Roteiro 5

Alexsandro Santos Soares
prof.asoares@gmail.com

Programação Lógica
Faculdade de Computação
Universidade Federal de Uberlândia

25 de junho de 2022

Este roteiro tem por finalidades:

- Ilustrar o uso de predicados pré-construídos para imprimir termos na tela.
- Praticar a definição e uso de operadores definidos pelo usuário.
- Praticar o uso de cortes e da negação como falha.

1 Predicados para impressão de termos

Nesta sessão prática, pretendemos introduzir alguns predicados pré-construídos para imprimir termos na tela.

O primeiro predicado a ser visto é `display/1`, que recebe um termo e o imprime na tela.

```
?- display(ama(vicente,maria)).
ama(vicente,maria)
true.

?- display('julio come um grande sanduíche').
julio come um grande sanduíche
true.
```

Mais estritatemente falando, `display` imprime a representação interna do Prolog para termos.

```
?- display(2+3+4).
+(+(2,3),4)
true.
```

De fato, esta propriedade de `display` torna-o muito útil para aprender como operadores funcionam em Prolog. Assim, antes de aprender mais como escrever coisas na tela, tente as seguintes consultas. Assegure-se de compreender porque o Prolog responde da forma que ele faz.

```
?- display([a,b,c]).
?- display(3 is 4 + 5 / 3).
?- display(3 is (4 + 5) / 3).
?- display((a:-b,c,d)).
?- display(a:-b,c,d).
```

Assim, `display` é bom para olhar na representação interna dos termos na notação de operadores, mas normalmente nós provavelmente preferiríamos imprimir na notação mais amigável. Especialmente na impressão de listas, seria muito melhor ter `[a,b,c]` do que `'[|]'(a'[|]'(b'[|]'(c,[])))`. Isto é o que faz o predicado pré-construído `write/1`. Ele recebe um termo e o imprime na tela usando a notação mais amigável.

```
?- write(2+3+4).
2+3+4
true

?- write(+(2,3)).
2+3
true

?- write([a,b,c]).
[a, b, c]
true

?- write('[|]'(a,'[|]'(b,[]))).
[a, b]
true
```

E aqui está o que acontece quando o termo a ser escrito contém variáveis.

```
?- write(X).
_G204
true

?- X = a, write(X).
a
X = a.
```

O exemplo seguinte mostra o que acontece quando você coloca dois comandos write, um após o outro.

```
?- write(a), write(b).
ab
true
```

Prolog apenas executa um após o outro sem colocar qualquer espaço entre a saída dos diferentes comandos write. Naturalmente, você pode dizer ao Prolog para imprimir espaços pedindo para escrever o termo `' '`:

```
?- write(a), write(' '), write(b).
a b
```

E se você quiser mais que um espaço, por exemplo cinco espaços, diga ao Prolog para escrever ' ':

```
?- write(a), write('     '), write(b).
a     b
```

Uma outra forma de imprimir espaços é pelo uso do predicado `tab/1`. Este predicado recebe um número como argumento e então imprime tantos espaços quanto especificado por este número.

```
?- write(a), tab(5), write(b).
a     b
```

Um outro predicado útil para formatação é `nl`. Este predicado diz ao Prolog para saltar uma linha:

```
?- write(a), nl, write(b).
a
b
```

2 Exercícios

Ex. 1 Quais das seguintes consultas tem sucesso e quais falham?

```
?- 12 is 2*6.

?- 14 =\= 2*6.

?- 14 = 2*7.

?- 14 == 2*7.

?- 14 \== 2*7.

?- 14 =:= 2*7.

?- [1,2,3|[d,e]] == [1,2,3,d,e].

?- 2+3 == 3+2.

?- 2+3 =:= 3+2.

?- 7-2 =\= 9-2.

?- p == 'p'.

?- p =\= 'p'.

?- vicente == VAR.

?- vicente=VAR, VAR==vicente.
```

Ex. 2 Como Prolog responde às seguintes consultas?

```
?- '[|]'(a,'[|]'(b,'[|]'(c,[]))) = [a,b,c].

?- '[|]'(a,'[|]'(b,'[|]'(c,[]))) = [a,b|[c]].

?- '[|]'('[|]'(a,[]),'[|]'('[|]'(b,[]),'[|]'('[|]'(c,[]),[]))) = X.

?- '[|]'(a,'[|]'(b,'[|]'('[|]'(c,[]),[]))) = [a,b|[c]].
```

