Aula XII

Alexsandro Santos Soares
prof.asoares@gmail.com

Bacharelado em Sistemas de Informação
Faculdade de Computação
Universidade Federal de Uberlândia

4 de maio de 2011

Sumário

Sumário

Sumário

Sumário

Objetivos desta aula

- Mostrar como simular construções típicas de linguagens imperativas.
- Implementar algumas operações da lógica proposicional.

Laços

- Em Prolog usamos recursão quando queremos realizar laços.
- Entretanto, podemos simular a iteração com o predicado `repeat/0` que já vem pré-definido no Prolog e cuja definição é:

```
repeat.
repeat:- repeat.
```

- Exemplo de um menu simples:

```
% Um menu simples
menu:-
  repeat,
  write('1 - Opção A.'), nl,
  write('2 - Opção B.'), nl,
  write('3 - Sair.'), nl, nl,
  write('Digite a opção: '),
  read(Opcao),
  processa(Opcao),
  Opcao=3.
```

Laço *for*

- A combinação de `repeat` com `fail` nos permite simular um laço do tipo *for*

```
for(X,[X,Z]) :- X =< Z.
for(X,[Y,Z]) :-
   W is Y+1,
   W =< Z,
   for(X,[W,Z]).
```

- Um possível uso deste novo predicado é

```
?- for(X,[1,3]),write(X),tab(1),fail.
1 2 3
false.
```

Condicional if–then–else

- Podemos também simular o comando condicional *if–then–else*:

```
ifThenElse(T,E,_S):- T, !, E.
ifThenElse(_T,_E,S):- S.
```

- Um possível uso deste novo predicado é na definição do predicado `maior` que encontra o maior dentre dois números

```
maior(X,Y,Z):-
    ifThenElse(X>Y,Z=X,Z=Y).
```

- Uso

```
?- maior(2,3,Z).
Z = 3.

?- maior(4,3,Z).
Z = 4.
```

Lógica proposicional

- O alfabeto da lógica prosicional (LP) é constituído por:
  - **símbolos verdade**: `true` e `false`;
  - **Símbolos proposicionais**: P, Q, R, S, P_1, P_2, P_3, etc;
  - **Conectivos proposicionais**: $\neg$ (não), $\vee$ (ou inclusivo), $\wedge$ (e), $\rightarrow$ (implica) e $\leftrightarrow$ (equivalência, bi-implicação); e
  - **Símbolos de pontuação**: (e).

Definição dos operadores da LP em Prolog

- Para evitar os parênteses em uma fórmula, usamos esta precedência:
  - **maior**: $\neg$
  - **intermediária**: $\vee$ e $\wedge$
  - **menor**: $\rightarrow$ e $\leftrightarrow$

Definição dos operadores da LP em Prolog

- Para evitar os parênteses em uma fórmula, usamos esta precedência:
  - **maior**: $\neg$
  - **intermediária**: $\vee$ e $\wedge$
  - **menor**: $\rightarrow$ e $\leftrightarrow$
- Em Prolog, os conectivos e as precedências anteriores ficam assim:

```
:- op(970,xfx,<=>). % bi-implicação
:- op(960,xfy,=>). % implicação
:- op(950,xfy,v).   % ou
:- op(940,xfy,&).   % e
:- op(930,fy,~).    % não
```

Definição dos operadores da LP em Prolog

- Para evitar os parênteses em uma fórmula, usamos esta precedência:
  - **maior**: $\neg$
  - **intermediária**: $\vee$ e $\wedge$
  - **menor**: $\rightarrow$ e $\leftrightarrow$
- Em Prolog, os conectivos e as precedências anteriores ficam assim:

```
:- op(970,xfx,<=>). % bi-implicação
:- op(960,xfy,=>). % implicação
:- op(950,xfy,v).    % ou
:- op(940,xfy,&).    % e
:- op(930,fy,~).     % não
```

- Para verificarmos a definição podemos usar as seguinte consultas

```
?- display( ((p & q) v ((~p) => (~q))) ).
v(&(p,q),=>(~(p),~(q)))

?- display( (p v r) => true  <=>  q & s ).
<=>(=>(v(p,r),true),&(q,s))
```

