Programação lógica: operadores, repetições e listas

Profs. Diogo S. Martins e Emilio Francesquini

{santana.martins,e.francesquini}@ufabc.edu.br

MCTA016 - Paradigmas de Programação (Prática)

14 de agosto de 2018

UFABC

CMCC
Centro de Matemática, Computação e Cognição

CC BY SA

Crédito de parte das imagens, a menos se especificado: Wikipedia

Objetivos

- Compreender o mecanismo de operadores na linguagem Prolog
- Aprender a desenvolver programas que envolvam repetição
- Analisar o processamento de listas em Prolog

Instalação SWI Prolog

SWI Prolog

- Implementação de Prolog com múltiplas extensões (inclusive multiparadigma, web semântica, etc.):
- Instalação, documentação, etc.: `http://www.swi-prolog.org/`
- Mantida desde 1987, originou-se na Universidade de Amsterdam[1]
- Instalação no Ubuntu:

```
$ sudo apt-add-repository ppa:swi-prolog/stable
$ sudo apt-get update
$ sudo apt-get install swi-prolog
```

- Teste:

```
$ echo "hello :- write('hello world'), nl." > hello.pl
$ swipl -l hello.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- hello.
hello world
true.
?- halt.
$
```

[1] SWI refere-se a *Sociaal-Wetenschappelijke Informatica*, nome de um grupo de pesquisa da UvA

Operadores

Operadores

- Em Prolog, operadores são um açúcar sintático
- Objetivo: tornar as cláusulas menos verbosas
- Sem sintaxe especial, notação seria similar a S-expressions:

```
+(2, /(*(5, 7), 2), 5)
```

Operador

Termo binário ou unário (aridade 1 ou 2) que pode ser sintaticamente usado na notação infixa ou sufixa, sem parênteses.

- É possível definir seus próprios operadores (não veremos)
- Operações aritméticas são efetuadas via operadores pré-definidos

Operações aritméticas

Expressão aritmética

Expressão construída via operadores aritméticos.

- REPL não aceita resultados numéricos, é preciso encapsular o valor de uma expressão em algo que retorne um valor-verdade
- Solução: amarração de resultados de operações aritméticas envolve operador especial

Operador `is`

Operador binário com as seguintes propriedades:

1. O operando da direita é uma expressão aritmética
2. A operação consiste em unificar o operando da direita com o operando da esquerda

- Unificação pode ser bem sucedida ou não, a depender do operando da direita

Operadores

- Operadores são funções, e não predicados
- **Predicados** têm um **valor lógico** como resultado
- **Funções** podem ter **valores numéricos** como resultado

Operações aritméticas

Exemplo

```
?- X is 10.5 + 3.
X = 13.5.

?- X is 10, Z is X+1.
X = 10,
Z = 11.

?- X is sqrt(36).
X = 6.0.

?- X is 7, X is 6+1.
X = 7.

?- X is 7, X is 6+2.
false.

?- A is 10, B is 11.
A = 10,
B = 11.
```

```
?- C is A+B.
ERROR: is/2: Arguments are not
sufficiently instantiated

?- X is "a".
X = 97.

?- X is "ab".
ERROR: is/2: Type error: `[]'
expected, found `"ab"' (a
string) ("x" must hold
one character)

?- 2 mod 2.
ERROR: toplevel: Undefined
procedure: (mod)/2 (DWIM
could not correct goal)

?- X is 2 mod 2.
X = 0.
```

Operações aritméticas

Exemplo

- **Exemplo**: crie um programa que incremente um número `N` com passo `P`
  - **Tentativa 1**:

```
1 increase(N,P) :- N is N + P.
```

```
?- increase(10,3).
false.
```

  - **Tentativa 2:**

```
1 increase2(N,P) :- R is N + P.
```

```
?- increase2(10,3).
true.
```

  - **Tentativa 3:**

```
1 increase3(N,P,R) :- R is N+P.
```

```
?- increase3(10,3,X).
X = 13.
```