Ex. 3 Escreva um predicado binário `tipotermo(+Termo,?Tipo)` que recebe um termo e devolve o(s) tipo(s) daquele termo: átomo, número, constante, variável, etc. Os tipos devem ser devolvidos em ordem de generalidade. O predicado deveria, por exemplo, comportar-se da seguinte forma.

```
?- tipotermo(Vicente, variável).
true
?- tipotermo(maria, X).
X = átomo ;
X = constante ;
X = termo_simples ;
X = termo ;
false
?- tipotermo(vivo(zeca), X).
X = termo_complexo ;
X = termo ;
false
```

Ex. 4 Escreva um programa que define o predicado `termoaterrado(+Termo)` que testa se `Termo` é um termo aterrado. Termos aterrados são termos que não contém variáveis. Aqui estão exemplos de como o predicado deveria comportar-se:

```
?- termoaterrado(X).
false
?- termoaterrado(francês(bic_mac,le_bic_mac)).
true
?- termoaterrado(francês(mentiroso,X)).
false
```

Ao fazer os exercícios não use qualquer predicado pré definido ou de alguma biblioteca que resolva diretamente o problema pedido.

3 Uso de operadores definidos pelo usuário

Considere a seguintes regras de um sistema que auxilie na descoberta de um vazamento:

- Se a cozinha está seca e o corredor molhado então o vazamento de água está no banheiro.

- Se o corredor está molhado e o banheiro está seco então o problema está na cozinha.
- Se a janela está fechada ou não chove então não entra água do exterior.
- Se o problema está na cozinha e não entra água do exterior então o vazamento de água está na cozinha.

Considere também as seguintes evidências:

- O corredor está molhado.
- O banheiro está seco.
- A janela está fechada.

O que deseja-se saber é:

- Onde está o vazamento?

Uma forma de resover este problema em Prolog é codificar as regras, fatos e a consulta no formato das cláusulas de Horn que o Prolog aceita diretamente. Este solução é mostrada no código a seguir.

```
% Se a cozinha está seca e o corredor molhado
% então o vazamento de água está no banheiro.
vazamento(banheiro):- seco(cozinha), molhado(corredor).

% Se o corredor está molhado e o banheiro está seco
% então o problema está na cozinha.
problema(cozinha):- molhado(corredor), seco(banheiro).

% Se a janela está fechada ou não chove
% então não entra água do exterior.
não_entra_água(exterior):- fechado(janela); não(chove).

% Se o problema está na cozinha e não entra água do exterior
% então o vazamento de água está na cozinha.
vazamento(cozinha):- problema(cozinha), não_entra_água(exterior).

% Evidências:
% O corredor está molhado.
molhado(corredor).
% O banheiro está seco.
seco(banheiro).
% A janela está fechada.
fechado(janela).
```

Salve este código em um arquivo e depois o consulte. A consulta que resolve o problema é feita da seguinte forma:

```
?- vazamento(Onde).
```

Agora, imaginando que a pessoa que de fato vai escrever as regras sobre encanamento não seja versada em lógica formal, pode-se escrever estas mesmas regras, fatos e consulta com a ajuda dos operadores definidos pelo usuário. Assim, uma outra solução seria a mostrada a seguir.

```
:-op(875,xfx, fato).
:-op(875,xfx, #).
:-op(825,fx,  se).
:-op(850,xfx, então).
:-op(800,xfy, ou).  % Associatividade à direita
:-op(775,xfy, e).   % Associatividade à direita
:-op(750,fy,  não). % Associatividade à direita

% Se a cozinha está seca e o corredor molhado
% então o vazamento de água está no banheiro.
r1 # se cozinha_seca e corredor_molhado
     então vazamento_no_banheiro.

% Se o corredor está molhado e o banheiro está seco
% então o problema está na cozinha.
r2 # se corredor_molhado e banheiro_seco
     então problema_na_cozinha.

% Se a janela está fechada ou não chove
% então não entra água do exterior.
r3 # se janela_fechada ou não chove
     então não entra_água_do_exterior.

% Se o problema está na cozinha e não entra água do exterior
% então o vazamento de água está na cozinha.
r4 # se problema_na_cozinha e não entra_água_do_exterior
     então vazamento_na_cozinha.

% Evidências:
% O corredor está molhado.
f1 fato corredor_molhado.

% O banheiro esta seco.
f2 fato banheiro_seco.

% A janela está fechada.
f3 fato janela_fechada.

deduz(P):- _ fato P.
deduz(P):- _ # se C então P, deduz(C).
deduz(não P):- \+ deduz(P).
deduz(P1 e P2):- deduz(P1), deduz(P2).
deduz(P1 ou  _):- deduz(P1).
deduz(_  ou P2):- deduz(P2).
```

