Université libanaise
Faculté des sciences I

TP PROLOG – I2209

Fiche No2

Etudiant : Omar ElKhatib

Exercice 1

Définir les prédicats suivants :
Pgcd de deux entiers,

```
pgcd(A,0,A).
pgcd(A,B,R) :- A < B , R is A.
pgcd(A,B,R) :- A > B , B>0 , Rem is rem(A,B) , pgcd(B , Rem , R).
```

Fibonacci : retourne le terme d'ordre N de la suite de Fibonacci,

```
fib(0, 0).
fib(1, 1).
fib(N, R) :-
   N>1,
   N1 is N - 1,
   N2 is N - 2,
   fib(N1, R1),
   fib(N2, R2),
   R is R1+R2.
```

Somme_chiffres : retourne la somme des chiffres d'un nombre donné.

```
sumNum(A,R) :- sumNum(A,R,0).
sumNum(A , R , S) :- A < 1 , R is S.
sumNum(A,R,S) :- A>0,Div is div(A,10) , Rem is rem(A,10) , N is S+Rem,
                 sumNum(Div , R , N).
```

Exercice 2

Définir les prédicats suivants :

1. Concaténation de deux listes.

   ```
   concat(A,B,R) :- append(A,B,R).
   ```

2. Appartenance d'un élément à une liste :
   a. Au premier niveau
   b. A tous les niveaux

   ```
   find([H|T] , Target) :- find(T,Target); H = Target.
   ```

3. Nombre d'éléments d'une liste :

a. Au premier niveau
b. A tous les niveaux

```
count(List , R) :- count(List , 0 , R).
count([], N , R) :- R is N.
count([_|T], Count , R) :- N is Count+1 , count(T,N,R).
```

4. Nombre d'occurrences d'un élément dans une liste :
   a. Au premier niveau
   b. A tous les niveaux

```
appearence(List , Target , R) :- appearence(List, Target , [] , R).
appearence([], _ , Founds , R) :- length(Founds,R).
appearence([H|T] , Target , Founds , R) :-
  (  H = Target ->
  append(Founds,[Target],X),
  appearence(T,Target,X,R) ; appearence(T,Target,Founds,R)).
```

5. Suppression d'un élément dans une liste :
   a. Au premier niveau
   b. A tous les niveaux

```
supression(List , Target , R) :- supression(List , Target , [] , R).
supression([] , _ , FinalList , R) :- R = FinalList.
supression([H|T], Target , FinalList , R) :-
  (  H = Target ->  supression(T,Target,FinalList,R);
       append(FinalList , [H] , X),supression(T,Target,X,R)
  ).
```

6. Substitution d'un élément par un autre dans une liste

```
substitution(List , Target , With , R) :- substitution(List , Target , With , [] , R).
substitution([] , _ , _ , FinalList , R) :- R = FinalList.
substitution([H|T], Target , With , FinalList , R) :-
  (  H = Target ->  append(FinalList , [With] , X) , substitution(T,Target,With,X,R);
       append(FinalList , [H] , X),substitution(T,Target,With,X,R)
  ).
```

7. Inversion d'une liste

```
inverse(List , R) :- inverse(List,R,[]).
inverse([],X,X).
inverse([H|T], R , Acc) :- inverse(T , R , [H|Acc]).
```

8. Dernier élément d'une liste

```
last([H],H).
last([_|T] , R) :- last(T,R).
```

9. Linéarisation d'une liste.

¯_(ツ)_/¯

10. Les tours de Hanoi en renvoyant la liste des déplacements