UNIVERSITÉ TOULOUSE III PAUL SABATIER

Rédigé avec LaTeX
Version du 24 février 2014

Programmation logique

Prolog

L3 Informatique
Semestre 6

Cours donné par Maurel et Arcangeli
Rédigé par Antoine de Roquemaurel

2014

Table des matières

1 Programmation par induction – unification

1.1 Unification

1.1.1 Exercice 1

```
genere(0,0).
genere(K,K) :- K > 0.
genere(K,W) :- K > 0, R is K-1,
        genere(R,W).
```

Listing 1.1 – Entiers de K à 0

Pour l'ordre croisant, il suffit d'inverser les clauses 2 et 3.

```
genere(0,0).
genere(K,W) :- K > 0, R is K-1, genere(R,W).
genere(K,K) :- K > 0.
```

Listing 1.2 – Entiers de 0 à K

1.1.2 Exercice 2

Une structure de données est nécessaire pour cet exercice, ici nous allons utiliser un tuple.

```
/*
 * Fonction basique : retourner une pièce
 * {pile,face} x {pile,face} -> {Vrai,Faux}
*/
opp(pile,face).
oppp(face,pile).

retourne(table(A,B,C),table(AA,BB,C)) :- opp(A,AA) ,opp(B,BB).
retourne(table(A,B,C),table(AA,B,C)) :- opp(A,AA) ,opp(C,CC).
retourne(table(A,B,C),table(A,BB,CC)) :- opp(B,BB) ,opp(C,CC).

jeu(T0) :- retourne(T0,T1), retourne(T1,T2), retourne(T2,table(X,X,X)),
      writeln(T0),writeln(T1),writeln(T2,writeln(table(X,X,X)).
```

Listing 1.3 – Exercice 2

1.2 Exercice 3 – Unification

```
/* 1- somme(L,S) est vrai si S est la somme des entiers d'une ligne L */
somme(lig(A,B,C),S) :- S is A+B+C.

/* 2- */
colo1(mat(lig(A,_,_),lig(B,_,_),lig(C,_,_))) :- lig(A,B,C).
colo2(mat(lig(_,A,_),lig(_,B,_),lig(_,C,_))) :- lig(A,B,C).
colo3(mat(lig(_,_,A),lig(_,_,_),lig(_,_,C))) :- lig(A,B,C).

diag1(mat(lig(A,_,_),lig(_,B,_),lig(_,_,C))) :- lig(A,B,C).
diag2(mat(lig(_,_,C),lig(_,B,_),lig(A,_,_))) :- lig(A,B,C).

/* l'unification ne permet pas de tester que deux paramètres sont différents :
 *  ils faut tous les testers 2 à 2
 */
tousdiff(mat(lig(A,B,C)), mat(lig(D,E,F)), mat(lig(G,H,I))) :- A \==B, A\==C, ↩
   A\==D.  /* .... */

magique(mat(lig(A,B,C), lig(D,E,F), lig(G,H,I)) :-
                        somme(L1,S),
                        somme(L2,S),
                        somme(L3,S),
                        colo1(mat(L1,L2,L3), C1), somme(C1,S),
                        colo2(mat(L1,L2,L3), C2), somme(C2,S),
                        colo3(mat(L1,L2,L3), C3), somme(C3,S),
                        diag1(mat(L1,L2,L3), D1), somme(D1,S),
                        diag2(mat(L1,L2,L3), D2), somme(D2,S),
                        tousdiff(mat(L1,L2,L3)).
```

Listing 1.4 – Exercice 3

1.3 Induction

1.3.1 Arbre binaire

```
/* somme des feuilles d'un arbre */
somme(feuille(X), X).
somme(noeud(G,D), S) :- somme(G, SG),
            somme(D, SD),
            S is SG + SD.

/* appartient à un arbre */
member(feuille(X), X).
member(noeud(G,_), N) :- member(G, N).
member(noeud(_,D), N) :- member(D, N).

/* hauteur de l'arbre */
max(X,Y,X) :- X >= Y.
max(X,Y,Y) :- Y > X.

hauteur(feuille(_), 0).
hauteur(noeud(G,D), H) :- hauteur(G, HG),
              hauteur(D, HD),
              max(HD,HD,MX),
              H is MX+1.

miroir(feuille(F), feuille(F)).
miroir(noeud(G,D), noeud(RD,RG)) :- miroir(noeud(G, RG)),
                    miroir(noeud(D,RD)).
```

Listing 1.5 – Arbre binaire

1.3.2 Exercice 4 – Arbre binaire étiqueté

```
/* appartient à un arbre */
estDansAbe(N, abe(_,N,_)).
estDansAbe(N, abe(G,_,_)) :- estDansAbe(N, G).
estDansAbe(N, abe(_,_,D)) :- estDansAbe(N, D).

/* appartient à un arbre ordonné */
estDansAbeo(N, abe(_,N,_)).
estDansAbeo(N, abe(G,_,_)) :- X < R, estDansAbeo(N, G).
estDansAbeo(N, abe(_,_,D)) :- X > R, estDansAbeo(N, D).

/* Insérer dans un arbre ordonné */
inserer(X, vide, abe(vide,X,vide)).
inserer(X, abe(G,R,D), abe(RG,R,D) ) :-  X < Y, inserer(X, G, RG).
inserer(X, abe(G,R,D), abe(G,R,RD) ) :-  X > Y, inserer(X, G, RD).
```

Listing 1.6 – Arbre binaire étiqueté

1.3.3 Listes

```
append([],L2,L2).
append([X|L],L2,[X|R]) :- append(L, L2, R).

replace(_, _, [], []).
replace(E1, E2, [X|L], [Y|R]) :- replace(E1, E2, L, R).
replace(E1, E2, [W|L], [X|R]) :- W /== X, replace(E1, E2, L, R).

prefixe([], _).
prefixe([X|L1], [Y|L2]) :- prefixe(L1, L2).

segment(S,L) :- prefixe(S,L).
segment(S,[_|L]) :- segment(S,L).

prefixe2(P, L) :- append(P,_,L).
segment2(S, L) :- append(_,K,L), append(S,_,K), S \== [].
```

Listing 1.7 – Méthodes sur les listes

```
decomposer([], _, [], []).
decomposer([X|L], P, [X|I], S) :- X =< P, decomposer(L, P, I, S).
decomposer([X|L], P, I, [X|S]) :- X > P, decomposer(L, P, I, S).

triPivot([], []).
triPivot([P|L], T) :- decomposer(L, P, I, S),
            triPivot(I, TI),
            triPivot(S, TS),
            append(TI, [P|TS],T).
```

Listing 1.8 – Tri pivot

```
inserer(X, [], [X]).
inserer(X, [Y|L], [X|[Y|L]]) :- X =< Y.
inserer(X, [Y|L], [Y|R]) :- X > Y, inserer(X, L, R).

triInsertion([], []).
triInsertion([X|L], T) :- triInsertion(L, TL),
              inserer(X, Tl, T).
```

Listing 1.9 – Tri par insertion

```
nuplet(0, []).
nuplet(N, [0|L]) :- M is N-1, nuplet(M, L).
nuplet(N, [1|L]) :- M is N-1, nuplet(M, L).

/* insérer dans toutes les positions*/
insertion(X, L, [X|L]).
insertion(X, [Y|L], [Y|R]) :- insertion(X,L,R).

permutation([], []).
permutation([X|L], P) :- permutation(L, PL), insertion(X, PL, P).
```

Listing 1.10 – Nuplet

A Liste des codes sources