Para esta nova versão foi construído um predicado de nome `deduz` que se encarrega de fazer as deduções lógicas. Salve o código anterior em um novo arquivo e o consulte.

A nova consulta para o problema pode ser formulada assim:

```
?- deduz(vazamento_na_cozinha).
```

Ex. 5 Assuma que temos as seguintes definições de operadores:

```
:- op(300, xfx, [são, é_um]).
:- op(300, fx, gosta_de).
:- op(200, xfy, e).
:- op(100, fy, famoso).
```

Quais dos termos seguintes são bem formados? Qual é o operador principal? Reescreva-os com parênteses na ordem correta de avaliação.

```
?- X é_um bruxo.
?- harry e ron e hermione são amigos.
?- harry é_um mago e gosta_de quadribol.
?- dumbledore é_um famoso famoso mago.
```

Ex. 6 Defina um operador para indicar horários. Por exemplo, para indicar

(a) duas horas e quinze minutos, o Prolog deveria aceitar o termo `2 h 15`.

(b) 1 hora e cinquenta minutos, o Prolog deveria aceitar o termo `1 h 50`.

Ex. 7 Escreva um predicado `soma_hora/3` que recebe dois horários no formato indicado no exercício anterior e instancia o terceiro argumento com a soma dos dois horários juntos:

```
?- soma_hora(2 h 30, 1 h 50, Resultado).
Resultado = 4 h 20
```

Ex. 8 Defina um predicado `mult_hora/3` que recebe um número natural positivo, um horário e instancia o terceiro argumento com o resultado de multiplicar o horário pelo natural dado:

```
?- mult_hora(3, 1 h 25, Resultado).
Resultado = 4 h 15
```

Ex. 9 Defina um operador infixo `++` que combina dois horários em uma expressão, tal como `3 h 20 ++ 4 h 10`. Para este exercício basta definir o operador, depois o usaremos para representar a soma de dois horários.

Ex. 10 Defina um operador infixo `**` que combina um natural e um horário em uma expressão, tal como `3 ** 4 h 10`. Para este exercício basta definir o operador, depois o usaremos para representar a multiplicação de um horário por um natural. Dê a este operador uma precedência menor que a de `++`.

Ex. 11 Defina um operador infixo `<-` e um predicado adequado para este operador funcionar com expressões horárias, tais como as que definiu nos dois exercícios anteriores, da mesma forma que o operador `is` funciona para aritmética. Ou seja, ele deve avaliar expressões horárias.

O segundo argumento do operador (à direita) deveria ser um expressão horária usando `h`, `++` e `**`. O operador `<-` deveria avaliar a expressão horária à direita dele e unificar o horário resultante com o argumento do lado esquerdo. Por exemplo:

```
?-  Horário <- 3 h 10 ++ 5 h 20.
Horário = 8 h 30.

?- Horário <- 3 ** 1 h 10  ++  2 ** 2 h 40.
Horário = 8 h 50.
```

4 Exercícios envolvendo cortes

Ex. 12 Assuma que se tenha o seguinte banco de dados:

```
p(1).
p(2):- !.
p(3).
```

Escreva todas as respostas do Prolog às seguintes consultas:

```
?- p(X).

?- p(X),p(Y).

?- p(X),!,p(Y).
```

Ex. 13 Primeiro, explique o que o seguinte programa faz:

```
classe(Numero, positivo):- Numero > 0.
classe(0, zero).
classe(Numero, negativo):- Numero < 0.
```

Depois, melhore-o pela adição de cortes.

