Linguagem Prolog
Conceitos Básicos

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

ADAPTADOS A PARTIR DE SLIDES OBTIDOS NA INTERNET – PARTE 2 DE 3

Operadores

```
module(if,[se/1, entao/2,
senao/2]).
:- op(800,xfx,entao).
:- op(750,fx, se).
:- op(810,xfx,senao).
se X :- X.
X entao Y :- X, Y.
X senao _ :- X,!.
_ senao Z :- Z.
```

```
?- X = 12, se X=10.
   false.

?- X = 12, X=12 entao Y=1.
                    % implicação
   X=12,
   Y=1.

?- X = 10, X==10 senao Y = 2.
                    % ou
   X=10.
```

Operadores

```
module(if,[se/1, entao/2, senao/2]).
:- op(800,xfx,entao).
:- op(750,fx, se).
:- op(810,xfx,senao).
se X :- X.
X entao Y :- X, Y.
X senao _ :- X,!.
_ senao Z :- Z.
```

```
?- X = 12, se X=10 entao Y=1.
false.
?- X = 12, se X=10 entao Y=1 senao Y = 2.
X=12,
Y=2.
?- X = 10, se X=10 entao Y=1 senao Y = 2.
X=10,
Y=1.
```

Operadores

Especificador	Classe	Associatividade
fx	prefix	não associativo
fy	prefix	associativo a direita
xfx	infix	não associativo
xfy	infix	associativo a direita
yfx	infix	associativo a esquerda
xf	postfix	não associativo
yf	postfix	associativo a esquerda

Operadores

Prioridade	Especificador	Operador(es)
1200	xfx	:-
1200	fx	:-
1100	xfy	;
1050	xfy	->
1000	xfy	,
900	fy	\+
700	xfx	=, \=
700	xfx	==, \==, @<, @=<, @>, @>=
700	xfx	is, =:=, =\=, <, =<, >, >=
500	yfx	+, -, /\, \/
400	yfx	*, /, //, rem, mod, <<, >>
200	xfx	**
200	xfy	^
200	fy	-, \

Operadores

Prioridade	Especificador	Operador(es)
1000	xfy	|
900	fy	not
700	xfx	is_string
600	yfx	&
200	fy	+

Unificando com Variável

variável com variável

X = Y

- X unifica com Y e retorna true.

  ?- X = Y.

  X = Y

  - passam a representar a mesma variável.

X \= Y

- retorna sempre false.

  ?- X \= Y.

  false.

Unificando com Variável

Variável em dados estruturados

há unificação se as estruturas forem isomórficas e se houver uma substituição sobre as variáveis que torne as estruturas idênticas.

```
?- X = arco(a,b,10).
   X = arco(a,b,10)
?- livro(autor(gilberto,  joao), Titulo) =
        livro(Autor, titulo('Amor e Paz')).
   Titulo = titulo('Amor e Paz') ,
   Autor = autor(gilberto, joao)
```

Unificando com Variável

Variável com constante

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

X = a , Xs = [b,c,d]

?- [X1,X2,c,d] = [a,b,c,d].

X1 = a , X2 = b

?- [[X|Xs],Y,c,d] = [[1,2,3],a,c,d].

X = 1 , Xs = [2,3] , Y = a

?- [[X,2,3],Y,b,c] = [[1|Xs],a,Z1,Z2].

X = 1 , Y = a , Xs = [2,3] , Z1 = b , Z2 = c

Unificando Constantes

T1 = T2

Unificam se tiverem valores idênticos e forem estruturalmente isomórficos.

?- 10 = 10.

true.

?- arco(a,b,10) = arco(a,b,10).

true.

?- ‘hamilton' = hamilton.

true.

?- "a b" = "a b".

true.

Compara Termos Arbitrários Segundo Ordem Padrão

Com os operadores

==/2 (idênticos) \==/2 @>=/2 @>/2 @</2 @=</2

Tipo	Compara
variáveis	endereços
números	valor numérico
átomos	valor ASCII

variáveis<números<strings<átomos<listas<termos-compostos

- Para termos do mesmo tipo, uma “comparação de tipo” é feita.

