

Programação Declarativa

Paradigmas de Programação Avançados

Programação com conjuntos

Problemas

Formulação

Uso

Programação com Conjuntos

Situação: querer saber **todas as soluções** de qualquer coisa.

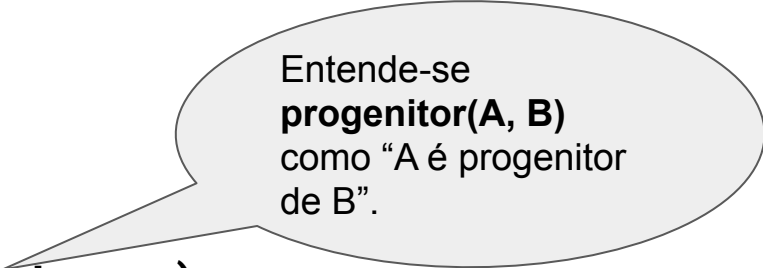
Recordando a base de dados familiar, com algumas definições...

Predicados “base de dados”

Todas as cláusulas têm **corpo vazio**.

Exemplo:

```
progenitor(francisco, tomas).  
progenitor(ana, tomas).  
progenitor(francisco, mariana).  
progenitor(ana, mariana).  
progenitor(antonio, francisco).  
progenitor(francisca, francisco).
```



Entende-se
progenitor(A, B)
como “A é progenitor
de B”.

Relações iteradas (ou recursivas)

Se quisermos falar dum pai, avô, bisavó, etc. estamos perante uma relação (“é **antepassado de**”) cuja definição é recursiva.

Dizemos que A é antepassado de B se:

A for progenitor de B, **OU:**

Se existir um X progenitor de A, que seja antepassado de B.

Traduzindo para Prolog:

```
antepassado(A,B) :- progenitor(A,B).
```

```
antepassado(A,B) :- progenitor(A,X), antepassado(X,B).
```

Hmmm... porque não este?

```
antepassado(A,B) :- progenitor(A,B).
```

```
antepassado(A,B) :- antepassado(A,X), progenitor(X,B).
```

Vamos ver todas os descendentes

Com a definição de “antepassado/2”, podemos fazer a query:

```
| ?- antepassado(antonio, Y).
```

Com o resultado:

```
Y = francisco ? ;  
Y = tomas ? ;  
Y = mariana ? ;
```

Se quisermos ter todos estes valores guardados simultâneamente, não conseguimos. Como fazer?

Sugestão: o `retract/1` é capaz de, sendo usado repetidamente, dar os vários valores dum predicado de base de dados, sem ter de retroceder (*cada uso retira um valor*).

Colecionar todas as soluções

Podemos usar o predicado `findall/3`, que tem a estrutura:

```
findall(PADRÃO, GERADOR, SOLUÇÕES)
```

Por exemplo:

```
| ?- findall(X, antepassado(antonio, X), Xs).  
Xs = [francisco,tomas,mariana]  
yes  
| ?-
```

Exemplo do mapa

Temos:

```
e(lisboa,santarem).  
e(santarem,coimbra).  
e(santarem,caldas).  
e(caldas,lisboa).  
e(coimbra,porto).  
e(lisboa,evora).  
e(evora,beja).  
e(lisboa,faro).  
e(beja,faro).
```

```
c(X) :- e(X, _).  
c(X) :- e(_, X).
```


Todas as soluções

```
| ?- findall(C, c(C), Cs).
```

```
Cs =
```

```
[lisboa,santarem,santarem,caldas,coimbra,lisboa,evora,lisboa,beja,santarem,coimbra,caldas,lisboa,porto,evora,beja,faro,faro]
```

Temos repetições.

Podemos em vez de **findall/3**, usar **setof/3**:

```
| ?- setof(C, c(C), Cs).
```

```
Cs =
```

```
[beja,caldas,coimbra,evora,faro,lisboa,porto,santarem]
```

Resumo

| ?- setof(K=V, e(K, V), KVs).

KVs =

[beja=faro, caldas=lisboa, coimbra=porto, evora=beja, lisboa=evora, lisboa=faro, lisboa=santarem, santarem=caldas, santarem=coimbra]

Temos 3 predicados:

- findall(PAT, GEN, SET)
- bagof(PAT, GEN, SET)
- setof(PAT, GEN, SET)

PAT pode incluir variáveis “existencialmente quantificadas”, que não vão ser usadas no resultado.

As diferenças

Primeira diferença: o que acontece à variáveis livres no goal, que não sejam colecionadas?

- No caso do **findall/3** temos todas as variáveis livres como sendo existencialmente quantificadas (i.e. não interessa o seu valor, desde que haja pelo menos uma)
- No caso do **bagof/3** e **setof/3**, as variáveis livres são transmitidas para fora do goal de conjunto. Funciona um pouco como um “group by” do SQL...

Segunda diferença: o que fazer com duplicados?

- No caso do **findall/3** e **bagof/3**, pode haver duplicados e a ordem é aquela induzida pela ordem pela qual as soluções individuais são produzidas.
- No caso do **setof/3**, a coleção vê-se ordenada e os duplicados removidos.

exemplo

```
| ?- findall(X, filho(X, Y), XX).  
XX = [tomas,tomas,mariana,mariana,francisco,francisco]
```

```
| ?- bagof(X, filho(X, Y), XX).  
XX = [tomas,mariana]  
Y = ana ? ;  
XX = [francisco]  
Y = antonio ? ;  
XX = [francisco]  
Y = francisca ? ;  
XX = [tomas,mariana]  
Y = francisco
```

```
| ?- setof(X, filho(X, Y), XX).  
XX = [mariana,tomas]  
Y = ana ? ;  
XX = [francisco]  
Y = antonio ? ;  
XX = [francisco]  
Y = francisca ? ;  
XX = [mariana,tomas]  
Y = francisco
```

```
| ?-
```

exemplo

```
| ?- findall(X, filho(X, Y), XX).
```

```
XX = [tomas,tomas,mariana,mariana,francisco,francisco]
```

yes

```
| ?- bagof(X, Y^filho(X, Y), XX).
```

```
XX = [tomas,tomas,mariana,mariana,francisco,francisco]
```

yes

```
| ?- setof(X, Y^filho(X, Y), XX).
```

```
XX = [francisco,mariana,tomas]
```

yes

```
| ?-
```

Não- Determinismo e Base de Dados

Assert/retract com backtracking

A BD pode interagir com a execução Prolog

Por exemplo, suponhamos que queremos coleccionar as soluções dum goal:

```
collect(G, GG) :-  
    retractall(gg(_)),  
    collect1(G, GG).
```

```
collect1(G, _) :- G, assertz(gg(G)), fail.  
collect1(_, GG) :- collect2([], GG).
```

```
collect2(L, GG) :- retract(gg(G)), !, collect2([G|L], GG).  
collect2(GG, GG).
```

exemplo

```
| ?- antepassado(X, tomas).
```

```
X = francisco ? a
```

```
X = ana
```

```
X = antonio
```

```
X = francisca
```

```
no
```

```
| ?- collect(antepassado(X, tomas), G).
```

```
G =
```

```
[antepassado(francisca,tomas),antepassado(antonio,tomas),  
antepassado(ana,tomas),antepassado(francisco,tomas)]
```

```
yes
```


O collect/2 e os predicados de conjunto

Os predicados de “conjunto” (findall, bagof, setof) podem ser implementados como o collect/2.

A diferença é que são mais gerais pois separam o goal que produz soluções do padrão que se recolhe.

Podíamos definir collect/2 em termos do findall, assim:

```
collect(G, GG) :-  
    findall(G, G, GG).
```

GG!