Operadores e funções aritméticos

Exemplos

X+Y	The sum of X and Y
X-Y	The difference of X and Y
X*Y	The product of X and Y
X/Y	The quotient of X and Y
X//Y	The 'integer quotient' of X and Y (the result is truncated to the nearest integer between it and zero)
X mod Y	The remainder when X is divided by Y
X^Y	The value of X to the power of Y
-X	The negative of X
abs(X)	The absolute value of X
sin(X)	The sine of X (for X measured in radians)
cos(X)	The cosine of X (for X measured in radians)
max(X,Y)	The larger of X and Y
round(X)	The value of X rounded to the nearest integer
sqrt(X)	The square root of X

[?]

Operadores relacionais

Igualdade entre expressões aritméticas

- **Exemplos**: Igualdade entre expressões aritméticas

```
?- 6 + 4 =:= 6 * 3 - 8.
true.

?- sqrt(36)+4 =:= 5 * 11 - 45 .
true.
```

- **Exemplo**: construa um predicado que determine se um inteiro é par.
  - **Alternativa 1**:

```
1 even?(N) :- N mod 2 =:= 0 .
```

```
?- even?(4).
true.

?- even?(11).
false.
```

  - **Alternativa 2**:

```
1 even2?(N) :- N mod 2 =\= 1.
```

```
?- even2?(2).
true.

?- even2?(13).
```

Operadores relacionais

Igualdade de termos

Igualdade de átomos

Dois átomos são iguais se podem ser unificados.

```
?- term1 == term2.
false.

?- term1 == term1.
true.
```

Igualdade de amarrações

Duas amarrações são iguais se amarram o mesmo valor (i.e. se os valores unificam).

```
?- X = 'a', Y = 'a', X == Y.
X = Y, Y = a.

?- X = 1, Y is 0 + 1, X == Y.
X = Y, Y = 1.

?- X = 1, Y is 1 + 1, X == Y.
false.
```

Operadores relacionais

Igualdade de termos

Igualdade de termos

Dois termos são iguais se:

1. Possuem o mesmo funtor
2. Possuem a mesma aridade
3. Os argumentos são iguais

```
?- likes(X,prolog) == likes(X,prolog).
true.

?- likes(X,prolog) == likes(Y,prolog).
false.

?- X is 10, pred1(X) == pred1(10).
X = 10.

?- 6 + 4 == 3 + 7.
false.
```

```
?- +(6,4) == +(3,7).
false.

?- 6 + 4 == 6 + 4.
true.

?- 6 + 4 \== 3 + 7.
true.
```

Operador de unificação

Operador de unificação

Operador que é bem sucedido se é possível unificar dois termos.

```
?- 1 = 1.
true.

?- a = a.
true.

?- pred(10) = pred(10).
true.

?- pred(X) = pred(10).
X = 10.

?- likes(X, prolog) = likes(john, Y).
X = john,
Y = prolog.

?- 6 + 3 = 3 * 3.
false.
```

```
?- 6 + X = 6 + 3.
X = 3.

?- likes(X, prolog) = likes(Y,prolog).
X = Y.

?- likes(X, prolog) = likes(Y, java).
false.

?- Z = prolog,
  likes(X, Z) = likes(Y, prolog).
Z = prolog,
X = Y.

?- 6 + 3 \= 3 * 3.
true.
```

Operadores lógicos

- **Negação**:

```
?- not(X is 0).
false.

?- assert(dog(fido)).
true.

?- dog(fido).
true.

?- not(dog(fido)).
false.
```

- **Conjunção**:

```
?- X = 0, X is 0.
X = 0.

?- X = 0, X is 1-1.
X = 0.

?- X = 0, not(X is 1).
X = 0.
```

Operadores lógicos

- **Disjunção**:

```
?- 6 < 3 ; 7 is 5 + 2.
true.

?- 6 * 6 == 36 ; 10 = 8 + 3 .
false.
```

