Introdução a Prolog

Bases de dados dinâmicas

Predicados extras para manipulação de listas

- flatten(L1,L2) – remove todos os [ ] extras de L1, devolvendo o resultado em L2. Assim cria uma lista plana.

  Ex.: flatten([[1],[2,3]],X) => X = [1,2,3]

- list_to_set_(L1,L2) – remove elementos repetidos de L1, devolvendo em L2.

  Ex.: list_to_set_ ([1,2,1,3],X) => X = [1,2,3]

Programas ou base de dados?

bagof(X,P,L) – Lista L com todos objetos X q satisfazem P

setof(X,P,L) – Lista L com todos objetos X q satisfazem P, sendo L ordenada e sem itens duplicados

```
classe(a, vog). classe(b, con). classe(c, con). classe(d, con).
classe(e, vog). ...
```

```
?- bagof(Letra, classe(Letra, con), Consoantes).
=> [ b, c, d, f, ..., z ]
?- bagof(Letra, classe(Letra, Classe), Letras).
=> Classe = con, Letras = [b, c, d]
=> Classe = vog, Letras = [a, e]
```

Programas ou base de dados?

findall(X,P,L) – semelhante ao bagof, mas sem classificar em tipos diferentes para P (se não houver nenhum X => L = [ ])

```
classe(a, vog). classe(b, con). classe(c, con). classe(d, con).
classe(e, vog). ...
```

```
teste(L) :- findall(Letra, classe(Letra, vog), L).
?- teste(Vogais).
=> Vogais = [ a, e, i, o , u]

teste2(L) :- findall(Letra, classe(Letra, con), L).
?- teste2(Consoantes).
=> Consoantes = [ b, c, d, f, ...]
```

ENTRADA E SAÍDA

Leitura/escrita:
Caracter: get/1, get0/1 e put/1
String/termos: read/1 e write/1

```
cubo :-   % digitar . após o valor, e fim para finalizar
        write('Próximo valor: '), read(X), processa(X).
processa(fim) :- !.
processa(N) :-
        C is N*N*N, write('O cubo de '), write(N), write(‘ é '),
        write(C), nl, cubo.
```

?- cubo.

Base de dados dinâmica

- asserta(Clausula)

— armazena de modo permanente a cláusula passada como parâmetro no **início** da lista de cláusulas associadas ao predicado.

- assertz(Clausula)

— armazena de modo permanente a cláusula passada como parâmetro no **fim** da lista de cláusulas associadas ao predicado.

- assert(Clausula)

— armazena de modo permanente a cláusula passada como parâmetro no **ponto atual** da lista de cláusulas associadas ao predicado.

Base de dados dinâmica

- retract(Clausula)

— remove **uma** cláusula da base de fatos. As cláusulas a serem removidas devem ser declaradas como dinâmicas.

- abolish(Functor/Aridade)

— remove **todas** as cláusulas do predicado definido pelo functor e pela aridade. A declaração do predicado como sendo do tipo dinâmico é juntamente removida.

Base de dados dinâmica

Em prolog os programas tem habilidade de modificar-se a si próprios – Aprendizagem por memorização.

Para tal, usa-se o predicado sempre no início do programa:

:- dynamic(Functor/Aridade).

— é uma assertiva, declara que o predicado Functor/Aridade pode ser modificado durante a execução do programa. É a chave para a fazer uma base de dados dinâmica.

Sem este predicado não há possibilidade de modificação da base de dados.

Base de dados dinâmica

Ex. do SWI-Prolog – jogadores e esportes

```
:- dynamic joga/2.
joga(pele, futebol).
joga(guga, tenis).
esporte(X):- joga(_,X).
```

Adicionando manualmente novas cláusulas:

?- assert**z**(joga(oscar, basquete)). => true

?- assert**a**(joga(hortencia, basquete)). => true

?- **listing**(joga).

joga(hortencia, basquete).
joga(pele, futebol).
joga(guga, tenis).
joga(oscar, basquete).
true

Base de dados dinâmica

Ex. do SWI-Prolog – jogadores e esportes

Removendo manualmente cláusulas:

?- retract(joga(X, basquete)).
X = hortencia
X = oscar
false

?- listing(joga).
joga(pele, futebol).
joga(guga, tenis).
true

Base de dados dinâmica

Ex. do SWI-Prolog.

```
:- dynamic estou/1.

estou(paulista).
ando(Destino):-
        retract(estou(Origem)),
        asserta(estou(Destino)),
        format('Ando da ~w até a ~w', [Origem, Destino]).
```

_____

Obs.: e como seria com mais pessoas sendo controladas ?

Base de dados dinâmica

Ex. do SWI-Prolog - Gera a tabuada.

```
:- dynamic produto/3.
tabMult :-
        L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        membro(X, L), membro(Y, L),
        Z is X*Y, assertz(produto(X, Y, Z)),
        write(produto(X, Y, Z)),write('.'), nl, fail.
tabMult.
membro(X, [X | _]).
membro(X, [_ | C]) :- membro(X, C).
```

___

Alguns interpretadores Prolog já tem o predicado member.

Base de dados dinâmica

O programa anterior implementa um raciocínio *não-monotônico*, pois as conclusões mudam à medida que se conhecem novos fatos – mas isso ocorre apenas em memória.

Para **salvar em disco** as alterações numa base de dados usa-se os predicados:

- **tell**(fonte): abre a fonte de saída (arquivo)
- **told** : fecha a fonte

Para recuperar uma base de dados a partir de um arquivo:

- **consult**(arquivo)
- **see**(arquivo)

A entrada/saída padrão é **user.** Usa-se estes predicados para alteração desse fluxo quando necessário (see(user), see(arq), tell(user), tell(arq)).

Base de dados dinâmica

Ex. do SWI-Prolog - Gera a tabuada e salva em arquivo.

```
:- dynamic produto/3.
tabMult :-
        L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        membro(X, L), membro(Y, L),
        Z is X*Y, assertz(produto(X, Y, Z)),
        write(produto(X, Y, Z)),write('.'), nl, fail.
tabMult.
grava:- tabMult, read(Arq), tell(Arq), listing(produto), told,
tell(user).
carrega:- read(Arq), exists_file(Arq), % se existir arquivo
  consult(Arq), listing(produto). % faz consulta e mostra predic
membro(X, [X | _]).
membro(X, [_ | C]) :- membro(X, C).
```

BD dinâmica - memorização de capitais

```
:- dynamic capital/2.

geo:- carrega('C:/Users/.../prolog/geo.bd'), format('~n*** Memoriza capitais ***~n~n'),
   repeat, pergunta(E), responde(E), continua(R), R = 2, % ate dizer nao !!
   !, salva(capital,'C:/.../prolog/geo.bd').

carrega(Arq):- exists_file(Arq), % se existir consulta
   consult(Arq);
   true.  % senao nao faz nada (mas nao falha p/continuar programa)

pergunta(E):- format('~nQual o estado cuja capital você quer saber? '), read(E).

responde(E) :- capital(C, E), !, format('A capital de ~w é ~w.~n',[E,C]).

responde(E) :- format('Também não sei. Qual é a capital de ~w? ',[E]), read(C), asserta(capital(C,E)).

continua(R) :- format('~nContinua? [1 sim/2 nao] '), read(R).

salva(Predicado,Arq) :-  tell(Arq), listing(Predicado), told.
```