Roteiro 5

Alexsandro Santos Soares
prof.asoares@gmail.com

Programação Lógica
Faculdade de Computação
Universidade Federal de Uberlândia

21 de agosto de 2021

Este roteiro tem por finalidades:

- Continuar o estudo de DCGs simples e aquelas que utilizam argumentos e testes adicionais.
- Ilustrar o uso de predicados pré-construídos para imprimir termos na tela.
- Praticar a definição e uso de operadores definidos pelo usuário.

1 Exercícios envolvendo DCGs

Ex. 1 A linguagem formal $a^n b^{2m} c^{2m} d^n$ consiste de todas as strings da seguinte forma: um bloco contíguo de `a`s seguido por um bloco contíguo de `b`s, seguido por um bloco contíguo de `c`s, seguido por um bloco contíguo de `d`s, tal que os blocos `a` e `d` são exatamente do mesmo tamanho, e os blocos `b` e `c` são exatamente do mesmo tamanho e, além disto, consistem de um número par de `b`s e `c`s. Por exemplo, ϵ, *abbccd* e *aaabbbbccccddd* pertecem a $a^n b^{2m} c^{2m} d^n$. Escreva uma DCG que gere esta linguagem.

Ex. 2 A linguagem que os lógicos chamam de *lógica proposicional sobre os símbolos proposicionais p, q e r* pode ser definida pela seguinte gramática livre de contexto

$$\begin{aligned}
prop &-> p \\
prop &-> q \\
prop &-> r \\
prop &-> \neg prop \\
prop &-> (prop \wedge prop) \\
prop &-> (prop \vee prop) \\
prop &-> (prop \rightarrow prop)
\end{aligned}$$

Escreva uma DCG que gere esta linguagem. De fato, como ainda não aprendemos sobre operadores Prolog, você terá que fazer algumas concessões que parecerão bastante desajeitadas.

Por exemplo, ao invés de reconhecer

$$\neg(p \rightarrow q)$$

você terá que reconhecer coisas como

```
[não, '(', p, implica, q, ')']
```

Use `ou` para $\vee$, e `e` para $\wedge$.

Ex. 3 Dada a seguinte DCG:

```
s --> foo,bar,wiggle.
foo --> [chu].
foo --> foo,foo.
bar --> mar,zar.
mar --> me,my.
me --> [eu].
my --> [sou].
zar --> blar,car.
blar --> [um].
car --> [trem].
wiggle --> [tchu].
wiggle --> wiggle,wiggle.
```

Escreva as regras Prolog comuns que correspondam a estas regras DCGs. Quais são as primeiras três respostas que o Prolog dá à consulta `s(X,[])`?

Ex. 4 A linguagem formal $a^n b^n - \{\epsilon\}$ consiste de todas as strings $a^n b^n$, exceto a string vazia. Escreva uma DCG que gere esta linguagem.

Ex. 5 Seja $a^n b^{2n}$ a linguagem formal que contém todas as strings da seguinte forma: um bloco contíguo de `a`s de tamanho n seguido por um bloco contíguo de `b`s de tamanho $2n$, e nada mais. Por exemplo, *abb*, *aabbbb* e *aaabbbbbb* pertencem a $a^n b^{2n}$, assim como a string vazia. Escreva uma DCG que gere esta linguagem.

2 Gramáticas de cláusulas definidas com argumentos e testes extras

Primeiro alguns exercícios de fixação:

Ex. 6 Rastreie alguns exemplos de DCG que utilizem argumentos extras para tratar a distinção sujeito/objeto, de DCG que produza análise sintática e de DCG que utilize testes extras para separar o léxico das regras. Certifique-se de que você compreenda totalmente o modo com o qual todas as três DCGs funcionam. Use os slides da aula teórica dessa semana.

Ex. 7 Realize alguns rastreamentos para a DCG para a linguagem $a^n b^n c^n$. Experimente casos onde os três blocos de *a*s, *b*s e *c*s sejam de fato do mesmo tamanho, assim como casos onde isto não ocorre.

Agora alguns exercícios para treinar a técnica.

Ex. 8 Abaixo encontra-se a nossa DCG básica.

```
s --> sn, sv.

sn --> det, n.

sv --> v, sn.
sv --> v.

det --> [o].
det --> [a].

n --> [mulher].
n --> [homem].

v --> [bate].
```

Suponha que adicionemos o nome "homens", que é plural, e o verbo "batem". Então, gostaríamos de uma DCG que diga que "O homem bate" está correto, "Os homens batem" está correto, "O homem batem" não está correto e que "Os homens bate" também não está correto. Altere a DCG tal que ela corretamente trate estas sentenças. Use um argumento extra para lidar com a distinção singular/plural.

Ex. 9 Traduza a seguinte regra DCG em um formato padrão de regras do Prolog:

```
cangu(V,R,Q) --> ru(V,R), salta(Q,Q), {marsupial(V,R,Q)}.
```

Finalmente, alguns exercícios de programação.

Ex. 10 Primeiro, reúna todas as coisas que aprendeu sobre DCGs para Português em uma única DCG. Em particular, nessa semana vimos como usar argumentos extras para lidar com a distinção sujeito/objeto, e nos exercícios anteriores você usou argumentos adicionais para lidar com a distinção singular/plural. Escreva uma DCG que trate ambos. Além disto, escreva a DCG de tal forma que ela produza árvores sintáticas e faça uso de um léxico separado.

Ex. 11 Escreva uma DCG que reconheça numerais cardinais entre zero e mil escritos por extenso em Português.

```
?- cardinal([zero], []).
true

?- cardinal([vinte,e,um], []).
true

?- cardinal([novecentos,e,trinta,e,sete], []).
true

?- cardinal([setecentos, e, setenta, e, sete], []).
true.
```

```
?- cardinal([mil], []).
true.

?- cardinal([cem,onze], []).
false.
```

Ex. 12 Modifique o reconhecedor do exercício anterior para que ele também produza os dígitos do número reconhecido em uma lista.

```
?- cardinal(N, [zero], []).
N = [0] .

?- cardinal(N, [vinte,e,um], []).
N = [2, 1] .

?- cardinal(N, [novecentos,e,trinta,e,sete], []).
N = [9, 3, 7] .

?- cardinal(N, [novecentos,e,trinta], []).
N = [9, 3, 0] .

?- cardinal([7,7,7], Extenso, []).
Extenso = [setecentos, e, setenta, e, sete] .

?- cardinal([1,0,0,0], Extenso, []).
Extenso = [mil].
```

3 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
```

4 Exercícios

Ex. 13 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. 14 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. 15 Escreva um predicado binário `tipotermo(+Termo,?Tipo)` que recebe um termo e devolve o(s) tipo(s) daquele termo(atomo, numero, contante, variavel, etc.). Os tipos devem ser devolvidos em ordem de generalidade. O predicado deveria, por exemplo, comportar-se da seguinte forma.

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

Ex. 16 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
```

5 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. 17 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. 18 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. 19 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. 20 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. 21 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. 22 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. 23 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.
```

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`