Exemplo: qsort

```
separa(B,[X|Xs],[X|Ys],Zs) :- X @=< B,
                              separa(B,Xs,Ys,Zs).
separa(B,[X|Xs],Ys,[X|Zs]) :- X @>  B,
                               separa(B,Xs,Ys,Zs).
separa(_, [],[],[]).

qsort([X|Xs],Rs) :- separa(X,Xs,Ys, Zs),
                    qsort(Ys,Ry),
                    qsort(Zs,Rz),
                    append(Ry,[X|Rz], Rs).
qsort([],[]).
```

Exemplo: qsort

?-qsort([hamilton, "Campeão", 1, campeao(ano(2020), esporte(formula,1),piloto(hamilton)), pi,e, 3.141516,-0.1, 1.0], Ordem).

Ordem = [-0.1, 1.0, 1, 3.141516, "Campeão", e, hamilton, pi, campeao(..., ..., ...)] .

significando:

Ordem = [-0.1, 1.0, 1, 3.141516, "Campeão", e, hamilton, pi, campeao(ano(2020), esporte(formula,1),piloto(hamilton))] .

Base de Conhecimentos

→ livro(autor('Fernando Albuquerque'), titulo('Orientacao a Objetos')).
livro(autor('Pedro Rezende'), titulo('Criptografia em Redes')).
livro(autor('Maristela Holanda'), titulo('Modelos de BD')).
livro(autor('Marcelo Ladeira'), titulo('Mineração de Dados')).
livro(autor('Fernando Albuquerque'), titulo('Redes de Computadores')).
livro(autor('Maria Emilia'), titulo('Genoma Humano')).
livro(autor('Mauricio Ayala'), titulo('Funcoes de Reescrita')).
livro(autor('Andre Drummond'), titulo('Infraestrutura de TI')).
livro(autor('Thiago de Paulo'), titulo('Mineração de Textos')).
livro(autor('Maria de Fatima'), titulo('IA na Educacao')).
livro(autor('Wilson Veneziano'), titulo('Informatica na Educacao')).
livro(autor('Li Weigang'), titulo('Transporte Aereo')).
livro(autor('Genaina Nunes'), titulo('Especificacoes de Requisitos')).
artigo(autor('Marcelo Ladeira'), titulo('UnBBayes: A Java Framework for Reasoning')).
artigo(autor('Marcos Caetano'), titulo('5G the next generation')).
artigo(autor('Rodrigo Bonifacio'), titulo('Adopting DevOps')).

Consulta sobre a Base de Livros

?- livro(X,Y).
X = autor('Fernando Albuquerque'),
Y = titulo('Orientacao a Objetos') ;
X = autor('Pedro Rezende'),
Y = titulo('Criptografia em Redes') ;
X = autor('Maristela Holanda'),
Y = titulo('Modelos de BD') ;
X = autor('Marcelo Ladeira'),
Y = titulo('Mineração de Dados')
?- livro(autor('Fernando Albuquerque'),Y).
Y = titulo('Orientacao a Objetos') ;
Y = titulo('Redes de Computadores');
false.

Há um ponteiro apontando para o fato corrente na base.

Quando uma solicitação é feita, o ponteiro é atualizado para refletir o novo fato que atende a solicitação.

Quando o ponteiro chega ao fim da base, a solicitação corrente falha.

Consulta sobre uma base

Backtracking (retrocesso)

```
?- artigo(autor(X),_).
   X = 'Marcelo Ladeira' ;
   X = 'Marcos Caetano' ;
   X = 'Rodrigo Bonifacio‘.
?-  artigo(autor(X),_), write(X), nl, fail.
   Marcelo Ladeira
   Marcos Caetano
   Rodrigo Bonifacio
   false.
```

Incluindo Cláusulas na BC

asserta(clausula).

Inclui clausula na BC, no início das cláusulas com mesmo nome.

?- dynamic(pai/2).

true.

?- asserta(pai(joao, maria)).

true.

?- listing(pai/2).

:- dynamic(pai/2).

pai(joao, maria).

true.

Incluindo Cláusulas na BC

assertz(cláusula) - inclui no final das cláusulas de mesmo nome.

```
?- assertz( (irmao(A,B) :- pai(P,A), pai(P,B), A \== B) ).
   true.
?- listing(irmao/2).
   :- dynamic irmao/2.
   irmao(A, C) :-
         pai(B, A),
         pai(B, C),
         A\==C.
   true.
```

Observe que a regra precisa estar entre parêntesis por conta da prioridade do operador :-/2 ser maior que 1000.