Fórmulas da LP

- As fórmulas da linguagem da Lógica Proposicional são construídas, de forma indutiva, conforme as regras a seguir:
  - todo símbolo de verdade é uma fórmula;
  - todo símbolo proposicional é uma fórmula;
  - se H é uma fórmula, então $(\neg H)$, a negação de H, é uma fórmula;
  - se H e G são fórmulas, então a disjunção de H e G, dada por $(H \vee G)$, é uma fórmula;
  - se H e G são fórmulas, então a conjunção de H e G, dada por $(H \wedge G)$, é uma fórmula;
  - se H e G são fórmulas, então a implicação de H em G, dada por $(H \rightarrow G)$, é uma fórmula.
  - se H e G são fórmulas, então a bi-implicação de H e G, dada por $(H \leftrightarrow G)$, é uma fórmula.

Reconhecimento de fórmulas

- todo símbolo de verdade é uma fórmula:

```
formula(true).
formula(false).
```

Reconhecimento de fórmulas

- todo símbolo de verdade é uma fórmula:

```
formula(true).
formula(false).
```

- todo símbolo proposicional é uma fórmula:

```
formula(S):- atom(S).
```

Reconhecimento de fórmulas

- todo símbolo de verdade é uma fórmula:

```
formula(true).
formula(false).
```

- todo símbolo proposicional é uma fórmula:

```
formula(S):- atom(S).
```

- se H é uma fórmula, então $(\neg H)$, a negação de H, é uma fórmula:

```
formula(~ H):-
   formula(H).
```

Reconhecimento de fórmulas

- todo símbolo de verdade é uma fórmula:

```
formula(true).
formula(false).
```

- todo símbolo proposicional é uma fórmula:

```
formula(S):- atom(S).
```

- se H é uma fórmula, então $(\neg H)$, a negação de H, é uma fórmula:

```
formula(~ H):-
   formula(H).
```

- se H e G são fórmulas, então a disjunção de H e G, dada por $(H \vee G)$, é uma fórmula;

```
formula(H v G):-
   formula(H),
   formula(G).
```

Reconhecimento de fórmulas – continuação

- se H e G são fórmulas, então a conjunção de H e G, dada por $(H \wedge G)$, é uma fórmula:

```
formula(H & G):-
   formula(H),
   formula(G).
```

Reconhecimento de fórmulas – continuação

- se H e G são fórmulas, então a conjunção de H e G, dada por $(H \wedge G)$, é uma fórmula:

```
formula(H & G):-
   formula(H),
   formula(G).
```

- se H e G são fórmulas, então a implicação de H em G, dada por $(H \rightarrow G)$, é uma fórmula:

```
formula(H => G):-
   formula(H),
   formula(G).
```

Reconhecimento de fórmulas – continuação

- se H e G são fórmulas, então a conjunção de H e G, dada por $(H \wedge G)$, é uma fórmula:

```
formula(H & G):-
   formula(H),
   formula(G).
```

- se H e G são fórmulas, então a implicação de H em G, dada por $(H \rightarrow G)$, é uma fórmula:

```
formula(H => G):-
   formula(H),
   formula(G).
```

- se H e G são fórmulas, então a bi-implicação de H e G, dada por $(H \leftrightarrow G)$, é uma fórmula:

```
formula(H <=> G):-
   formula(H),
   formula(G).
```

Reconhecimento de fórmulas – teste

- Para testar o código vamos escrever alguns predicados:

```
fexemplo(1, (p v r) => true ).
fexemplo(2, (((p v r) => true) <=> (q & s)) ).
fexemplo(3,  (p v r) => true  <=> q & s  ).
fexemplo(4, ((p v r) => true) <=> (q & s) ).
```

- No interpretador podemos fazer as consultas:

```
?- fexemplo(1,F), formula(F).
F = (p v r=>true)

?- fexemplo(2,F), formula(F).
F = (p v r=>true<=>q&s)

?- fexemplo(3,F), formula(F).
F = (p v r=>true<=>q&s)

?- fexemplo(4,F), formula(F).
F = (p v r=>true<=>q&s)
```

Reconhecimento de fórmulas – exercício

- Elimine o maior número possível de parênteses da fórmula, sem alterar seu significado original: $((\neg X) \vee ((\neg(X \vee Y)) \vee Z))$

