usar um predicado no lugar da expressão aritmética:
- Ex.: A is soma(A,3,4).
- Isso é ERRADO!O predicado não vai retornar um valor!

Aritmética simples

- Uso de operadores aritméticos:

```
largura(sala,4).
comprimento(sala,5).
largura(quarto,3).
comprimento(quarto,2).
area(X,Y) :- largura(X,L),
  comprimento(X,C), Y is L * C.
?- area(sala,X).
   X = 20
?- area(banheiro,X).
   No
```

Recursividade

- Deseja-se implementar:

fatorial(N) = N * fatorial(N-1), para N > 0

```
fatorial(0,1).
fatorial(N,X) :- N1 is N - 1,
                 fatorial(N1,X1),
                 X is N * X1.

?- fatorial(3,X).
X=6
```

Entrada e saída

- Predicados: **write**, **read**, **nl**, etc.

```
olafulano :-
        write('Digite seu nome:'),
        read(Nome),
        write('Ola, '),
        write(Nome),
        nl.
```

Listas em Prolog

- Seqüência finita de elementos

```
[a, b, c, d].
solucao( [casa(1, azul),
          casa(2, verde)] ).
pessoas([jose, 20, brasil, santa_maria],
        [maria, 24, uruguai, montevideo]).
```

- Exemplo com unificação

```
?- [1, 2, 3, 4] = [_, A, _, _].
   A = 2
?- [1, 2, 3, 4] = [X | Y].
   X = 1
   Y = [1, 2, 3, 4]
?- [1] = [X | Y].
   X = 1
   Y = [].
?- [] = [X | Y].
   No
```

Listas em Prolog

- Predicados

```
?- member(a, [a,b,c]).
   Yes
?- member(1, []).
   No
```

- Regras com listas

```
primeiro(Lista, A) :- Lista = [A,_].
ultimo([X],X).
ultimo(Lista, X) :- Lista = [_|U],
                    ultimo(U, X).
```