Programação em Lógica

Prof. A. G. Silva

21 de setembro de 2017

I – Exercícios sobre bases dinâmicas

II – Gramáticas

Exercícios

Sugestões de resoluções dos exercícios 1 a 6, da aula passada, em cor púrpura. Exercícios 7 e 8 ainda não resolvidos; pense em suas implementações, tomando como base o código da questão 6.

1. Escreva um predicado `estrelas(N)` que imprime `N` caracteres "`*`" no dispositivo de saida.

```
estrelas(0) :- !.
estrelas(N) :- N > 0,
               M is N - 1,
               write('*'),
               estrelas(M).
```

Exercícios (cont...)

2. Escreva um predicado `guess(N)` que incita o usuário a adivinhar o número `N`. O predicado repetidamente lê um número, compara-o com `N`, e imprime "`Muito baixo!`", "`Acertou!`", "`Muito alto!`, conforme o caso, orientando o usuário na direção certa.

```
compara(N,X) :- N == X,
                write('Acertou! ').
compara(N,X) :- N < X,
                write('Muito alto! '),
                guess(N).
compara(N,X) :- N > X,
                write('Muito baixo! '),
                guess(N).
guess(N) :- write('Adivinhe N: '),
            read(X),
            compara(N,X).
```

Exercícios (cont...)

3. Escreva um predicado que lê uma linha e imprime a mesma linha trocando todos os caracteres 'a' por 'b'.

```
troca :- write('Escreva algo: '),
         read(X),
         string_to_list(X,Y),
         troca(Y).
troca([]) :- !.
troca([A|B]) :- char_code('a',Code),
                A=:=Code, write('b'),
                troca(B).
troca([A|B]) :- char_code(Char,A),
                write(Char),
                troca(B).
```

Exercícios (cont...)

4. Implemente os predicados `liga`, `desliga` e `lâmpada` para que eles funcionem conforme indicado pelos exemplos a seguir:

```
?- liga, lampada(X).
X = acessa
Yes
?- desliga, lampada(X).
X = apagada
Yes
```

```
:- dynamic lampada/1.

lampada(acesa).
liga :- retract(lampada(_)),
        assert(lampada(acesa)).
desliga :- retract(lampada(_)),
           assert(lampada(apagada)).
```

Exercícios (cont...)

5. O predicado `asserta` adiciona um fato à base de dados, incondicionalmente, mesmo que ele já esteja lá. Para impedir essa redundância, defina o predicado `memorize`, tal que ele seja semelhante a `asserta`, mas só adicione à base de dados fatos inéditos.

```
% (testar com um predicado dinamico)

memorize(A) :- retractall(A),
               asserta(A),
               !.
memorize(A) :- asserta(A).
```

Exercícios (cont...)

6. Suponha um robô capaz de andar até um certo local e pegar ou soltar objetos. Além disso, suponha que esse robô mantém numa base de dados sua posição corrente e as respectivas posições de uma série de objetos. Implemente os predicados `pos(Obj,Loc)`, `ande(Dest)`, `pegue(Obj)` e `solte(Obj)`, de modo que o comportamento desse robô possa ser simulado, conforme exemplificado a seguir:

```
?- pos(O,L).
O = robô
L = garagem ;
O = tv
L = sala ;
No

?- pegue(tv), ande(quarto), solte(tv), ande(cozinha).
anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
Yes
```

Exercícios (cont...)

6.
```
:- dynamic pos/2.
:- dynamic posse/1.

pos(robo, garagem).
pos(tv, sala).
pos([], []). %objeto indefinido em local indefinido
posse([]).  %objeto que o robo esta segurando, inicialmente indefinido

pegue(O) :- pos(robo,L1), pos(O,L2),
            retract(posse(_)),
            asserta(posse(O)),
            retract(pos(robo,_)), asserta(pos(robo,L2)),
            format('anda de ~w ate ~w\npega ~w\n', [L1,L2,O]).
ande(L) :- pos(robo,L1),
           retract(pos(robo,_)),
           format('anda de ~w ate ~w\n', [L1,L]),
           asserta(pos(robo,L)),
           posse(O),
           retract(pos(O,_)), asserta(pos(O,L)).
solte(O) :- posse(O),
            retract(posse(_)),
            asserta(posse([])).
```

Exercícios (cont...)