Reconhecimento de fórmulas – exercício

- Elimine o maior número possível de parênteses da fórmula, sem alterar seu significado original: $((\neg X) \vee ((\neg(X \vee Y)) \vee Z))$
- Para resolver este exercício podemos usar o nosso programa. Primeiro acrescentamos mais um exemplo:

```
fexemplo(5, ((~x) v ((~(x v y)) v z)) ).
```

Reconhecimento de fórmulas – exercício

- Elimine o maior número possível de parênteses da fórmula, sem alterar seu significado original: $((\neg X) \vee ((\neg(X \vee Y)) \vee Z))$
- Para resolver este exercício podemos usar o nosso programa. Primeiro acrescentamos mais um exemplo:

```
fexemplo(5, ((~x) v ((~(x v y)) v z)) ).
```

- Depois fazemos a consulta

```
?- fexemplo(5,F), display(F).
v(~(x),v(~(v(x,y)),z))
F = (~x v ~ (x v y) v z).
```

Comprimento de uma fórmula

- Seja H uma fórmula da LP, o *comprimento* de H, denotado por $comp[H]$, é definido por
  - Se $H = P$ é um símbolo de verdade, então $comp[H] = 1$;
  - $comp[\neg H] = comp[H] + 1$;
  - $comp[H \vee G] = comp[H] + comp[G] + 1$;
  - $comp[H \wedge G] = comp[H] + comp[G] + 1$;
  - $comp[H \rightarrow G] = comp[H] + comp[G] + 1$;
  - $comp[H \leftrightarrow G] = comp[H] + comp[G] + 1$.

Comprimento de uma fórmula em Prolog

- Se H é um símbolo de verdade ou proposicional, então $comp[H] = 1$:

```
comp(true,1).
comp(false,1).
comp(S,1):- atom(S).
```

Comprimento de uma fórmula em Prolog

- Se H é um símbolo de verdade ou proposicional, então $comp[H] = 1$:

```
comp(true,1).
comp(false,1).
comp(S,1):- atom(S).
```

- $comp[\neg H] = comp[H] + 1$:

```
comp(~ H, C):-
    comp(H,Ch),
    C is Ch + 1.
```

Comprimento de uma fórmula em Prolog

- Se H é um símbolo de verdade ou proposicional, então $comp[H] = 1$:

```
comp(true,1).
comp(false,1).
comp(S,1):- atom(S).
```

- $comp[\neg H] = comp[H] + 1$:

```
comp(~ H, C):-
    comp(H,Ch),
    C is Ch + 1.
```

- $comp[H \vee G] = comp[H] + comp[G] + 1$:

```
comp(H v G, C):-
    comp(H,Ch),
    comp(G,Cg),
    C is Ch + Cg +1.
```

Comprimento de uma fórmula em Prolog – continuação

- $comp[H \land G] = comp[H] + comp[G] + 1$:

```
comp(H & G, C):-
    comp(H,Ch),
    comp(G,Cg),
    C is Ch + Cg +1.
```

Comprimento de uma fórmula em Prolog – continuação

- $comp[H \wedge G] = comp[H] + comp[G] + 1$:

```
comp(H & G, C):-
   comp(H,Ch),
   comp(G,Cg),
   C is Ch + Cg +1.
```

- $comp[H \rightarrow G] = comp[H] + comp[G] + 1$:

```
comp(H => G, C):-
   comp(H,Ch),
   comp(G,Cg),
   C is Ch + Cg +1.
```

Comprimento de uma fórmula em Prolog – continuação

- $comp[H \wedge G] = comp[H] + comp[G] + 1$:

```
comp(H & G, C):-
    comp(H,Ch),
    comp(G,Cg),
    C is Ch + Cg +1.
```

- $comp[H \rightarrow G] = comp[H] + comp[G] + 1$:

```
comp(H => G, C):-
    comp(H,Ch),
    comp(G,Cg),
    C is Ch + Cg +1.
```

- $comp[H \leftrightarrow G] = comp[H] + comp[G] + 1$:

```
comp(H <=> G, C):-
    comp(H,Ch),
    comp(G,Cg),
    C is Ch + Cg +1.
```

Comprimento de uma fórmula em Prolog – teste

- Vamos realizar alguns testes com o novo predicado:

```
?- comp(~p,C).
C = 2.

?- comp(p v q, C).
C = 3.

?- comp( p v q => true, C).
C = 5 .

?- comp( (p v r) => true  <=>  q & s, C).
C = 9
```