PROLOG 6

Baseado no material Prolog, de Silvio Lago.

Base de Dados Dinâmica Base de dados dinâmica é um recurso bastante útil para implementar lógica não-monotônica e programas que adquirem conhecimento dinamicamente.

6.1 Manipulação da base de dados dinâmica

De acordo com o modelo relacional, uma base de dados é a especificação de um conjunto de relações. Neste sentido, um programa em Prolog é uma base de dados em que a especificação das relações é parcialmente explícita (fatos) e parcialmente implícita (regras).

```
Programa 6.1: Jogadores e esportes.
joga(pelé,futebol).
joga(guga,tênis).
esporte(X) :- joga(_,X).
```

Para listar as cláusulas da base de dados, usamos o predicado listing. Por exemplo, supondo que o Programa 6.1 tenha sido carregado na base, podemos listar as cláusulas do predicado joga através da seguinte consulta:

```
?- listing(joga).
joga(pelé,futebol).
joga(guga,tênis).
```

Para adicionar uma nova cláusula à base de dados podemos usar o predicado asserta ou assertz. A consulta a seguir mostra a diferença entre eles:

```
?- assertz(joga(oscar,basquete)).
Yes
?- asserta(joga(hortência,basquete)).
Yes
?- listing(joga).
joga(hortência,basquete).
joga(pelé,futebol).
joga(guga,tênis).
joga(oscar,basquete).
Yes
```

Para remover uma cláusula da base de dados, usamos o predicado retract. Esse predicado recebe uma estrutura como argumento e remove da base uma ou mais cláusulas (por retrocesso) que unificam com essa estrutura:

```
?- retract(joga(X,basquete)).
X = hortência ;
X = oscar ;
No
?- listing(joga).
joga(pelé, futebol).
joga(guga, tênis).
Yes
```

6.2 Aprendizagem por memorização

Grande parte do poder da linguagem Prolog vem da habilidade que os programas têm de se modificar a si próprios. Esta habilidade possibilita, por exemplo, a criação de programas capazes de aprender por memorização.

```
Programa 6.2: Onde estou?
:- dynamic estou/1. %declara modif. dinâmica
estou(paulista).
ando(Destino) :-
 retract(estou(Origem)),
 asserta(estou(Destino)),
 format('Ando da ~w até a ~w',[Origem,Destino]).
```

Veja como o Programa 6.2 funciona:

```
?- estou(Onde).
Onde = paulista
Yes
?- ando(augusta).
Ando da paulista até a augusta
Yes
?- estou(Onde).
Onde = augusta
Yes
```

Para satisfazer o objetivo ando(augusta), o sistema precisa remover o fato estou(paulista) e adicionar o fato estou(augusta). Assim, quando a primeira consulta é feita novamente, o sistema encontra uma nova resposta.

6.3 Atualização da base de dados em disco

Como podemos observar, o Programa 6.2 implementa um tipo de raciocínio não-monotônico; já que as conclusões obtidas por ele mudam à medida que novos fatos são conhecidos. Porém, como as alterações causadas pela execução dos predicados asserta e retract são efetuadas apenas na memória, da próxima vez que o programa for carregado, a base de dados estará inalterada e o programa terá "esquecido" que andou da Paulista até a Augusta. Para salvar em disco as alterações realizadas numa base de dados, podemos usar os predicados tell e told.

```
Programa 6.3: Salvando a base de dados em disco.
salva(Predicado,Arquivo) :-
   tell(Arquivo),
   listing(Predicado),
   told.
```

Para recuperar uma base salva em disco, usamos o predicado consult.

6.4 Um exemplo completo: memorização de capitais

Como exemplo de aprendizagem por memorização, vamos apresentar um programa capaz de memorizar as capitais dos estados. Esse programa será composto por dois arquivos: 1. geo.pl: contendo as cláusulas responsáveis pela iteração com o usuário; 2. geo.bd: contendo as cláusulas da base de dados dinâmica. No início da execução, o programa geo.pl carregará na memória as cláusulas existentes no arquivo geo.dat. No final, essas cláusulas serão salvas em disco, de modo que ela sejam preservadas para a próxima execução.

```
Pgm 6.4: Um programa que memoriza as capitais.
:- dynamic capital/2.
geo :- carrega('geo.bd'),
   format('~n*** Memoriza capitais ***~n~n'),
   repeat,
   pergunta(E),
   responde(E),
   continua(R),
   R = n,
   !,
   salva(capital,'geo.bd').

carrega(A) :-
  exists_file(A),
  consult(A)
  ;
  true.
pergunta(E) :-
  format('~nQual o estado cuja capital deseja?'),
  gets(E).
responde(E) :-
  capital(C, E),
  !,
  format('A capital de ~w é ~w.~n',[E,C]).
responde(E) :-
  format('Não sei. Qual a capital de ~w? ',[E]),
  gets(C),
  asserta(capital(C,E)).
continua(R) :-
  format('~nContinua? [s/n] '),
  get_char(R),
  get_char('\n').
gets(S) :-
  read_line_to_codes(user,C),
  name(S,C).
salva(P,A) :-
  tell(A),
  listing(P),
  told.
```

Na primeira vez que o Programa 6.4 for executado, a base de dados geo.bd ainda não existirá e, portanto, ele não saberá nenhuma capital. Entretanto, à medida que o usuário for interagindo com o programa, o conhecimento do programa a respeito de capitais vai aumentado. Quando a execução terminar, a base de dados será salva em disco e, portanto, na próxima vez que o programa for executado, ele se "lembrará" de todas as capitais que aprendeu. O Programa 6.4 faz uso de uma série de predicados extra-lógicos primitivos que são dependentes do interpretador. Maiores detalhes sobre esses predicados podem ser obtido no manual on-line do SWI-Prolog. Por exemplo, para obter esclarecimentos sobre o predicado name, faça a seguinte consulta: `?- help(name).`

☞ Para você fazer

6.1. Supondo que a base de dados esteja inicialmente vazia, indique qual será o seu conteúdo após terem sido executadas as seguintes consultas.

```
?- asserta( metal(ferro) ).
?- assertz( metal(cobre) ).
?- asserta( metal(ouro) ).
?- assertz( metal(zinco) ).
?- retract( metal(X) ).
```

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

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

6.3. 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.

6.4. 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
```

6.5. 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
```

.