Precedência de operadores

1200	xfx	-->, :-
1200	fx	:-, ?-
1150	fx	**dynamic**, **discontiguous**, **initialization**, **meta_predicate**, **module_transparent**, **multifile**, **public**, **thread_local**, **thread_initialization**, **volatile**
1100	xfy	;, |
1050	xfy	->, *->
1000	xfy	,
990	xfx	:=
900	fy	\+
700	xfx	<, =, =.., =@=, \=@=, =:=, =<, ==, =\=, >, >=, @<, @=<, @>, @>=, \=, \==, **as**, **is**, >:<, :<
600	xfy	:
500	yfx	+, -, /\, \/, **xor**
500	fx	?
400	yfx	*, /, //, **div**, **rdiv**, <<, >>, **mod**, **rem**
200	xfx	**
200	xfy	^
200	fy	+, -, \
100	yfx	.
1	fx	$

http://www.swi-prolog.org/pldoc/man?predicate=op/3

Repetição

Repetições

Exemplos

- Repetições em Prolog são construídas via recursão
  - Análogo ao que vimos em Lisp
- **Exemplo**: construa um programa que imprima os inteiros no intervalo $(0, N]$

```
1 count(0).
2 count(N) :- N > 0, write(N), nl, M is N-1, count(M).
```

```
?- count(6).
6
5
4
3
2
1
true
```

Repetições

Exemplos

- **Exemplo**: construa um programa que imprima os inteiros no intervalo $(I, N]$

```
1 count(I, I).
2 count(I, N) :- N > I, write(N), nl, M is N-1, count(I, M).
```

```
?- count(2,11).
11
10
9
8
7
6
5
4
3
true
```

- **Exercício**: adapte o programa para imprimir no intervalo $[I, N]$.

Repetições

Exemplos

- **Exemplo**: construa um programa que resulte na soma dos inteiros no intervalo $[0, N]$

```
1 sum_n(0,0).
2 sum_n(N, S) :- N > 0, N1 is N-1, sum_n(N1, S1), S is N + S1.
```

```
?- sum_n(10,S).
S = 55 .
```

Repetições

Exemplos

- **Exemplo**: construa um programa que calcule o fatorial de N.
  - **Tentativa 1**:

```
1 fatorial(0,1).
2 fatorial(N,F) :- N1 is N - 1, fatorial(N1, F1),
3   F is F1 * N .
```

```
?- fatorial(4,F).
F = 24 ;
ERROR: Out of local stack
```

  - **Tentativa 2**:

```
1 fatorial2(0,1).
2 fatorial2(N,F) :- N > 0, N1 is N - 1, fatorial2(N1, F1),
3   F is F1 * N .
```

```
?- fatorial2(4,F).
F = 24 ;
false.
```

Listas

Listas

Lista

Estrutura de dados com as seguintes propriedades:

1. `[]` é uma lista vazia
2. `[A|B]` é uma lista não vazia em que A é cabeça e B é a cauda da lista
3. A cauda de uma lista é sempre uma lista
4. A cabeça de uma lista pode ser qualquer termo

- Exemplos:

```
?- Z = [john, mary, 10, robert, 20, jane, X, bill].
Z = [john, mary, 10, robert, 20, jane, X, bill].

?- Z = [[john,28], [mary,56,teacher], robert,
  parent(victoria,albert), [a,b,c,[d,e,f],g]] .
Z = [[john, 28], [mary, 56, teacher], robert,
  parent(victoria, albert), [a, b, c, [...|...]|...]].

?- X = alpha, Y = 27, Z = [alpha, beta],
  write('List is: '), write([X, Y, Z]), nl.
List is: [alpha,27,[alpha,beta]]
X = alpha,
Y = 27,
Z = [alpha, beta].
```

Construção de listas

Operador cons

- Listas são construídas via o operador cons (|), que é análogo ao utilizado em Lisp

```
?- X = [], Y = [a|X], Z = [b|Y], W = [c|Z].
X = [],
Y = [a],
Z = [b, a],
W = [c, b, a].

?- write([1|[2,3,4,5]]).
[1,2,3,4,5]
true.

?- write([[6,7]|[2,3,4,5]]).
[[6,7],2,3,4,5]
true.
```