Univ

Operador “=../2” transforma termo em lista

- Apenas um dos seus operandos pode ser variável
- Exemplos

```
?- struct(hello, X) =.. L.
   L = [struct, hello, X]
?- Term =.. [baz, foo(1)]
   Term = baz(foo(1))
?- Pai = [pai, joao, ze].
   Pai = [pai,joao,ze]
?- P =.. [pai,joao,X], call(P).        % relembre pai(joao,maria)!
   P = pai(joao,maria) , X = maria
```

Unificação

É o processo que torna dois termos idênticos, ainda que para isso se faça substituição de variáveis em termos por outros termos.

- o termo t_1 unifica com o termo t se existe uma substituição θ_1 que torna t_1 idêntico a t (representada por $t \equiv t_1\theta_1$)
  - seja t, t_1 e t_2 termos, t é uma instância comum de t_1 e t_2 se $\exists\ \theta_1$ e θ_2 substituições tais que $t \equiv t_1\theta_1$ e $t \equiv t_2\theta_2$.
  - Exemplo

    t1= concat([1,2,3],[4,5],Rs),

    t2= concat([X|XS],Ys,[X|Zs]),

    t = concat([1,2,3],[4,5],[1|Zs]).

    onde

    θ_1= (Rs/[1|Zs]) e θ_2= (X/1, Xs/[2,3], Ys/[4,5])

Unificação

Um unificador de dois termos t e t_1 é uma substituição θ que torna $t \equiv t_1/\theta$ (t e t_1 ficam idênticos).

Regras da unificação

a) variáveis unificam com variáveis

X = Y.

b) variáveis unificam com termos

X = gosta(joao,ler).

c) termos unificam com termos, se eles casam

- pai(P,F) = pai(joao, ze) com θ=(P/joao, F/ze).
- 2 = 2, joao = 'joao', etc.

d) ....

Unificação

d) Um termo não-atômico unifica com outro se houver correspondência estrutural.

livro(autor(Sn,N),titulo('Clarissa')) =livro(autor('Verissimo','Erico'), T).

com θ = (Sn/'Verissimo', N/'Erico', T/titulo('Clarissa'))

Execução

?- livro(autor(Sn,N),titulo('Clarissa')) = livro(autor('Verissimo','Erico'), T).

Sn = 'Verissimo',

N = 'Erico',

T = titulo('Clarissa').

Prova por Refutação

$$
\begin{aligned}
BC \Rightarrow P \quad &\Leftrightarrow \quad \neg\neg(BC \Rightarrow P) \\
&\Leftrightarrow \quad \neg\neg(\neg BC \vee P) \\
&\Leftrightarrow \quad \neg(BC \wedge \neg P) \\
&\Leftrightarrow \quad \neg(BC \wedge \neg P) \vee \text{False} \\
&\Leftrightarrow \quad (BC \wedge \neg P) \Rightarrow \text{False}
\end{aligned}
$$

Resolução

$$a)\ \frac{A \vee B, \neg B \vee C}{A \vee C}$$

$b)$ Seja os literais p_i, q, r e s_i,

onde o unificador $\theta = (pj/\neg q)$, então

$$\frac{r :- p_1, \ldots, p_j, \ldots p_m \qquad q :- s_1, \ldots, s_n}{r :- p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m}$$

Resolução

Duas cláusulas de Horn são resolvidas em uma nova cláusula se uma delas contiver um predicado negado que corresponda a um predicado não-negado na outra cláusula.

- Tautologia

A nova cláusula elimina o termo de correspondência e fica disponível para uso em resposta a pergunta.

- As variáveis são substituídas por constantes associadas de maneira consistente

Exemplo de Resolução por

come (urso, peixe).
come (urso, raposa).

come (cavalo, mato).
animal (urso).
animal (peixe).

animal (raposa).
presa(X) :- come(Y,X), animal(X).

Observe que a regra (Cláusula de Horn)
presa(X) :- come(Y,X), animal(X)

Corresponde a wff
∀X∀Y(come(Y,X) ^ animal(X) -> presa(X))

Corresponde a cláusula
¬(come(Y,X) ^ animal(X)) v presa(X)
¬come(Y,X) v ¬animal(X) v presa(X)

Regra de Inferência: Resolução

?- presa(X).

- O Prolog procura na BC por uma regra com o predicado presa(X) como o conseqüente
- Busca outras cláusulas que possam ser resolvidas com a regra
- Faz as substituições das variáveis na cláusula regra

1. ¬come(Y,X) v ¬animal(X) v presa(X)
2. come(urso,peixe)
3. ¬animal(peixe) v presa(peixe) {resolvente de 1 e 2}
4. animal (peixe)
5. presa (peixe) {resolvente de 3 e 4}