INFf3
Programmation logique

Cours 4 : Technique de programmation (suite)

Benoît Lemaire

Université Grenoble Alpes
L2 - MIASHS
Grenoble – France

Ces diapos se sont inspirées de celles créées par Jean-Michel Adam

Exemple de programmation avec accumulateur

?- supprimePairesIdentiques([[4,5],[3,0],[2,2],[7,4],[1,1]],R).
R = [[4, 5], [3, 0], [7, 4]].

```
% SANS ACCUMULATEUR
supprimePairesIdentiques([],[]).
supprimePairesIdentiques([[A,A]|L],R):-!,supprimePairesIdentiques(L,R).
supprimePairesIdentiques([X|L],[X|R]):-supprimePairesIdentiques(L,R).

% AVEC ACCUMULATEUR
supprimePairesIdentiques2(L,R) :- supprimePairesIdentiques2(L,[],R).
supprimePairesIdentiques2([],Acc,Acc).
supprimePairesIdentiques2([[A,A]|L],Acc,R):- !,supprimePairesIdentiques2(L,Acc,R).
supprimePairesIdentiques2([X|L],Acc,R):- append(Acc,[X],NewAcc),
                                         supprimePairesIdentiques2(L,NewAcc,R).
```

Création dynamique de faits/règles

- **assert/1 permet d'ajouter des faits ou des règles.**

- **Exemple :**

```
assert(etoile(soleil)).
assert(planete(X):-astre(X),etoile(E),satellite(X,E)).
```

Création dynamique de faits/règles

- **retract/1 permet de retirer des faits ou des règles.**

- **Exemple :**

```
meilleurScore(marie,2556).


...
retract(meilleurScore(_,_)),assert(meilleurScore(Joueur,
Score)),...
```

Récupération de toutes les solutions

- **findall(Var,Requete,Solutions)**

- **Exemple :**

```
?-findall(X,planete(X),L).
L=[terre,venus]

?-findall(X,(between(0,9,X),X mod 3 =:= 0),L).
L=[0,3,6,9]
```

Les intégrammes

- Les **intégrammes** appelés parfois logigrammes sont un type de casse-tête logique.
- On donne un certain nombre d'indices, desquels il faudra déduire l'intégralité des relations entre tous les éléments.

Exemple d’intégramme simple

Max, Eric et Luc habitent chacun une maison différente. Ils possèdent chacun un animal domestique distinct.

On sait que :

- Max a un chat.
- Eric n’habite pas en pavillon.
- Luc habite un studio, il n’a pas le cheval.

La problématique à résoudre est :

- Qui habite le château et qui a le poisson ?

Question : comment représenter le problème en Prolog ?

Modélisation : les faits

```
% les maisons :
maison(studio).
maison(pavillon).
maison(château).
% les animaux :
animal(chat).
animal(cheval).
animal(poisson).
```

Modélisation : les règles

```
% le prédicat habitation représente la relation
% entre une personne, sa maison et son animal :
% Max a un chat :
habitation(max,M,chat) :- maison(M).
% Eric n'habite pas en pavillon :
habitation(eric,M,A) :- maison(M), M\==pavillon, animal(A).
% Luc habite un studio, il n'a pas le cheval :
habitation(luc,studio,A):-animal(A),A\==cheval.
```

Résolution du problème : prédicat résoudre

```
% le prédicat resoudre décrit l'ensemble du problème à résoudre
% avec ses 4 éléments inconnus  et affiche la solution :
resoudre :-
    habitation(max,MM,chat),
    habitation(eric,ME,AE),
    habitation(luc,studio,AL),
    MM\==studio, MM\==ME, ME\==studio,
    AE\==chat, AE\==AL, AL\==chat,
    writef("max %t chat\n",[MM]),
    writef("eric %t %t\n",[ME,AE]),
    writef("luc studio %t\n",[AL]).
```

ou is_set([MM,ME,studio])

ou is_set([chat,AE,AL])

Interrogeons Prolog

```
?- resoudre.
max pavillon chat
eric chateau cheval
luc studio poisson
true ;
false.
```

On souhaite maintenant résoudre l’intégramme suivant

Dans une rue, 3 maisons voisines sont de couleurs différentes : rouge, bleue et verte. Des personnes de nationalités différentes vivent dans ces maisons et elles ont chacune un animal de compagnie différent.