Ex. 14 Sem usar corte, escreva um predicado `separa/3` que separa uma lista de inteiros em duas listas: uma contendo os números positivos e zero, e uma outra contendo números negativos. Por exemplo:

```
?- separa([3,4,-5,-1,0,4,-9],P,N).
P = [3,4,0,4]
N = [-5,-1,-9].
```

Ex. 15 Agora, usando o corte, melhore o programa anterior, *sem* alterar seu significado.

As consultas a seguir devem ser feitas para cada um dos exercícios informados a seguir.

```
?- f(p).

?- f(q).

?- f(r).
```

```
?- f(X).
```

Diga quais seriam as respostas do Prolog para cada uma das consultas anteriores, se os programas carregados fossem os mostrados na sequência. Além disso, desenhe a **árvore de prova** de cada consulta.

Ex. 16
```
f(X) :- !,X=p.
f(X) :- !,X=q.
f(X) :- X = r.
```

Ex. 17
```
f(X) :- X=p,!.
f(X) :- X=q,!.
f(X) :- X=r.
```

Ex. 18
```
f(X) :- X=p,!.
f(X) :- !,X=q.
f(X) :- X=r.
```

Ex. 19
```
f(X) :- !,X=p.
f(X) :- X=q,!.
f(X) :- X=r.
```

Ex. 20
```
f(X) :- X=p.
f(X) :- X=q,!.
f(X) :- X=r.
```

Ex. 21
```
f(p) :- !.
f(q) :- !.
f(r).
```

Ex. 22
```
f(p).
f(q):- !.
f(r).
```

5 Corte e negação como falha

Ex. 23 Faça experimentações com as três versões do predicado `max/3` definidas na aula teórica: a versão sem corte, a versão com corte verde e a versão com corte vermelho. Como usual, "experimentar" significa "executar o trace", assegurando-se que rastreie consultas na qual todos os três argumentos estão instanciados para inteiros e, também, consultas onde o terceiro argumento é uma variável.

Ex. 24 Experimente todos os métodos discutidos na aula para lidar com as preferências de Vicente. Isto é, faça experimentos com o programa que usa a combinação de corte com `fail`, com o programa que usa negação como falha corretamente e também com o programa que torna-se errôneo quando utiliza negação no lugar errado.

Ex. 25 Defina um predicado `nu/2` ("não unificável") que recebe dois termos como argumentos e sucede se os dois termos não unificam. Por exemplo:

```
?- nu(foo,foo).
false

?- nu(foo,blob).
true

?- nu(foo,X).
false
```

Você deve definir este predicado de três formas diferentes:

a) Primeiro (e mais fácil), escreva-o com a ajuda de `=` e `\+`.

b) Segundo, escreva-o com a ajuda de `=`, mas não use `\+` e `not`.

c) Terceiro, escreva usando uma combinação de corte e `fail`. Não use `=`, `\+` e `not`.

Ex. 26 Defina um predicado `unificável(Lista1,Termo,Lista2)` onde `Lista2` é a lista de todos os membros da `Lista1` que poderiam se unificar com `Termo`, mas *não* são instanciados pela unificação. Por exemplo,

```
?- unificável([X,b,t(Y)],t(a),Lista).
Lista = [X,t(Y)].
```

Note que `X` e `Y` ainda *não* estão instanciadas na resposta. Assim a parte complicada é: como verificar se elas unificam com `t(a)` sem instanciá-las? (Dica: considere usar o teste `\+ (termo1 = termo2)`. Por quê? Pense sobre isto. Talvez você deseje também pensar sobre o teste `\+(\+ (termo1 = termo2))`).

6 Sugestões de leitura

- Luiz A. M. Palazzo. *Introdução à programação Prolog*

  http://puig.pro.br/Logica/palazzo.pdf

- Eloi L. Favero. *Programação em Prolog: uma abordagem prática*

  http://www3.ufpa.br/favero

- Wikilivro sobre Prolog em

  http://pt.wikibooks.org/wiki/Prolog

- Patrick Blackburn, Johan Bos and Kristina Striegnitz. *Learn Prolog Now!*

  http://www.learnprolognow.org