Decomposição de listas

Operador cons

- A unificação do operador cons com uma lista pode ser usada para decompor a lista

```
?- [H|T] = [1,2,3,4].
H = 1,
T = [2, 3, 4].

?- [a,b,c,d] = [X|Y].
X = a,
Y = [b, c, d].
```

- **Exemplo**: construa um programa que imprima elemento a elemento de uma lista.

```
1 write_all([]).
2 write_all([H|T]) :- write(H), nl, write_all(T) .
```

```
?- write_all([1,2,3,4]).
1
2
3
4
true.
```

Processamento de listas

Exemplos

- **Exemplo**: construa um programa que some todos os valores de uma lista de inteiros.

```
1 list_sum([],0).
2 list_sum([H|T], S) :- list_sum(T,S1), S is S1 + H .
```

```
?- list_sum([0,0,1,2,3,4,5,6,7,8,9,10], S).
S = 55.
```

- **Exemplo**: construa um programa que gere uma lista dos quadrados dos valores de outra lista.

```
1 list_square([], []).
2 list_square([H|T], L) :- Sq is H * H, list_square(T, L1),
3   L = [Sq | L1] .
```

```
?- list_square([1,2,3,4,5,6,6,7], R).
R = [1, 4, 9, 16, 25, 36, 36, 49].

?- list_square([], R).
R = [].
```

Processamento de listas

Exemplos

- **Exemplo**: Construa um procedimento que determine a quantidade de elementos em uma lista.
  - **Tentativa 1**:

```
1 list_length([], 0) .
2 list_length([H|T], L) :- list_length(T, L1), L is L1 + 1 .
```

```
?- consult("length.pro").
Warning: (...)
  Singleton variables: [H]

?- list_length([], L).
L = 0.

?- list_length([1,2,3,4], L).
L = 4.
```

  - **Tentativa 2**:

```
1 list_length2([], 0) .
2 list_length2([_|T], L) :- list_length2(T, L1), L is L1 + 1 .
```

Amarração anônima

Amarração anônima

Amarração sem identificador na qual cada ocorrência corresponde a um endereço diferente de memória.

```
?- write(X), nl, write(Y), nl, X is 10, nl, write(X), nl,
  Y = X, nl, write(Y).
_G11335
_G11337

10

10
X = Y, Y = 10.

?- write(_), nl, write(_), nl, _ is 10, write(_), nl.
_G11335
_G11337
_G11342
true.
```

Exercícios

Nos exercícios a seguir não é permitido usar predicados pré-definidos, a menos se especificado em contrário.

1. Construa um programa que verifique se um inteiro é membro de uma lista de inteiros. Você não pode usar predicados pré-definidos.
2. Adapte o programa anterior para que a lista possa conter termos e números. Se precisar verificar o tipo do elemento da lista, você pode usar um dos predicados pré-definidos listados em: `http://www.swi-prolog.org/pldoc/man?section=typetest`
3. Defina a operação shift(L1, L2) que faça uma rotação à esquerda em L1 e amarre o resultado em L2. Exemplo: L1 = [1,2,3,4] L2 = [2,3,4,1].
4. Defina a operação `reverse` que apresente os elementos de uma lista em ordem reversa.
5. Defina a operação `max` que determine se o maior elemento de uma lista de inteiros.

Materiais complementares

- Caps. 4, 6 e 9: Bramer, Max. “Logic programming with Prolog”. 2nd edition (2013).

Programação lógica: operadores, repetições e listas

Profs. Diogo S. Martins e Emilio Francesquini

{santana.martins,e.francesquini}@ufabc.edu.br

MCTA016 - Paradigmas de Programação (Prática)

14 de agosto de 2018

UFABC

CMCC
Centro de Matemática, Computação e Cognição

CC BY SA

Crédito de parte das imagens, a menos se especificado: Wikipedia