Les données du problème sont :

- l'anglais vit dans la maison rouge.
- le jaguar est l'animal de l'espagnol.
- le japonais vit à droite de la maison du possesseur de l'escargot.
- le possesseur de l'escargot vit à gauche de la maison bleue.

La problématique à résoudre est :

- Qui possède le serpent ?

Modélisation du problème

- On va représenter la solution par une liste de 3 termes, chaque terme aura la forme suivante : `maison(couleur,nationalité,animal)`
- Le programme Prolog ne contient qu'un prédicat :

  `serpent(N, Rue) :- …`
  - `N` représente la nationalité du possesseur du serpent
  - `Rue` représente la solution complète, c'est-à-dire la liste de 3 termes `maison(couleur,nationalité,animal)`

Solution

```
serpent(N, Rue) :-
    % Rue est représentée par une liste de 3 maisons :
    length(Rue,3),
    % il y a une maison rouge, une maison bleue et une maison verte :
    member(maison(rouge,_,_),Rue),
    member(maison(bleue,_,_),Rue),
    member(maison(verte,_,_),Rue),
    % l'anglais vit dans la maison rouge :
    member(maison(rouge,anglais,_),Rue),
    % le jaguar est l'animal de l'espanol :
    member(maison(_,espagnol,jaguar),Rue),
    % le japonais vit à droite de la maison du possesseur de
l'escargot :
    nextto(maison(_,_,escargot),maison(_,japonais,_),Rue),
    % le possesseur de l'escargot vit à gauche de la maison bleue :
    nextto(maison(_,_,escargot),maison(bleue,_,_),Rue),
    % le troisième animal est un serpent :
    member(maison(_,N,serpent),Rue).
```

Interrogeons Prolog

```
?- serpent(N,Rue).
N = japonais,
Rue = [maison(rouge, anglais, escargot),
       maison(bleue, japonais, serpent),
       maison(verte, espagnol, jaguar)]
```

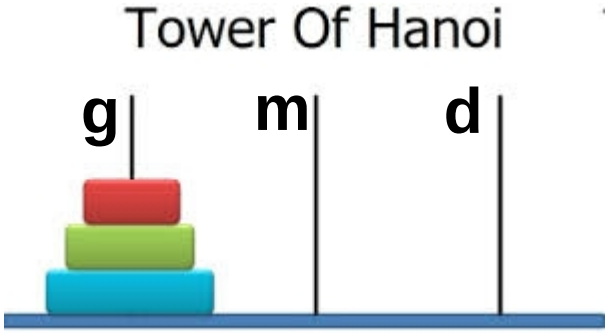
Utilisons cette modélisation pour le problème précédent

```
resoudre2(S):-
    % il y a 3 habitations: studio, chateau, pavillon (non
ordonnés)
    S = [habitation(_,studio,_), habitation(_,chateau,_),
     habitation(_,pavillon,_)],
    % il y a un poisson
    member(habitation(_,_,poisson),S),
    % il y a un cheval
    member(habitation(_,_,cheval),S),
    % Max a un chat
    member(habitation(max,_,chat),S),
    % Eric n'habite pas en pavillon :
    member(habitation(eric,HE,_),S), HE \== pavillon,
    % Luc habite un studio, il n'a pas le cheval :
    member(habitation(luc,studio,AL),S), AL \== cheval.
```

Interrogeons Prolog

```
?- resoudre2(S).
S = [habitation(luc, studio, poisson),
     habitation(eric, chateau, cheval),
     habitation(max, pavillon, chat)]
```

Les tours de Hanoi en Prolog



?- hanoi(4,g,m,d,L).

[[g,d],[g,m],[d,m],[g,d],[m,g],[m,d],[g,d],[g,m],[d,m],
[d,g],[m,g],[d,m],[g,d],[g,m],[d,m]]

```
hanoi(0,_,_,_,[]).
hanoi(N,A,B,C,L):-N1 is N-1,hanoi(N1,A,C,B,L1),
      hanoi(N1,C,B,A,L2), append(L1,[[A,B]|L2],L).
```