7. *(não resolvido)* Modifique o programa desenvolvido no exercício anterior de modo que, quando for solicitado ao robô pegar um objeto cuja posição é desconhecida, ele pergunte ao usuário onde está esse objeto e atualize a sua base de dados com a nova informação. Veja um exemplo:

```
?- pos(O,L).
O = robô
L = cozinha ;
O = tv
L = quarto ;
No

?- pegue(lixo), ande(rua), solte(lixo), ande(garagem).
Onde está lixo? quintal
anda de cozinha até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
Yes

?- pos(O,L).
O = robô
L = garagem ;
O = lixo
L = rua ;
O = tv
L = quarto ;
No
```

Exercícios (cont...)

8. *(não resolvido)* Acrescente também ao programa do robô o predicado `leve(Obj,Loc)`, que leva um objeto até um determinado local. Por exemplo:

```
?- leve(tv,sala).
anda de garagem até quarto
pega tv
anda de quarto até sala
solta tv
Yes
```

Gramáticas

Gramáticas

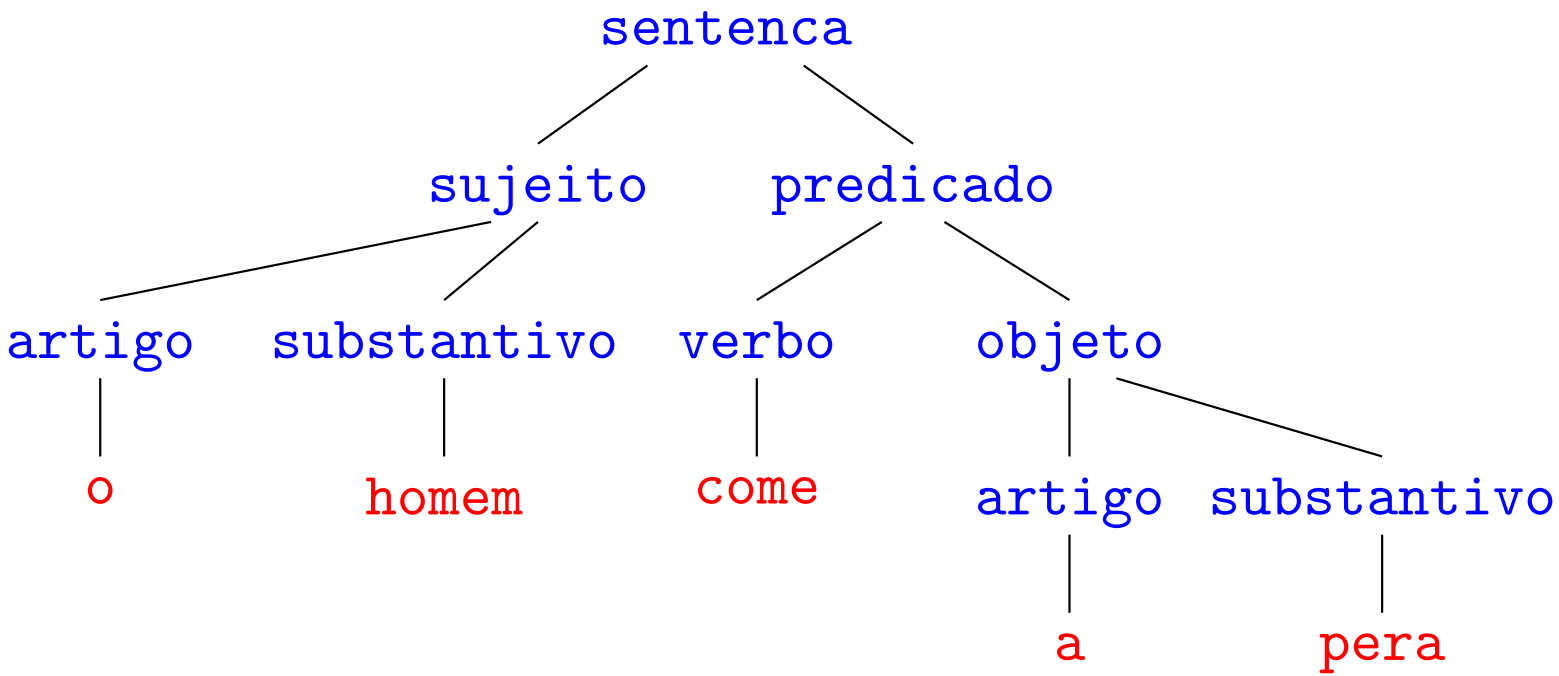
- Processamento de linguagens naturais é um dos principais usos de Prolog
- Por ser tão utilizado, SWI-Prolog (entre outras implementações) oferece mecanismos específicos para tratar este tema
- Para entender uma frase em uma língua, o primeiro passo geralmente é verificar se a frase segue as regras gramaticais
- Computacionalmente, tal verificação pode ser feita usando gramáticas "livres de contexto" G
  - $G = (N, \Sigma, P, S)$, onde N são símbolos não terminais; Σ ou alfabeto são símbolos terminais; P são regras ou produções; S é o símbolo inicial; o vocabulário de G é $V = N \cup \Sigma$
  - Gramática livre de contexto possui regras na forma $A \rightarrow \beta$, onde A é um símbolo não terminal, e β é uma sequência qualquer de V^*

Exemplo (I)

```
sentenca --> sujeito, predicado.

sujeito --> artigo, substantivo.

predicado --> verbo.
predicado --> verbo, objeto.

artigo --> [o].
artigo --> [a].

substantivo --> [pera].
substantivo --> [homem].

verbo --> [come].
verbo --> [canta].

objeto --> artigo, substantivo.
```

Exemplo (II)

- Para verificar se uma sentença é válida, é preciso tentar "encaixá-la" nas definições anteriores. Por exemplo: "o homem come a pera"



- Há um item principal, do qual emanam os demais: `sentenca`
- Há itens terminais, com definição dada em termos de exemplos concretos. Neste caso: `verbo`, `artigo` e `substantivo`

O problema da análise léxica

- Dada uma frase qualquer, determinar se ela se encaixa numa descrição de gramática como a exemplificada
- Construir uma árvore léxica para ela, nos moldes da figura anterior
- Um programa que faça isto é chamado de analisador léxico
- A formalização de regras sintáticas ilustradas aqui é chamada de DCG (*Definite Clause Grammar*)
- Prolog fornece mecanismos para lidar com estas gramáticas e executar análise léxica

Análise léxica em Prolog

- Ao realizar análise léxica em Prolog, é usual representar palavras por átomos e frase por listas de átomos. Para "o homem come a pera":

  `[o, homem, come, a, pera]`

- Predicado satisfeito quando a frase (em forma de lista) segue as regras gramaticais indicadas, e falha caso contrário (não confundir predicado do Prolog com predicado da sentença)

- Deste modo, haverá o predicado `sentenca(X)` que é satisfeito quando `X` for uma lista de palavras que formam uma sentença válida

- Predicados (de Prolog) para validar outros itens da frase: `sujeito(X)`, `predicado(X)`

Análise léxica em Prolog – uso de append (I)

```
sentenca(X) :-
    append(Y, Z, X), sujeito(Y), predicado(Z).

sujeito(X) :-
    append(Y, Z, X), artigo(Y), substantivo(Z).

predicado(X) :- verbo(X).
predicado(X) :-
    append(Y, Z, X), verbo(Y), objeto(Z).

artigo([o]).
artigo([a]).

substantivo([pera]).
substantivo([homem]).

verbo([come]).
verbo([canta]).

objeto(X) :-
    append(Y, Z, X), artigo(Y), substantivo(Z).
```

Análise léxica em Prolog – uso de append (II)

- Exemplo de pergunta com resposta correta

```
?- sentenca([o, homem, come, a, pera]).
true
```

- Exemplo de pergunta com resposta incorreta

```
?- sentenca([homem, pera, come, a, o]).
false
```

- Apesar de correto, tal esquema não é eficiente, pois `append` testa todas as combinações de quebra da frase

- Pode ser contornado com uso de acumuladores (predicados agora com dois parâmetros)

Análise léxica em Prolog – uso de acumuladores (I)

- Exemplo: `sujeito(X, Y)` é satisfeito quando existe um sujeito no início da lista `X` e o resto da lista, após o sujeito, é `Y`

- Para ajudar a entender, algumas perguntas respondidas afirmativamente:

```
?- sujeito([o, homem, come, a pera], [come, a, pera]).
?- sujeito([a, pera, canta], [canta]).
?- sujeito([o, homem, come, a pera], X).
?- sujeito([a, pera, canta], X).
```

  Nas duas últimas, a variável `X` fica instanciada com o que sobrar da lista após um sujeito

- Modificação das definições dos predicados para o novo significado:

```
sujeito(X, Y) :- artigo(X, Z), substantivo(Z, Y).
```

Análise léxica em Prolog – uso de acumuladores (II)

```
sentenca(S0, S) :- sujeito(S0, S1), predicado(S1, S).

sujeito(S0, S) :- artigo(S0, S1), substantivo(S1, S).

predicado(S0, S) :- verbo(S0, S).
predicado(S0, S) :- verbo(S0, S1), objeto(S1, S).

artigo([o|S], S).
artigo([a|S], S).

substantivo([pera|S], S).
substantivo([homem|S], S).

verbo([come|S], S).
verbo([canta|S], S).

objeto(S0, S) :- artigo(S0, S1), substantivo(S1, S).
```

Notação para regras gramaticais

- SWI-Prolog (como outras implementações) oferece uma maneira conveniente de escrever a gramática usando o funtor `-->`, exatamente como foi apresentado no início
- Prolog traduz esta notação em cláusulas da maneira eficiente (com acumulador)
- Exemplos de perguntas:

```
?- sentenca([o, homem, come, a, pera], [ ]).
true

?- sujeito([o, homem, canta], X).
X = [canta]
```

Exemplo (I)

```
sentenca --> sujeito, predicado.

sujeito --> artigo, substantivo.

predicado --> verbo.
predicado --> verbo, objeto.

artigo --> [o].
artigo --> [a].

substantivo --> [pera].
substantivo --> [homem].

verbo --> [come].
verbo --> [canta].

objeto --> artigo, substantivo.
```

Argumentos adicionais (I)

- Argumentos adicionais podem ser utilizados para propósitos específicos
- Por exemplo, tratar a concordância em número, entre sujeito e verbo
- Sentenças como "os homens come a pera" ou então "o homem comem a pera" não são corretas, mas seriam aceitas pela extensão do artigo "os", substantivo "homens" e verbo "comem" na gramática
- Solução: argumento adicional para indicar se a frase está no singular ou no plural

Argumentos adicionais (II)

```
sentenca --> sentenca(X).
sentenca(X) --> sujeito(X), predicado(X).

sujeito(X) --> artigo(X), substantivo(X).

predicado(X) --> verbo(X).
predicado(X) --> verbo(X), objeto(_).

artigo(singular) --> [o].
artigo(singular) --> [a].
artigo(plural) --> [os].
artigo(plural) --> [as].

substantivo(singular) --> [pera].
substantivo(singular) --> [homem].
substantivo(plural) --> [peras].
substantivo(plural) --> [homens].

verbo(singular) --> [come].
verbo(singular) --> [canta].
verbo(plural) --> [comem].
verbo(plural) --> [cantam].

objeto(X) --> artigo(X), substantivo(X).
```

Argumentos adicionais (III)

- Exemplos de perguntas

```
?- sentenca(X, [os, homens, comem, as, peras], [ ]).
X = plural

?- sentenca(X, [o, homens, come, as, pera], [ ]).
false
```

- Na segunda cláusula para predicado, a concordância se dá entre predicado e verbo, mas o objeto não precisa concordar com este verbo, visto que está ligado ao sujeito. Frases como:

```
?- sentenca(X, [o, homem, come, as, peras], [ ]).
X = singular
```

também são corretamente aceitas e classificadas

Exercícios

1. O que ocorre ao se efetuar a seguinte pergunta?

   ```
   sentenca(_, [os, homens, comem, os, peras], []).
   ```

   Altere o código DCG em Prolog para, além da concordância de número, verificar a concordância de gênero (masculino e feminino).

2. Considerando a gramática DCG abaixo:

   ```
   s --> sa, sb.
   sa --> [a].
   sa --> [a, a], sa.
   sb --> [b].
   sb --> [b], sb.
   ```

   Escreva um funtor que adicione argumentos apropriados a símbolos não terminais `s`, `sa` e `sb`, de modo que se possa calcular a diferença entre o número de ocorrências de símbolos "`s`" e de "`b`" na sentença. Por exemplo (5 `a`'s menos 2 `b`'s):

   ```
   ?- s( D, [a,a,a,a,a,b,b], [ ]).
   D = 3
   ```

Exercícios (cont...)

3. Considerando a gramática DCG abaixo:

```
s(N) --> n(N).
s(S) --> n(N), s(S0), { S is N + S0}.
n(1) --> [one].
n(2) --> [two].
n(3) --> [three].
```

Quais as respostas de Prolog para as seguintes questões?

- `?- s( Sa, [three], []).`
- `?- s( Sb, [ one, two, three], []).`
- `?- s( 3, L, []).`

Exercícios (cont...)

4. Considerando a gramática DCG abaixo:

```
s --> w.
s --> w, [and], s.
p --> s.
p --> s, [times], p.
w --> [one].
w --> [two].
w --> [three].
```

Quais as respostas de Prolog para as seguintes questões?

- `?- s( [ one, and, three], []).`
- `?- s( [one, two, three], []).`
- `?- s( [X, Y, three], []).`
- `?- p([ one, P, two, Q, three], []).`